

# Computer Application

Class XI



171018



**PUNJAB SCHOOL EDUCATION BOARD**

**Sahibzada Ajit Singh Nagar**

© Punjab Government

Edition 2023-24 ..... 1,000 copies

All rights, including those of translation, reproduction  
and annotation, etc., are reserved by the  
Punjab Government

### **Warning**

1. The Agency-holders shall not add any extra binding with a view to charge extra money for the binding. (Ref. Cl. No.7 of agreement with Agency-holders).
2. Printing, Publishing, Stocking, Holding or Selling etc., of spurious Text-books qua text-books printed and published by the Punjab School Education Board is a cognizable offence under Indian Penal Code.  
(The text-books of the Punjab School Education Board are printed on paper carrying 'watermark' of the Board.)

Price :

## Foreward

Punjab School Education Board, since its inception has been engaged in an endeavour to prepare text books on various subjects at school level. This textbook is one in the series and is designed to focus on student-centered learning for the students of subject Computer. The book has been developed as per recommendations of Punjab Curriculum Framework 2013 based on National Curriculum Framework 2005, according to which the school life of the students needs to be linked to the outside world. The book is a step forward to move the traditional book-based learning to activity-based learning.

The need for study of Computer as a subject is increasing day by day because in the age of scientific and technological advancement computer education improves the work efficiency in every field related to socio-economic life. With the advancement of communication and information technology, computerization is being done in every department. Computer Education has become mandatory to have access to information from various departments, e-governance and internet access in various aspects of modern life such as education, business, health, transportation and many more.

Keeping in view such requirements, Punjab School Education Board has implemented the guidelines of the Punjab Government to make the Computer Application subject as an optional subject for Class XI and XII. Every effort has been made to include requisite information according to level of class XI in this book. Hopefully this book will be useful for students and teachers.

Punjab School Education Board acknowledges the sincere efforts made by writers, translators and vetters for the preparation of this book. Board also welcomes comments and suggestions for any further improvement.

**Chairman**

Punjab School Education Board

## **Text-book Development Committee**

### **Writers :**

1. Mr. Vikas Kansal, Shaheed Udham Singh Govt. Sen. Sec. School (Girls), Sunam Udham Singh Wala, Sangrur
2. Mr. Sukhwinder Singh, Shaheed Udham Singh, Govt. Sen. Sec. School (Girls), Sunam Udham Singh Wala, Sangrur
3. Mrs. Bindu, Govt. Model Senior Secondary School, Phase 3 B-1, S.A.S. Nagar
4. Mrs. Meenu Malhotra, Govt. High Smart School Garangan, S.A.S. Nagar

### **Vetters :**

1. Mr. Vikas Kansal, Shaheed Udham Singh Govt. Sen. Sec. School (Girls), Sunam Udham Singh Wala, Sangrur
2. Mr. Sukhwinder Singh, Shaheed Udham Singh Govt. Sen. Sec. School (Girls), Sunam Udham Singh Wala, Sangrur
3. Mr. Gagandeep Singh, Govt. Model Senior Secondary School, Phase 3 B-1, S.A.S. Nagar
4. Mr. Inderjit Singh Govt. Senior Secondary School, Nandpur Kalour, Shri Fatehgarh Sahib

### **Co-ordinator :**

1. Mr. Manvinder Singh, Subject Expert (Computer), Punjab School Education Board, S.A.S. Nagar

### **Cover Title :**

1. Mr. Manjit Singh Dhillon, Artist, Punjab School Education Board, S.A.S. Nagar



## INDEX

Sr. No.	Description	Page No.
<b>Lesson 1 :</b>	<b>Fundamental of Information Technology</b>	<b>01-14</b>
1.1	Computer System	
1.2	Components of Computer System	
1.3	Software & its types	
1.4	Trends in IT	
<b>Lesson 2 :</b>	<b>Number System &amp; Logic Gates</b>	<b>15-37</b>
2.1	Introduction to Number system	
2.2	Types of Number system	
2.3	Conversion between Number Systems	
2.4	Logic Gates	
2.5	Proof of NAND and NOR as Universal Gates	
2.6	Demorgan Law	
<b>Lesson 3 :</b>	<b>Introduction to Programming Languages</b>	<b>38-51</b>
3.1	Introduction	
3.2	Concept of Program, Programming and Programmar	
3.3	Programming Language	
3.4	Language Translator	
3.5	Types of Errors	
<b>Lesson 4 :</b>	<b>Introduction to OOP with JAVA</b>	<b>52-74</b>
4.1	Introduction to Object Oriented Programming	
4.2	Principles of Object-Oriented Programming-abstraction, encapsulation, inheritance and polymorphism	
4.3	Introduction to Java	
4.4	History of Java	
4.5	Features of Java	
4.6	Basic Structure of Java Program	
4.7	Creating an Executing a Simple Java Program	
4.8	Basic Elements for Java Programming : Character set, tokens, comments	

<b>Lesson 5 :</b>	<b>Data Types and Operations in Java</b>	<b>75-92</b>
5.1	Data Types	
5.2	Variables	
5.3	Operators	
5.4	Expressions	
5.5	Precedence of Operators	
5.6	Types Conversation	
<b>Lesson 6 :</b>	<b>Control Statements in Java</b>	<b>93-110</b>
6.1	Introduction	
6.2	Branching Statements : if else, switch case	
6.3	Looping Statements : for, while, do while	
6.4	Jumping Statements : break, continue	
<b>Lesson 7 :</b>	<b>Classes and Objects</b>	<b>111-129</b>
7.1	Class	
7.2	Objects	
7.3	Static Members	
7.4	Constructors	
7.5	Overloading in Java (Methods and Constructors)	
<b>Lesson 8 :</b>	<b>String and Wrapper Classes</b>	<b>130-143</b>
8.1	Strings	
8.2	Performing operation on Strings	
8.3	Wrapper Classes	
<b>Lesson 9 :</b>	<b>Inheritance</b>	<b>144-169</b>
9.1	Inheritance and its types	
9.2	Interfaces	
9.3	Use of super method	
9.4	Abstract Class and Methods	
9.5	Method Overriding	
9.6	Final Class, Method and Variables	



# Fundamentals of Information Technology



## OBJECTIVES OF THIS CHAPTER

- 1.1 Computer System
- 1.2 Components of Computer System
- 1.3 Software & its types
- 1.4 Trends in IT

## Introduction

When we talk about IT, first of all what comes to our mind are computer, network and internet. IT or Information Technology is the study or use of electronic equipments, such as computers, for collecting, storing, processing, and exchanging electronic data or information.

### 1.1 Computer System

The most important part of IT is computer system. **A Computer is an electronic device which accepts, store and processes data into useful information.**



Figure 1.1 : Computer System



### 1.1.1 Main functions of a Computer System:

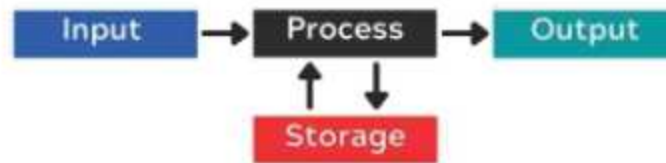


Figure 1.2 : Four main processes of Computer System

1. **INPUT:** The data and raw facts, which we want the system to process, we enter into the computer is called the input. It accepts data.
2. **PROCESSING:** It processes data to provide information.
3. **STORE:** It stores data and information.
4. **OUTPUT:** Processed data given us as a result/outcome. It gives output.

### 1.1.2 Characteristics of a Computer System :

The following are the characteristics of the computer system:

1. **Speed:** A computer works with very high speed as compared to normal human beings while performing mathematical calculations. Computers can process an instruction in a fraction of a second. The speed of a computer is measured in microseconds, milliseconds, nanoseconds and even in Picoseconds.
2. **Accuracy:** Computers perform each and every calculation with very high and same accuracy. Errors may occur due to inconsistent or inaccurate data provided to it.
3. **Diligence:** A computer can perform millions of tasks or calculations with the same consistency and accuracy. It is free from monotony, tiredness, lack of concentration etc. and hence can work for hours together without creating any error.
4. **Versatility:** Versatility is one of the most wonderful things about the computer. Versatility refers to the capability of a computer to perform different kinds of works with same accuracy and efficiency. In a moment it can do any one operation and next moment it can perform any other operation. A computer is capable of performing almost any task according to given instructions.
5. **Reliability:** It performs all operations with 100 % accuracy and reliability. A computer is reliable as it gives consistent result for similar set of data i.e., if we give same set of input any number of times, we will get the same result. Reliability can only be affected by errors made by human beings.
6. **Automation:** Computer performs all the tasks automatically i.e. it can perform tasks without manual intervention.
7. **Memory:** A computer has built-in memory called primary memory and secondary storage devices such as HDD, CDs, pen drives, etc., where it can store data and instruction with a lot of volume and very high efficiency.

### 1.1.3 Limitations or Drawbacks of Computer:

1. **No I.Q.:** Computer is not a magical device. It has no intelligence or thinking power. It performs tasks which a man can do but the main difference is that computer can perform operations with very high speed and reliable accuracy.
2. **No Feeling:** As we all know computer is only a machine; it has no feelings like human beings. It has no brain for thinking as man. It's a man who made this machine and memory with his brain but he couldn't make heart.
3. Problems can occur due to system breakdown.

### 1.2 Components of Computer System:

A computer system is not a single device. Items which together form a computer system are known as computer components. Computer system is made of several components which have specific functions and features to support the entire system.

The main three components of computer system are – Input Unit, Output Unit, and CPU – Central Processing Unit.

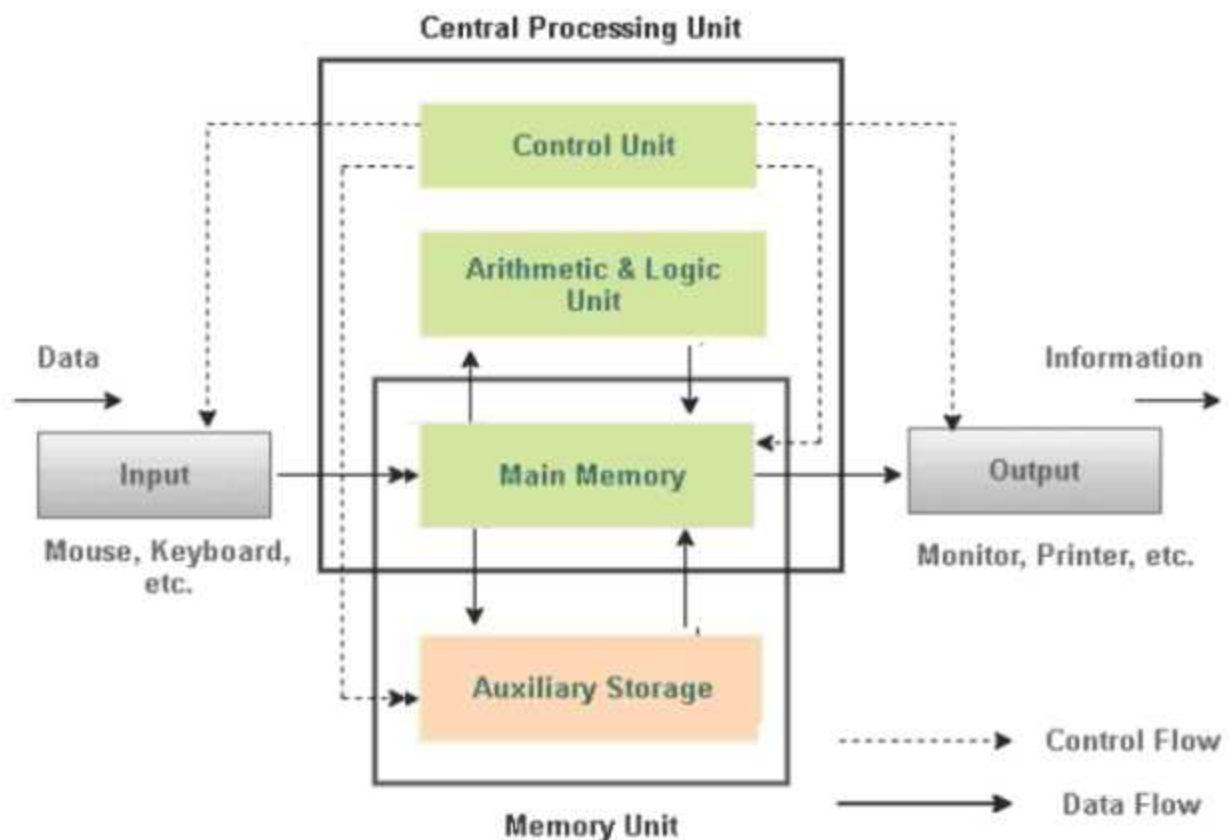


Figure : 1.3 Components of Computer System



**1.2.1 Input Unit**–The devices that allow the user to give instruction or provide data to the computer system are known as Input Unit. It enables the user to communicate with the computer.

- Following are the commonly used Input devices:

1. **Keyboard:** The **keyboard** is one of the most common devices for inputting data or instructions to a computer. The keys allow access to a range of alphanumeric characters, special characters and a number of commands by using the function keys. It also provides shortcuts to specific tasks. Sometimes it is not possible or appropriate to enter data using a keyboard and so other devices are available.



Figure : 1.4 (Keyboard)

2. **Mouse:** A mouse is a pointing and clicking device, it allows the user to communicate with the computer by placing a cursor at a particular point on the screen. The computer screen tracks the movement of the mouse and displays its location on the screen as a cursor.



Figure : 1.5 (Mouse)

3. **Joystick:** A joystick is an input device that can be used for controlling the movement of the cursor or a pointer. The pointer/cursor movement is controlled by a lever on the joystick. The input device is mostly used for gaming applications and, sometimes, in graphics applications. A joystick also can be helpful as an input device for people with movement disabilities.



Figure : 1.6 (Joystick)

4. **Scanner:** Scanner converts a document or photograph into a digital file that a computer can read or display. Scanners have been developed for other uses such as biometrics. Examples include iris and fingerprint scanning, where the iris or fingerprint is illuminated and photographed.



Figure : 1.7 (Scanner)

5. **Touch displays:** A touch display/screen is a display device that allows users to interact with a computer using their finger or stylus. They're a useful alternative to a mouse or keyboard for navigating a GUI (graphical user interface). Touch screens are used on various devices, such as computer and laptop displays, smart phones, tablets etc.



Figure : 1.8 (Touch display)

6. **Graphics tablet:** This can be used in a similar way to the keyboard by tapping on **icons** (a symbol or image on a computer screen). It is also useful for entering data, such as handwriting or drawings of objects which cannot be captured easily in other ways.



Figure : 1.9 (Graphic tablet)

7. **Microphone:** Microphone record instructions from the user, or other sounds that the user wishes to collect. For those who cannot use a traditional keyboard or mouse, this is a very useful device for inputting data.



Figure : 1.10 (Microphone)

8. **Barcode reader:** Barcode reader may be handheld or static, and is used to capture information from a barcode by using a light source, scanner and decoder. They reduce the errors often introduced by manual input.



Figure : 1.11 (Barcode reader)

9. **Bio-metric:** A biometric device is a security identification and authentication device. Such devices use automated methods of verifying or recognizing the identity of a living person based on a physiological or behavioral characteristic. These characteristics include fingerprints, facial images, iris and voice recognition.



Figure : 1.12 (Bio-metric)

- 1.2.2 **Output devices** –The devices which allow the computer system to provide information, data or instructions to the user are known as Output Unit. It enables the computer to communicate with the user.

- Following are the commonly used Output devices:

1. **Printers:** The purpose of a **printer** is to produce hard copy of documents, data, information, images and photographs. Different types and sizes of printers are available such as inkjet, laser, photo printers, LCD and LED, line printers, thermal printers and 3D printers.



Figure : 1.13 (Printer)

2. **Plotters:** Plotters are machines used in areas such as computer-aided design (CAD) for large and accurate drawings from architectural designs to the latest space probe. Originally plotters used a pen to draw lines on paper, and later multiple pens brought colour into the process. While pens have been almost completely replaced by inkjet and laser print systems, it does help us to understand the process to remember that plotters are based on pens. The two main types are the flatbed and drum plotters.



Figure : 1.14 (Plotter)



3. **Speakers and headphones:** These provide the user with audio output. The speakers convert the electronic signals in the sound card to sound waves that can be heard by the user. Speakers broadcast the audio to everyone. Headphones ensure that privacy is maintained; however, the volume of the output can result in temporary or permanent hearing loss if it is set too high for any period of time.



Figure : 1.15 (Speaker)

4. **Visual display unit (VDU) or screen:** This can be treated both input device as well as an output device, because it displays requests or information from the computer. It does, however, help the user who is inputting data or instructions because it allows the user to check their work visually, allowing them time to correct errors before confirming their decision.



Figure : 1.16 (VDU)

**1.2.3 System Unit** – The CPU also known as **Central Processing Unit**, keeps the track of all the hardware activities required to receive instructions and data, performs all arithmetic and logical operations and presents output as the result. It also manages the various storage devices, both internal and external and records where all programs and instructions are located for future retrieval.

**1.2.3.1 Components of CPU** – CPU is further divided into control unit, arithmetical and logical unit and memory unit. Each unit has its own special feature to assist the whole system.

- a. **Control unit** – The control unit generates the appropriate timing and control signals to all the operations involved with a computer. The main function of a control unit is to fetch the data from the main memory, determine the devices and the operations involved with it, and produce control signals to execute the operations. The flow of data between the processor, memory, and other peripherals is controlled using the timing signals of the control unit. It ensures that the instructions required to operate the computer are retrieved and interpreted in the correct sequence.
- b. **ALU**–ALU is a main component of the central processing unit, which stands for **arithmetic logic unit** and performs arithmetic and logic operations. Here the computer undertakes the mathematical or logical operations required to complete the instruction. The data is placed in a special register called the accumulator and arithmetic (additions and subtractions) and logical

operations such as 'and', 'or' and 'not' are carried out.

- c. **Registers:** - Registers are the smallest, fastest. They are used directly by the CPU. A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters)

**1.2.3.2 Memory Unit** - Unit where all the programs and data are stored is known as Memory unit. It is further divided into two different categories: primary storage and secondary storage.

- **Primary storage** - Computer Primary storage can further be divided into following categories:
  - **Random Access Memory(RAM)** -It is fast and larger in size, but slower than those within the CPU. It holds almost all other data and instructions that the computer may need for any programs that is currently running on the computer. As the programs' requirements change or programs are replaced by others, so-the memory is over-written with the new data and instructions.
  - **Read Only Memory (ROM)** - This contains pre-programmed permanent instructions such as the Basic Input/output (BIOS) program, which, when you switch on the computer, checks that all the connected devices are in place, checks that they are working and loads the operating system into the computer. The BIOS is an integral part of the computer when it is built, unlike operating systems, which can be added later.
- **Secondary storage** - A secondary storage device is any non-volatile storage device that is internal or external to the computer. It is attached to the computer system to store programs and data permanently for the purpose of retrieving them for future use such as Floppy disk, Hard disk, CD, DVD, Optical disks, USB (Universal serial bus) Flash drive etc.

**1.3 Software & its types** - Software is the programs or instructions which tell the computer what to do or make the hardware run. Software comprises the entire set of programs, procedures, and routines associated with the operation of a computer system. A set of programs intended to provide users with a set of interrelated functionalities is known as a **software package**.

Computer software can be classified in many ways:

**1.3.1 Types of software based on their development** -

- One of the ways to categorize software is by considering how it was developed and considering following main points:
  1. Is it copyright protected or available to all?
  2. Can the user make changes to the source code?
  3. Is it developed for a particular organization or for general use by many individuals and organizations?





These differences are described below:

- **Open source software** – Open source means that the source code, produced by the programmer, is made available to anyone who wishes to debug, improve or modify it. Some producers of the original source code may restrict access and modification for all or some of the code but the principle is access for all.
- **Closed source software** – Closed source is guarded and can be protected by property right and copyright, with the full force of the law being imposed upon those who break the restrictions. When purchasing such software, you are purchasing only a right to use it but you cannot treat it as you own it.
- **Off the shelf and bespoke software** – A business or an organization wanting to replace or introduce applications such as financial planning or inventory systems may buy a general package and then use the parts they need, or if their work is different, they can have a package written especially for them. The first option is often referred to as “**off the shelf**”, as the package is pre-designed, and came in a box that any one literally took off the shelf. One may be able to make small adjustments, such as to color or layout, or design of forms and reports. One may have to change some of one's systems and work practices to fit the software.
- **Bespoke software**, like a bespoke suit or shirt, it is made to one's exact measurements. These are expensive and take time for the designer to identify one's needs. The time it takes to provide a working program also needs to be considered. Who owns this software possesses the intellectual property rights.
- **Shareware** – If the user is not sure what they need then they can try shareware for a period of time, often 30–60 days, before they have to purchase the product. At the end of the trial period they either pay the purchase price or the software locks them out. All intellectual property rights and copyright remain the property of the shareware author.
- **Freeware** – Freeware is distributed free of cost, although there may be restrictions on upgrades or no warranty. All rights remained with the writer or publisher. The range of freeware products run from small, special utility programs to well-known programs such as Acrobat Reader, MSN messenger which are available to all.
- **Embedded software** – This type of software may be embedded in a traditional PC or devices we do not consider to be computers, such as microwave ovens, GPS systems, smart watches and guided missile technology etc.



### 1.3.2 Types of Software based on their working:

The other way of classification of software depends upon its working. So the Computer software can be classified as **System software**, **Application software** and **Utility Software**.

- **System software**– System software refers to the programs required for the computer to operate and it provides a platform for application software. System software is a collection of one or more programs that controls and manages the overall operation and performance of a computer system. Without system software, computer hardware will not be able to perform its functions appropriately. It can be classified into the following categories: Operating System, Language Processors, Device Drivers, and Utility Software.
- **Operating System** – The operating system acts as an interface between the user and the hardware. An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs. **Operating system (OS)** manages all of the **software** and **hardware** on the computer. Most of the time, there are several different computer programs running at the same time, and they all need to access computer's **CPU**, **memory**, and **storage**. The operating system coordinates all of this to make sure each program gets what it needs. Operating systems usually come pre-loaded on any computer. Most people use the operating system that comes with their computer, but it's possible to upgrade or even change operating systems. The three most common operating systems for personal computers are **Microsoft Windows**, **Mac OS**, and **Linux**.
- **Language Processors** – A language processor is a software program designed for converting program code written in high level language into machine code. Language processors are found in languages such as FORTRAN and COBOL. There are three types of Language processors:
  - Assembler
  - Interpreter
  - Compiler
- **Device Drivers** – A device driver is a particular form of software program that allows interaction between system and hardware device or controls a specific hardware device such as printer attached to a computer. Device drivers are essential for a computer to work properly.
- **Utility software** – Computers working can be affected such as loss of memory, slowness at some point due to **viruses**. Utility programs are

small pieces of software that can help the computer perform more efficiently or effectively. Examples include backup facilities, which remind you to back up your data in case of hardware failure, loss or theft of the computer. The antivirus utility regularly scans the computer for evidence of malware activity or presence; it also checks for new threats and removes them to a 'quarantine location' where they are held until the user confirms that they are to be deleted.

- **Application software** – Application software is the software who helps the user to perform a specific task. Application software is designed to carry out tasks such as accounts management, text creation and editing, presentation preparation and design drawings etc. Examples of application software are web browser which helps user to access internet, Adobe Photoshop – used for editing and creating images and photos.

## 1.4 Trends in IT

With the advancement in technology and growth of information systems, the need for adoption of information technology in every sector is there. Information technology is used in context of computers and telecommunication equipment. IT Industry is considered as an emerging industry and all of us face several trends in this industry since its origin. New trends arise within this industry every year, and it becomes important for professionals to be familiar with these different trends. These trends include the study, design, development, application, implementation, support or management of computer-based information systems. Let's discuss some of these:

- 1.4.1 **GPS:** The Global Positioning System (GPS) is a U.S.-owned utility that provides users with positioning, navigation, and timing (PNT) services. This system consists of three segments: the space segment, the control segment, and the user segment.
- 1.4.2 **Bluetooth:** Bluetooth uses radio waves to transmit information between two devices directly. The radio waves used by Bluetooth are much weaker than those involved with Wi-Fi or cellular signals, two other common ways to connect devices.
- 1.4.3 **Wi-Fi:** Wi-Fi is the wireless technology used to connect computers, tablets, smartphones and other devices to the internet. Wi-Fi is the radio signal sent from a wireless router to a nearby device, which translates the signal into data you can see and use. The device transmits a radio signal back to the router, which connects to the internet by wire or cable.

A Wi-Fi network is simply an internet connection that's shared with multiple devices in a home or business via a wireless router. The router is connected



directly to your internet modem and acts as a hub to broadcast the internet signal to all your Wi-Fi enabled devices. This gives you flexibility to stay connected to the internet as long as you're within your network coverage area.

- 1.4.4 Cloud Computing:** One of the biggest trends is cloud computing. More and more industries are realizing that it is important for a company to have a designated place for all of their digital information and resources, and having a well-protected place that can take care of everything and keep the information safe. Cloud computing is for, who want to improve their work and make it more efficient in a digital space. Cloud migration has also proven to be incredibly beneficial for businesses that want to move in a digital direction and who want to maintain better records of their digital data.
- 1.4.5 Firewall:** Firewall is essentially the barrier that sits between a private internal network and the public Internet. Firewall is a network security device that monitors and filters incoming and outgoing network traffic based on security policies. Its main purpose is to allow non-threatening traffic in and to keep dangerous traffic out.
- 1.4.6 E-commerce:** E-commerce is the process of buying and selling of goods and services, or the transmitting of funds or data, over an electronic network, primarily the internet. **E-commerce** refers to all aspects of operating a business online. E-commerce has completely changed how the market operates, making it easier to discover and purchase products from anywhere in the world. In fact, e-commerce has created a worldwide market which was simply not possible in the traditional retail market.
- 1.4.7 Online shopping:** Online shopping refers to the online selling and purchasing of goods and services. Online shopping is an e-commerce activity which involves purchasing items on a seller's website via credit or debit card, and having the item delivered to your home. Online shopping also involves searching for items online and by conducting online research.
- 1.4.8 Mobile Apps:** Mobile applications have only grown in popularity over the past few years, and this year, they are surfacing in bigger and better ways. Brands and industries all over the world are trying to find ways in which one can improve their work through the use of mobile apps and through the implementation of new resources that can make working on the go more efficient.
- 1.4.9 Big Data Analytics:** Big data analytics is a trend that has grown over the past few years, and this is something that is now being implemented in almost every kind of industry that makes use of large-scale production processes and manufacturing and supply. Big data analytics allows to process information in a better manner and enables user to reach a much better understanding of the areas they need to develop.
- 1.4.10 Automation:** Automation is one trend that has largely hit the manufacturing and

production units and is something that is estimated to only grow more in the coming years. Automation has also enabled processes to work at a faster pace and enables companies to reach their goals in a much more efficient manner.

**1.4.11 Artificial Intelligence:** While automation is growing, artificial intelligence is also growing. Smart Technology which includes smart machines that use artificial intelligence or automation is on the rise, even in small-scale units and smaller implementations. Homes are now becoming smarter with the numerous integrated devices that work to make our lives easier, as a result of smart technology being used. Simple tools such as Alexa have become an essential part of homes, and are increasing. Virtual Reality such as in the gaming industry has already started to become popular due to new technology, which improves the gaming experience for the user. Augmented reality is another approach to 'artificial experiences' that individuals are now being given access to. Augmented reality is a technology that superimposes a computer-generated image on a user's view of the real world, thus providing a composite view.

**1.4.12 Growth of IoT Networks:** The Internet Of Things is a concept that all digital devices are connected by a single medium through which one would be able to control everything within their homes. This concept can be described as – “things” – that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet.



## Points to Remember

1. A Computer is an electronic device which accepts, store and processes data into useful information.
2. Computer system is made of several components which have specific functions and features to support the entire system.
3. The main three components of computer system are – Input Unit, Output Unit, and CPU – Central Processing Unit.
4. The devices that allow the user to give instruction or provide data to the computer system are known as **Input Unit**.
5. The devices which allow the computer system to provide information, data or instructions to the user are known as **Output Unit**.
6. **CPU** is further divided into **control unit, arithmetical and logical unit** and **memory unit**.
7. One of the ways to categories software is by considering how it was developed



- and the other way of classification of software depends upon its working.
8. System software refers to the programs required for the computer to operate and it provides a platform for application software.
  9. Application software is the software who helps the user to perform a specific task.
  10. The trends in IT include the study, design, development, application, implementation, support or management of computer-based information systems.

## Exercise

### Q1. Multiple choice Questions:

1. A computer works with very high \_\_\_\_\_ as compared to normal human beings while performing mathematical calculations.  
1.5    Memory                      b. speed                      c. processing                      d. control
2. A joystick is an \_\_\_\_\_ pointing device.  
a. Inline                      b. insecure                      c. input                      d. output
3. Scanner converts a document or photograph into a \_\_\_\_\_ file  
a. Proper                      b. coded                      c. digital                      d. useless
4. A \_\_\_\_\_ is a display device that allows users to interact with a computer using their finger or stylus.  
a. touch screen                      b. joystick                      c. keyboard                      d. printer
5. The purpose of a printer is to produce \_\_\_\_\_ of documents  
a. Hard copy                      b. soft copy                      c. both a and b                      d. none of these

### Q2. Fill in the blanks:

1. The data we enter into the computer is called the \_\_\_\_\_.
2. Computers can process an instruction in a \_\_\_\_\_ of a second.
3. Speakers and headphones provide the user with \_\_\_\_\_ output.
4. A secondary storage device is any \_\_\_\_\_ storage device
5. The operating system acts as an \_\_\_\_\_ between the user and the hardware

### Q3. Full Forms:

1. Wi-Fi
2. CPU
3. ROM
4. RAM
5. GPS



6. OS
7. GUI
8. CU
9. ALU
10. IoT
11. USB
12. BIOS
13. VDU

**Q4. Short answer type questions:**

1. What is computer? What are its four main functions?
2. What are the Limitations or Drawbacks of Computer?
3. What is Primary storage? Explain.
4. What is Secondary storage? Explain.
5. What is Utility software? Explain.
6. Define GPS.
7. Explain Bluetooth.
8. What is Wi-Fi?
9. Describe the Cloud Computing.
10. What is Artificial Intelligence?
11. What is e-commerce and online shopping?
12. What is firewall?

**Q5. Long answer type questions:**

1. Explain the Characteristics of a Computer System.
2. Explain the Components of Computer System.
3. What is Input Unit? Describe any four input devices.
4. What is Output device? Describe any four output devices.
5. What is Software & How software can be categorized? Explain.
6. What are trends in IT? Explain any 5.
7. What is system software? Explain.
8. Draw Block diagram of computer system and explain its various parts.



# Number System & Logic Gates



## OBJECTIVES OF THIS CHAPTER

- 2.1 Introduction to Number system
- 2.2 Types of Number system
- 2.3 Conversion between Number Systems
- 2.4 Logic Gates
- 2.5 Proof of NAND and NOR as Universal Gates
- 2.6 Demorgan's Law

### 2.1 INTRODUCTION TO NUMBER SYSTEM

A computer can only understand numbers whatever we write whether letters or words, the computer translates them into numbers first. A computer can understand the number system by the position a digit occupies. The technique to represent and work with numbers is called number system. Number systems are used to count or measure something. In the daily life, we use decimal number system. There are two basic types of number systems:

- \* Non-positional number systems.
- \* Positional number system.

- a. **Non-Positional Number Systems** : In these number systems, the value of a numeral symbol does not depend on its position. A non-positional number system uses a limited number of symbols in which each symbol has a value. However, the position a symbol occupies in the number normally bears no relation to its value, the value of each symbol is fixed. The roman number system is a good example of a non-positional number system. This number system has a set of symbols  $S=\{I, V, X, L, C, D, M\}$ .

Values of symbols in the Roman number system

Symbol	I	V	X	L	C	D	M
Value	1	5	10	50	100	500	1000

Figure : 2.1

**b. Positional number systems :** In these number systems, the value of a numeral symbol depends on its position. In a positional number system, the value of each digits is determined by which place it appears in the full number. The lowest place value is the **rightmost position**, and each successive position to the left has a **higher place value**. Every positional number system has a base or radix value. Example of positional number system is: Decimal Number System.

**Decimal Position Notation**

← Continue	10	10	10	10	10	Radix
	4	3	2	1	0	Position
	$10^4$	$10^3$	$10^2$	$10^1$	$10^0$	Calculation
	10,000	1,000	100	10	1	Positional Value

Figure : 2.2

Other examples are Binary number system (with Radix 2), Octal Number System (with Radix 8), Hexadecimal number system (with Radix 16).

## 2.2 TYPES OF NUMBER SYSTEM

There are four types of Number Systems that a computer supports. They are:-

1. Binary Number System
2. Octal Number System
3. Decimal Number System
4. Hexadecimal Number System

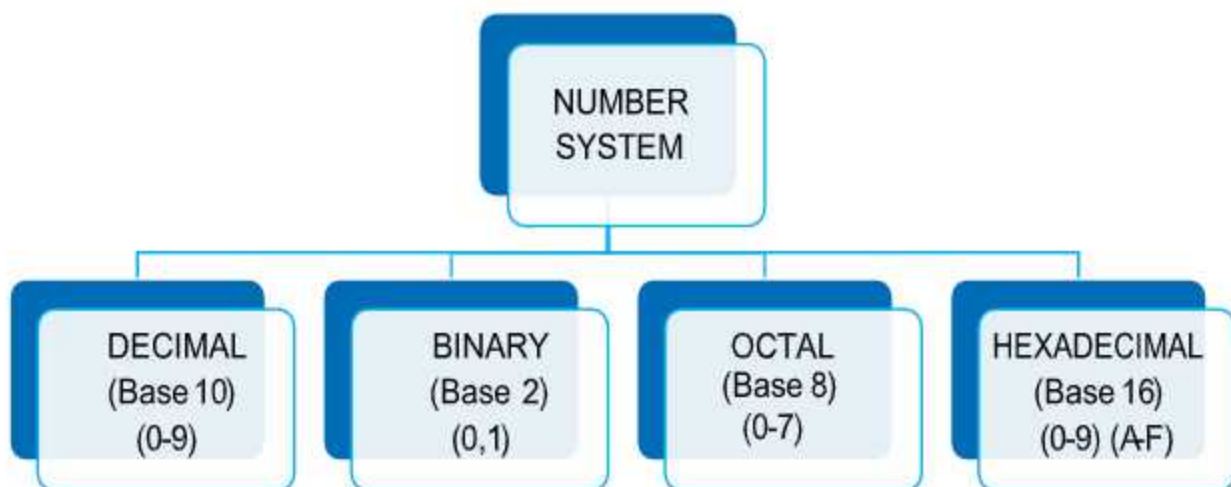


Figure : 2.3

S.No.	NumberSystem	BASE	Description	Example
1	Decimal Number System	10	• Digits used: 0 to 9	27
2	Binary Number System	2	• Digits used: 0, 1	11011
3	Octal Number System	8	• Digits used: 0 to 7	33
4	Hexadecimal Number System	16	• Digits used: 0 to 9 • Letters used: A- F	1A

Figure : 2.4

**BASE/RADIX:** The total number of digits in number system is called base or radix.

### Examples of Numbers in different number system

DECIMAL	BINARY	OCTAL	HEXADECIMAL
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

Figure : 2.5



### 2.2.1 DECIMAL NUMBER SYSTEM :

A number system which consists of 10 different symbols 0,1,2,3,4,5,6,7,8 and 9 is called Decimal number system. The base or radix value of this number system is 10. Any number in other number systems can be converted back into decimal number system. This number system is used by the human beings. This is a Positional number system.

Every digits have its own place value according to their position. When we move from right to left, the number value increases by 10. The position on the left of the decimal is for ones,tens, hundreds, thousands, and so on units.

For example – The decimal number 8432 consists of 2 at the unit position, 3 in the tens position, 4 in the hundreds position, and 8 in the thousands position.

$$(8 \times 10^3) + (4 \times 10^2) + (3 \times 10^1) + (2 \times 10^0)$$

$$(8 \times 1000) + (4 \times 100) + (3 \times 10) + (2 \times 1)$$

$$8000 + 400 + 30 + 2$$

$$8432$$

### 2.2.2 BINARY NUMBER SYSTEM

A number system which consists of two different symbols 0 and 1 is called Binary number system. The base or radix value of this number system is 2. A decimal number can be converted into binary number system. This number system is used by the computer systems. This is a Positional number system.

Bit stands for Binary Digits. These digits are 0 and 1. Binary Number system uses these digits. A digital computer stores everything in the form of binary digits. Combination of 8 bits is called a Byte. In other words, a byte is equal to 8 bits. It is the smallest unit of memory.

The most basic unit of storage in a device is represented by a bit. A computer uses a bit to show any sort of information.

Only these two symbols represent every number. 0 is for the lower rate while 1 is for the higher rate. The numbers are not as an individual unit here, but are made of groups of 1s and 0s. Each binary digit is a bit and the place values are ascending powers of two from left to right. The least significant bit is on the rightmost side while the most significant bit is on the leftmost side. For example – 11010

### 2.2.3 OCTAL NUMBER SYSTEM

Octal word comes from the latin word oct which means eight. It uses digits from 0 to 7. A



number system which consists of 8 different symbols 0,1,2,3,4,5,6 and 7 is called octal number system. The base or radix value of this number system is 8. A decimal no can be converted into octal number system. This number system is used by the computer systems. This is a Positional number system.

Formula for Octal Number System  $2^3=8$ . It uses 3 binary digits to represent octal number. The octal numeral system, or oct for short, is the base-8 number system, and uses the digits 0 to 7, that is to say 10 octal represents eight and 100 octal represents sixty-four.

In computing environments, it is commonly used as a shorter representation of binary numbers by grouping binary digits into three's. The chmod command in Linux or UNIX uses octal to assign file permissions.

#### 2.2.4 HEXADECIMAL NUMBER SYSTEM

Hexadecimal is divided into two words 'Hexa' and 'deci', where 'Hexa' means 6 and 'deci' means 10. The hexadecimal number system is described as a 16 digit number representation of numbers from 0 - 9 and digits from A - F. In other words, the first 9 numbers or digits are represented as numbers while the next 6 digits are represented as symbols from A - F. Hence, the 16 digits are 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. For example: 8A316, 7F16, are hexadecimal numbers.

A hexadecimal number system is also known as a positional number system as each digit has a weight of power 16.

#### 2.3 CONVERSION OF NUMBER SYSTEM

There are **three conversions** possible for binary number, i.e., binary to decimal, binary to octal, and binary to hexadecimal. In our previous section, we learned different types of number systems such as binary, decimal, octal, and hexadecimal.

As, we have four types of number systems so each one can be converted into the remaining three systems. There are the following conversions possible in Number System

1. Binary to other Number Systems.
2. Decimal to other Number Systems.
3. Octal to other Number Systems.
4. Hexadecimal to other Number Systems.

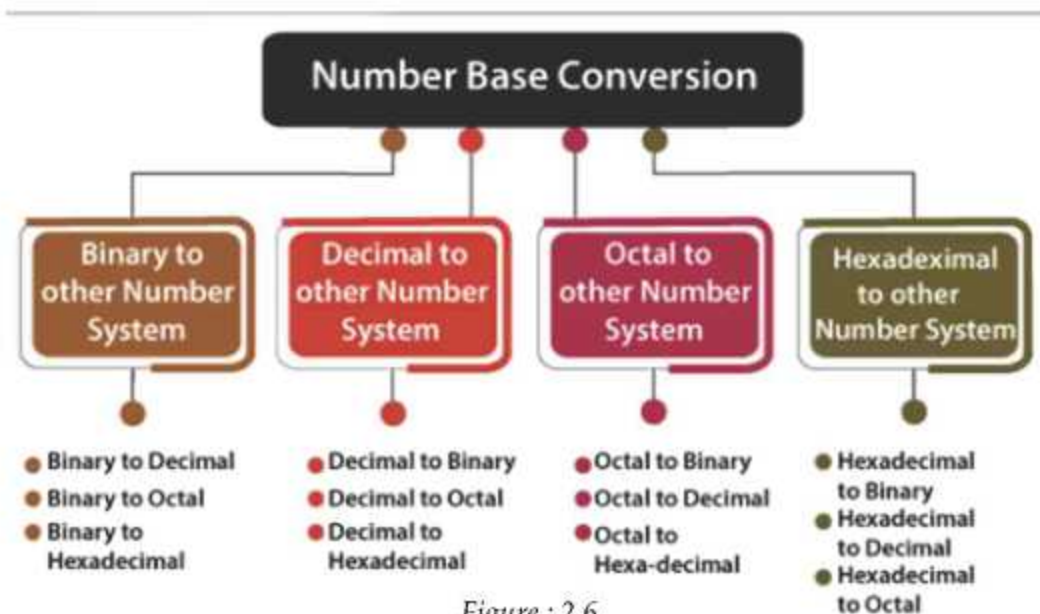


Figure : 2.6

### 1. Binary to other Number Systems

There are three conversions possible for binary number, i.e., binary to decimal, binary to octal, and binary to hexadecimal. The conversion process of a binary number to decimal differs from the remaining others.

#### (a) Binary to Decimal Conversion

In this process we start multiplying the bits of binary number with its corresponding positional weights. And lastly, we add all those products.

Let's take an example to understand how the conversion is done from binary to decimal.

**Example 1:**  $(1101.001)_2$

We multiplied each bit of  $(10110.001)_2$  with its respective positional weight, and last we add the products of all the bits with its weight.

$$\begin{aligned}
 (1101.001)_2 &= (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) \\
 &= (1 \times 8) + (1 \times 4) + (0 \times 2) + (1 \times 1) + \left(0 \times \frac{1}{2}\right) + \left(0 \times \frac{1}{4}\right) + \left(1 \times \frac{1}{8}\right) \\
 &= 8 + 4 + 0 + 1 + 0.125 \\
 &= (13.125)_{10}
 \end{aligned}$$

#### (b) Binary to Octal Conversion

The base number of binary is 2 and octal is 8. In a binary number, the pair of three bits is equal to one octal digit. There are only two steps to convert a binary number into an octal number which are as follows:

In the first step, we have to make the pairs of three bits on both sides of the binary point. If there will be one or two bits less in a pair of three bits pair, we add the required number of zeros on extreme sides.

In the second step, we write the octal digits corresponding to each pair.

**Example 1:  $(10101.0011)_2$**

1. Firstly, we make pairs of three bits on both sides of the binary point.

10    101.001    1

On the right side of the binary point, the last pair has only one bit. To make it a complete pair of three bits, we added two zeros on the extreme side and on the left side of the binary point, the last pair has two bits so we will add one zero on the extreme side.

010    101.001    100

2. Then, we wrote the octal digits, which correspond to each pair.

$$(10101.0011)_2 = (25.14)_8$$

**(c) Binary to Hexadecimal Conversion**

The base number of binary is 2 and hexadecimal is 16. In a binary number, the pair of four bits is equal to one hexadecimal digit. There are also only two steps to convert a binary number into a hexadecimal number.

In the first step, we have to make the pairs of four bits on both sides of the binary point. If there will be one, two, or three bits left in a pair of four bits pair, we add the required number of zeros on extreme sides.

In the second step, we write the hexadecimal digits corresponding to each pair.

**Example 1:  $(110101011.0011)_2$**

1. Firstly, we make pairs of four bits on both sides of the binary point.

1    1010    1011.0011

On the left side of the binary point, the first pair has one bit. To make it a complete pair of four bits, add three zeros on the extreme side.

0001    1010    1011.0011

2. Then, we write the hexadecimal digits, which correspond to each pair.

$$(110101011.0011)_2 = (1AB.3)_{16}$$

**2. Decimal to other Number System**

The decimal number can be an integer or floating-point integer. When the decimal number is a floating-point integer, then we convert both part (integer and fractional) of the decimal number in the isolated form. There are the following steps that are used to convert the decimal number into a similar number of any base 'r'.

In the first step, we perform the division operation on integer and successive part with base 'r'. We will list down all the remainders till the quotient is zero. Then we find out the remainders in reverse order for getting the integer part of the equivalent number of base 'r'. In this, the least and most significant digits are denoted by the first and the last remainders.



In the next step, the multiplication operation is done with base 'r' of the fractional and successive fraction. The carries are noted until the result is zero or when the required number of the equivalent digit is obtained. For getting the fractional part of the equivalent number of base 'r', the normal sequence of carrying is considered.

### (a) Decimal to Binary Conversion

For converting decimal to binary, there are two steps required to perform, which are as follows:

In the first step, we perform the division operation on the integer and the successive quotient with the base of binary (2).

Next, we perform the multiplication on the integer and the successive quotient with the base of binary (2).

**Example 1:**  $(68.25)_{10}$

#### Step 1:

Divide the number 68 and its successive quotients with base 2.

Radix	Number/Quotient	Remainder
2	68	0 (LSB)
2	34	0
2	17	1
2	8	0
2	4	0
2	2	0
2	1	1 (MSB)
	0	

$$(68)_{10} = (1000100)_2$$

Figure : 2.7

#### Step 2:

Now, perform the multiplication of 0.25 and successive fraction with base 2.

Operation	Result	carry
$0.25 \times 2$	0.50	0
$0.50 \times 2$	0	1

Figure : 2.8

$$(0.25)_{10} = (.01)_2$$

$$(68.25)_{10} = (1000100.01)_2$$

### (b) Decimal to Octal Conversion

For converting decimal to octal, there are two steps required to perform, which are as follows:

In the first step, we perform the division operation on the integer and the successive quotient with the base of octal(8).

Next, we perform the multiplication on the integer and the successive quotient with the base of octal(8).

#### Example 1: $(68.25)_{10}$

##### Step 1:

Divide the number 68 and its successive quotients with base 8.

Radix	Number quotient	Remainder
8	68	4 (LSB)
8	8	0
8	1	1 (MSB)
	0	

$$(68)_{10} = (104)_8$$

Figure : 2.9

##### Step 2:

Now perform the multiplication of 0.25 and successive fraction with base 8.

Operation	Result	carry
$0.25 \times 8$	0	2

Figure : 2.10

$$(0.25)_{10} = (.2)_8$$

So, the octal number of the decimal number 68.25 is **(104.2)<sub>8</sub>**.

### (c) Decimal to hexadecimal conversion

For converting decimal to hexadecimal, there are two steps required to perform, which are as follows:

In the first step, we perform the division operation on the integer and the successive quotient with the base of hexadecimal (16).

Next, we perform the multiplication on the integer and the successive quotient with the base of hexadecimal (16).

**Example 1: (68.25)<sub>10</sub>**

#### Step 1:

Divide the number 68 and its successive quotients with base 16.

Radix	Number/Quotient	Remainder
16	68	4 (LSB)
16	4	4 (MSB)
	0	

Figure : 2.11

$$(68)_{10} = (44)_{16}$$

#### Step 2:

Now perform the multiplication of 0.25 and successive fraction with base 16. Next we perform the multiplication on the integer and the successive quotient with the base of hexadecimal (16).

Operation	Result	Carry
$0.25 \times 16$	0	4

Figure : 2.12



$$(0.25)_{10} = (4)_{16}$$

So, the hexadecimal number of the decimal number  $(68.25)_{10}$  is  $(44.4)_{16}$ .

### 3. Octal to other Number System

Like binary and decimal, the octal number can also be converted into other number systems. The process of converting octal to decimal differs from the remaining one. Let's start understanding how conversion is done.

#### a. Octal to Decimal Conversion

The process of converting octal to decimal is the same as binary to decimal. The process starts from multiplying the digits of octal numbers with its corresponding positional weights. And lastly, we add all those products.

Let's take an example to understand how the conversion is done from octal to decimal.

**Example 1:  $(62.25)_8$**

##### Step 1:

We multiply each digit of **62.25** with its respective positional weight, and last we add the products of all the bits with its weight.

$$\begin{aligned} (62.25)_8 &= (6 \times 8^1) + (2 \times 8^0) + (2 \times 8^{-1}) + (5 \times 8^{-2}) \\ &= 48 + 2 + \left(2 \times \frac{1}{8}\right) + \left(5 \times \frac{1}{64}\right) \\ &= 48 + 2 + 0.25 + 0.078125 \\ &= 50.328125 \end{aligned}$$

So, the decimal number of the octal number  $(62.25)_8$  is  $(50.328125)_{10}$ .

#### b. Octal to Binary Conversion

The process of converting octal to binary is the reverse process of binary to octal. We write the three bits binary code of each octal number digit.

**Example 1:  $(62.25)_8$**

We write the three-bit binary digit for 6, 2, and 5.

$$(62.25)_8 = (110010.010101)_2$$

So, the binary number of the octal number  $(62.25)_8$  is  $(110010.010101)_2$ .

#### c. Octal to hexadecimal conversion

For converting octal to hexadecimal, there are two steps required to perform, which are as follows:

In the first step, we will find the binary equivalent of number **62.25**

Next, we have to make the pairs of four bits on both sides of the binary point. If there will be one, two, or three bits left in a pair of four bits pair, we add the required number of zeros on extreme sides and write the hexadecimal digits corresponding to each pair.

**Example 1:  $(62.25)_8$**

##### Step 1:

We write the three-bit binary digit for 6, 2, and 5.

$$(62.25)_8 = (110010.010101)_2$$

So, the binary number of the octal number 62.25 is  $(110010.010101)_2$

#### Step 2:

1. Now, we make pairs of four bits on both sides of the binary point.

11    0010.0101    01

On the left side of the binary point, the first pair has two digit, and on the right side, the last pair has also two-digit. To make them complete pairs of four bits, add zeros on extreme sides.

0011            0010.0101    0100

2. Now, we write the hexadecimal digits, which correspond to each pair.

$$(0011 \quad 0010.0101 \quad 0100)_2 = (32.54)_{16}$$

#### 4. Hexa-decimal to other Number System

Like binary, decimal, and octal, hexadecimal numbers can also be converted into other number systems. The process of converting hexadecimal to decimal differs from the remaining one. Let's start understanding how conversion is done.

##### (a) Hexa-decimal to Decimal Conversion

The process of converting hexadecimal to decimal is the same as binary to decimal. The process starts from multiplying the digits of hexadecimal numbers with its corresponding positional weights. And lastly, we add all those products.

Let's take an example to understand how the conversion is done from hexadecimal to decimal.

**Example 1:**  $(62A.25)_{16}$

#### Step 1:

We multiply each digit of **62A.25** with its respective positional weight, and last we add the products of all the bits with its weight.

$$\begin{aligned} (62A.25)_{16} &= (6 \times 16^2) + (2 \times 16^1) + (A \times 16^0) + (2 \times 16^{-1}) + (5 \times 16^{-2}) \\ &= (6 \times 256) + (2 \times 16) + (10 \times 1) + (2 \times 16^{-1}) + (5 \times 16^{-2}) \\ &= 1536 + 32 + 10 + \left(2 \times \frac{1}{16}\right) + \left(5 \times \frac{1}{256}\right) \\ &= 1578 + 0.125 + 0.125 \\ &= 1578.14453125 \end{aligned}$$

So, the decimal number of the hexadecimal number 62.25 is  $(1578.14453125)_{10}$

##### (b) Hexadecimal to Binary Conversion

The process of converting hexadecimal to binary is the reverse process of binary to hexadecimal. We write the four bits binary code of each hexadecimal number digit.

**Example 1:**  $(62A.25)_{16}$

We write the four-bit binary digit for , 6, A, 2, and 5.

$$(62A.25)_{16} = (0110\ 0010\ 1010.0010\ 0101)_2$$

So, the binary number of the hexadecimal number 62A.25 is

$$(0110\ 0010\ 1010.0010\ 0101)_2$$

### c. Hexadecimal to Octal Conversion

For converting hexadecimal to octal, there are two steps required to perform, which are as follows:

In the first step, we will find the binary equivalent of the hexadecimal number.

Next, we have to make the pairs of three bits on both sides of the binary point. If there will be one or two bits left in a pair of three bits pair, we add the required number of zeros on extreme sides and write the octal digits corresponding to each pair.

**Example 1:**  $(62A.25)_{16}$

#### Step 1:

We write the four-bit binary digit for 6, 2, A, and 5.

$$(62A.25)_{16} = (0110\ 0010\ 1010.0010\ 0101)_2$$

So, the binary number of hexadecimal number 62A.25 is

$$(0110\ 0010\ 1010.0010\ 0101)_2$$

#### Step 2:

Then, we make pairs of three bits on both sides of the binary point.

011 000 101 010.001 001 010

Then, we write the octal digit, which corresponds to each pair.

$$(0110\ 0010\ 1010.0010\ 0101)_2 = (3052.112)_8$$

So, the octal number of the hexadecimal number 62A.25 is **3052.112**

## 2.4 LOGIC GATES

Digital circuits are electronics that handle digital signals. These circuits work with the discrete states of analogue levels. Digital electronic circuits are usually made from logic gates.

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one outputs.

Logic gates are made up of diodes and transistors. It is used to allow and denied a digital signal. The relationship between the input and the output is based on a certain logic.

Common basic logic gates are AND gate, OR gate, and NOT gate, while NAND and NOR gates are called Universal Gates. These gates are explained below:



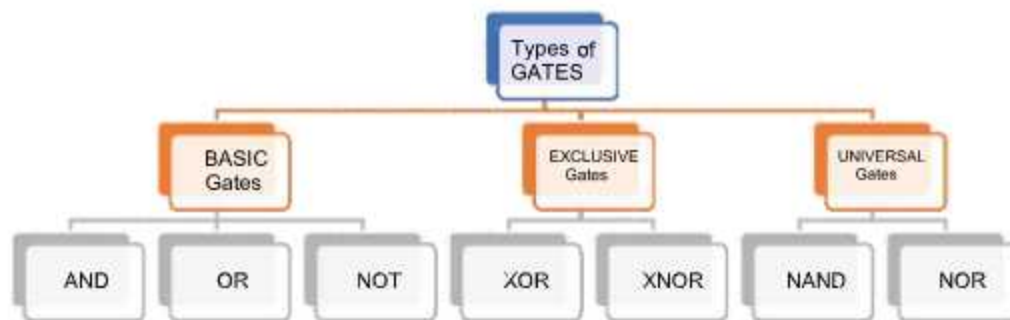


Figure : 2.13

### 2.4.1 BASIC GATES :

**a. AND GATE :** The AND gate plays a vital role in the digital logic circuit. The output value of the AND gate will always be 0 when any of the inputs states is 0. Simply, if any input state in the AND gate is low, then it will always return low output i.e 0.

The logic or Boolean expression for the AND gate is the logical multiplication of inputs denoted by a full stop or a single dot as

$$A \cdot B = Y$$

The value of Y will be true when both the inputs A and B are set to true. The logic symbol and truth table of AND gate for two logic variables A and B is shown below:

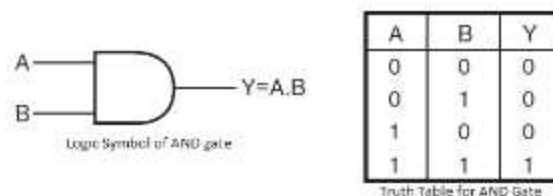


Figure : 2.14 (Two input AND gate)

**b. OR GATE :** It performs an O Ring operation on two or more than two logic variables. This gate can have two or more inputs and one output. The output of an OR gate is LOW (0) only when all of its inputs are LOW (0) otherwise output is HIGH (1). The OR operation on two logic variables A and B is written as  $Y = A + B$  and reads as Y equals A OR B. Here, A and B are input logic variables and Y is the output. The output value of the OR gate will always be 1 when any of the inputs states is 1. Simply, if all input state in the OR gate is low, then it will always return low output i.e 0.

The logic or Boolean expression for the OR gate is the logical ADDITION of inputs denoted by  $A + B$

$$X + Y = Z$$

The value of Z will be true when any of the inputs X and Y is set to true.

The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false." In other words, for the output to be 1, at least input one OR two must be 1. The logic symbol and truth table of OR gate for two logic variables A and B is shown below:

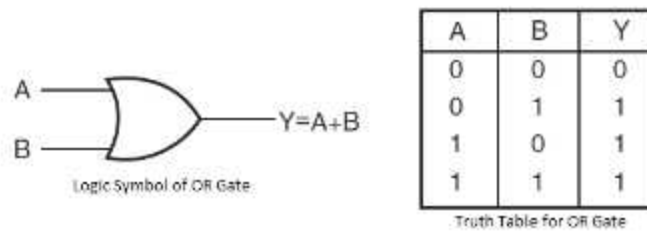


Figure : 2.15 (Two input OR gate)

**c. NOT GATE :** A NOT gate is also called an inverter . A NOT gate performs logical negation on its input. In other words, if the input is true, then the output will be false and vice -versa. The standard NOT gate is given a symbol whose shape is of a triangle pointing to the right with a circle at its end. This circle is known as an “inversion bubble” and is used in NOT, NAND and NOR symbols at their output to represent the logical operation of the NOT function.

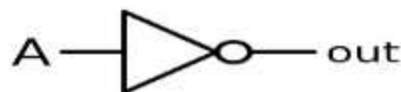


Figure : 2.16

#### 2.4.2 Exclusive Gates :

**a. XOR:** “XOR” GATE is called exclusive OR GATE .The simplest XOR gate is a two-input digital circuit that outputs a logical “1” if the two input values differ, i.e., its output is a logical “1” if either of its inputs are 1, but not at the same time (exclusively). The Boolean expression for a two-input XOR gate, with inputs A and B and output X:

$$X = A \oplus B$$

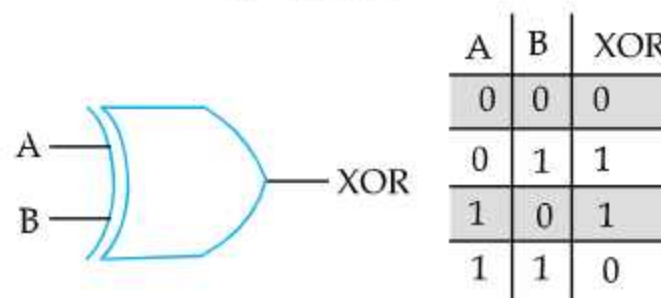
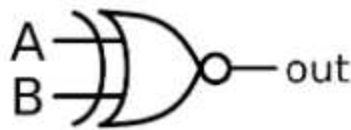


Figure 2.17. Symbol and truth table for a digital XOR gate

**b. XNOR:** The XNOR gate is a digital logic gate whose function is the logical complement of the Exclusive OR gate.

X-NOR gates are used mainly in electronic circuits that perform arithmetic operations and data checking such as Adders, Subtractors or Parity Checkers, etc.



TRUTH TABLE of XNOR GATE

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Figure : 2.18

### 2.4.3 Universal Gate

**a. NAND GATE:** A NAND Gate is a combination of two gates AND and NOT. It means opposite of an AND logic gate. It is also considered as a "universal" gate in Boolean algebra as through this gate all other logic gates can be constructed. The graphic symbol for the NAND gate consists of an AND symbol with a bubble on the output, denoting that a complement operation is performed on the output of the AND gate.

The truth table and the graphic symbol of NAND gate is shown in the figure.

The truth table clearly shows that the NAND operation is the complement of the AND.

X	Y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

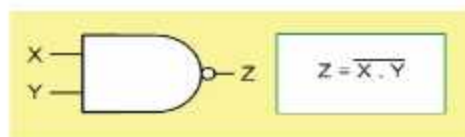


Figure : 2.19

The value of OUTPUT NAND will be true when any one of the input is set to 0.

**b. NOR Gate:** A NOR Gate is a combination of two gates NOT and OR . It means opposite of an OR logic gate. It is also considered as a "universal" gate in Boolean algebra as through this gate all other logic gates can be constructed.

The graphic symbol for the NOR gate consists of an OR symbol with a bubble on the output, denoting that a complement operation is performed on the output of the OR gate.

The truth table and the graphic symbol of NOR gate is shown in the figure.



X	Y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

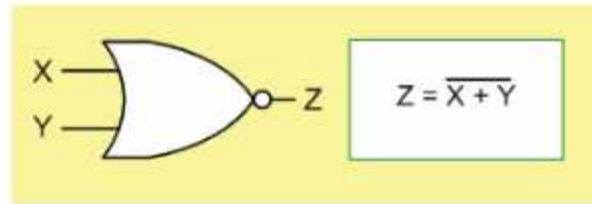


Figure : 2.20

The truth table clearly shows that the NOR operation is the complement of the OR.

### 2.5 Proof of NAND and NOR as UNIVERSAL GATES:

A universal gate is a gate which can implement any Boolean function without need to use any other gate type. The NAND and NOR gates are universal gates.

In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families.

**2.5.1 NAND as Universal Gate :** The NAND gate is a universal gate since it can implement the AND, OR and NOT functions. To prove that any Boolean function can be implemented using only NAND gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

#### a. Implementing an NOT GATE Using only NAND Gate

The figure shows two ways in which a NAND gate can be used as an inverter (NOT gate).

If there is a single input A to NAND GATE it gives  $(A.A)'$  as output. If  $A=0$  it will give output 1 and if  $A=1$  it will give output 0

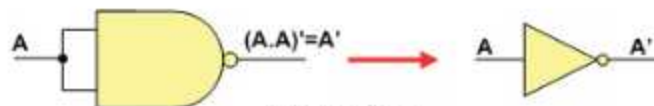


Figure : 2.21

#### b. Implementing AND Using only NAND Gates

We can make AND gate using NAND GATE. In the following figure The AND is replaced by a NAND gate with its output complemented by a NAND gate inverter.

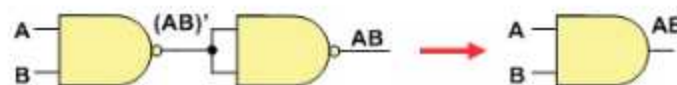


Figure : 2.22

### c. Implementing OR Using only NAND Gates

We can make OR gate using NAND GATE. In the following figure the OR gate is replaced by a NAND gate with all its inputs complemented by NAND gate inverters.

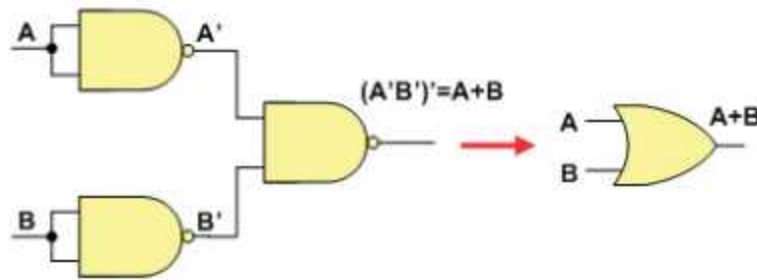


Figure : 2.23

So, the NAND gate is a universal gate because it can implement the AND, OR and NOT functions.

**2.5.2 NOR Gate is a Universal Gate:** In order to prove NOR gate is a universal gate we will show that the AND, OR, and NOT operations can be performed using these gates.

#### a. Implementing NOT GATE Using only NOR Gate

The figure shows two ways in which a NOR gate can be used as an inverter (NOT gate).

All NOR input pins connect to the input signal A gives an output A'.

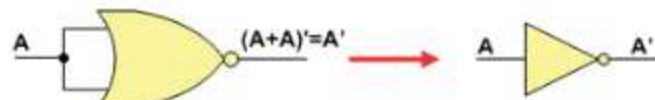


Figure : 2.24

#### b. Implementing OR Using only NOR Gates

An OR gate can be replaced by NOR gates as shown in the figure (The OR is replaced by a NOR gate with its output complemented by a NOR gate inverter)

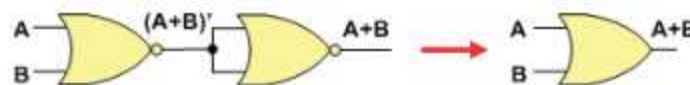


Figure : 2.25

#### c. Implementing AND Using only NOR Gates

An AND gate can be replaced by NOR gates as shown in the figure (The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters)

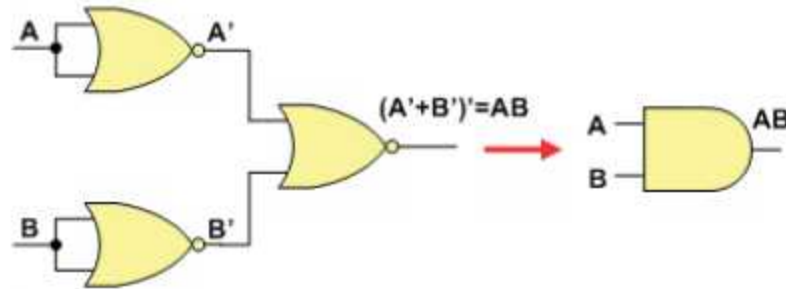


Figure : 2.26

Thus, the NOR gate is a universal gate since it can implement the AND, OR and NOT functions.

**2.5.2 DEMORGAN'S THEOREM** is used in designing digital circuits and also used to solve the various Boolean algebra expressions. It solves the basic gate operation likes NAND gate and NOR gate. It is based on the principal break the line and change sum to product and product to sum. The theorem states that the complement of the product of all the terms is equal to the sum of the complement of each term. Likewise, the complement of the sum of all the terms is equal to the product of the complement of each term. so, it has two laws.

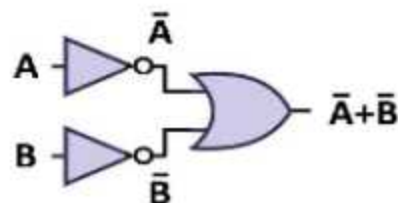
**THEOREM 1:** it proves that in conditions where two (or more) input variables are multiplied and negated, they are equal to the OR of the complements of the separate variables. In simple word it states the complement of a product is equals to the addition of the complement

$$(AB)' = A' + B'$$

A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0



... is equivalent to ...



$$\overline{AB} = \overline{A} + \overline{B}$$

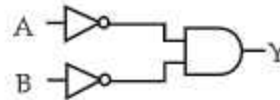
Figure : 2.27

**THEOREM 2 :** It proves that in conditions where two (or more) input variables are Added and negated, they are equal to the AND of the complements of the separate variables. In simple word it states the complement of a sum equals to the product of complement



Complement of sum is equal to the product of complement.

$$(A+B)' = A' \cdot B'$$



A	B	$\bar{A}$	$\bar{B}$	$A+B$	$\overline{A+B}$	$\bar{A} \cdot \bar{B}$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

Figure : 2.28

## Points to Remember

- There are two basic types of number system i.e Non-Positional Number System and Positional Number System.
- The total number of digits in number system is called base or radix.
- NAND and NOR are called Universal Gates
- There are four types of number system that computer supports i.e Decimal Number System, Binary Number System, Octal Number System, Hexadecimal Number system.
- XOR GATE is also called Exclusive OR GATE and XNOR GATE is called Exclusive NOR GATE.
- XNOR GATE is an electronic circuit is used in data checking such as Adders, Subtractors or Parity Checkers.

## Exercise

### Ques 1. Multiple choice Questions

1. A numbers system which uses 16 different symbols is called
  - a. Decimal Number system
  - b. Binary Number System
  - b. Octal Number system
  - c. Hexa decimal Number system
2. NAND GATE is a combination of
  - a. AND ,NOT
  - b. OR , NOT
  - b. AND,OR
  - c. NOT,XOR

3. Radix of octal numbers System is
  - a. 2
  - b. 5
  - c. 8
  - d. 16
4.  $(38CB)_{16}$  is an example of which number sytem is
  - a. Decimal Number system
  - b. binary Number System
  - b. Octal Number system
  - c. hexa decimal Number system
5. The value of  $(111)_2$  in decimal number sytem is
  - a. 7
  - b. 3
  - c. 6
  - d. 8

#### Ques 2. Fill in the blanks

- I. Logic GATES are made up of \_\_\_\_\_ and \_\_\_\_\_.
- II. A NOT GATE is also called \_\_\_\_\_ GATE.
- III. \_\_\_\_\_ and \_\_\_\_\_ are universal gate.
- IV. There are two types of basic number sytem \_\_\_\_\_ and \_\_\_\_\_.
- V. Roman Number System is an example of \_\_\_\_\_ number system.

#### Ques3. True and False

- I. The logic for AND gate is logical Addition.
- II. Octal word comes from latin word oct which means 8.
- III. The chmod command in unix operating sytem uses octal number system to assign file permission
- IV. XNOR means exclusive NOR.
- V. Hexa means 6

#### Ques 4. Short Answer type Questions:

- I. What is logic Gate?
- II. What are the basic types of number system?
- III. Give at least 3 examples of positional number system.
- IV. What is hexa decimal number system?
- V. Explain AND GATE with truth table.
- VI. Why NAND is called universal gate?
- VII. Explain XOR GATE with truth table.

VIII. Explain NAND Gate with truth table.

**Ques5. Long Answer type Questions:**

- I. What are universal gates ? Explain one in detail.
- II. Explain the four types of number system that computer supports.
- III. Explain De-Morgan's theorem in detail.
- IV. Explain in detail about the types of logic gates.

## Exercise

DECIMAL	BINARY	OCTAL	HEXADECIMAL
0	0000	0	0
1	0001	---	1
2	0010	2	---
3	---	3	3
4	0100	---	4
5	0101	---	5
6	0110	6	---
7	---	7	
8	1000	10	8
9	---	11	9
10	1010	12	---
11	1011	---	B
12	---	14	C
13	1101	15	---
14	1110	---	E
15	1111	17	---



**SOLVE THE FOLLOWING:**

Decimal	Binary	Octal	Hexadecimal
23			
	111101		
		27	
			38C
89			
	10001		
			8F
		37	
			9A
99			

Complete the following

1.  $\overline{(A+B)} + C =$

2.  $A.B + \overline{C} =$

3.  $(A+B) + C =$

4.  $A + \overline{\overline{(B+C)}} =$

5. 



# Introduction to Programming Languages



## OBJECTIVES OF THIS CHAPTER

- 3.1 Introduction
- 3.2 Concept of Program, Programming and Programmer
- 3.3 Programming Languages
- 3.4 Language Translator
- 3.5 Types of Errors

### 3.1 INTRODUCTION:

In this chapter, we are going to learn about the concept of program, programmer, programming, and different categories of programming languages used for computer systems. A program is basically a set of instructions to be executed by computer to perform some tasks. The process of writing a program is called programming. The person who writes the program is called programmer. When a programmer writes a program, he or she goes through a particular process. This process of developing a program is called programming process. The programmer can use any language from hundreds of available programming languages for program development.

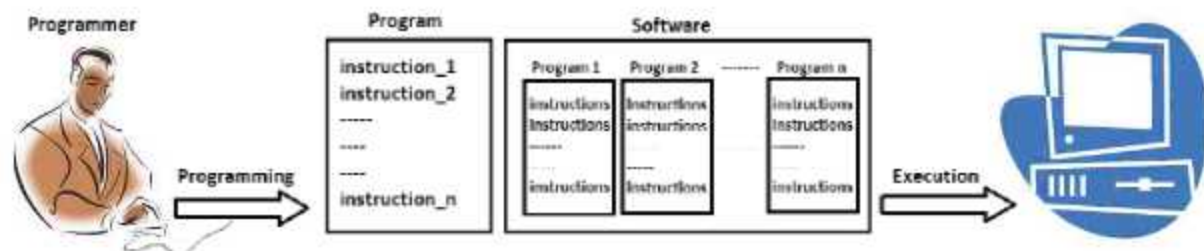
### 3.2 CONCEPT OF PROGRAM, PROGRAMMING AND PROGRAMMER

We know that a computer system basically consisting of two parts: hardware and software. Without software, computer hardware cannot do anything. Hence, computer is nothing but a piece of metal without software. To make the computer-hardware to do something, we must install and use software in our computer system. Now the question arises “what is software.”

**Software** is a set of computer programs which are designed and developed to perform desired tasks on computer. It is the software which makes a computer capable of data processing, storing and retrieval. Basically, softwares are categorised into two types: system software and application software. **System software** are designed and developed to control the functionality and to operate computer system hardware while

the **application software** are designed and developed to perform specific tasks using computer system. System softwares are more complex as compared to application softwares. The development (programming) of system softwares requires more skills as compared to application software development.

Software is usually not a single entity. It is a set (collection) of programs. A programmer must write instructions in a prescribed sequence of the programming language being used for development, so that the computer system becomes capable of successfully performing the desired task. Thus, we can say that a **Program** is a set of Instructions that the computer executes.



*Fig. 3.1 Program, Programmer, Programming and Software*

The process of writing system program is known as **System programming** and the programmer writing the same is called System Programmer, whereas writing application program is known as **application programming** and the programmer writing the same is called **application programmer**.

### 3.3 PROGRAMMING LANGUAGES

The programming languages are similar to natural languages which are used in our daily life such as Punjabi, Hindi and English etc. As we use natural languages for communication purposes, similarly computer programming languages are used to communicate with the computer systems and to make computers work as desired through softwares.

System programs (Example: Operating Systems) are designed to control and operate the input/output devices, memory, processor etc. To write system program, programmer needs to control the hardware components of computer system. It is possible only if the programmer knows the internal architecture of hardware components. Therefore, system programming is the task of skilled programmers that have a detailed knowledge of the hardware components of the computer system. Machine language, Assembly language and C languages are widely used to develop system programs.

Application programs are developed to perform a particular task or to solve a particular



problem. For example: student management system, library management system, payroll system, inventory control system, word processors, spread sheets, graphics software etc are application programs. Application programmer does not need to possess in-depth hardware knowledge. The most popular application programming languages are PYTHON, COBOL, FORTRAN, BASIC, PASCAL, C, C++, JAVA etc.

### 3.3.1 Types of Programming Languages:

In this era, hundreds of programming languages are available. These languages are categorised on the basis of their ability to develop different kinds of software. Some of the programming languages are best for writing system software while some others are best suitable for writing application software or mobile applications:

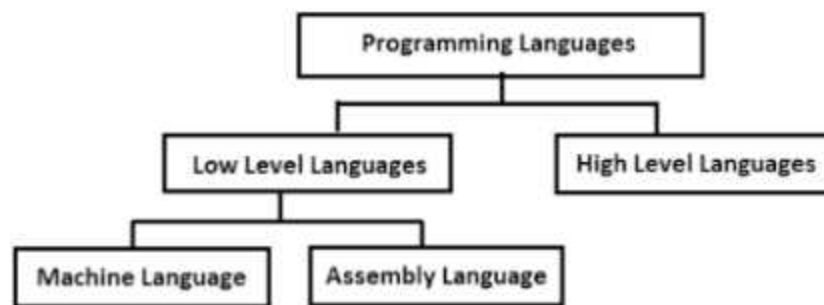


Fig. 3.2 Types of Programming Languages

For designing and developing system programs, low level programming languages are used while for developing application-programs, many high-level programming languages have been designed. Let's know more about different type of programming languages:

#### A. Low Level Languages:

Machine Language and Assembly Language are called Low Level Languages. These programming languages are close to computer hardware and have more direct access to the features of the hardware. These are used to develop drivers, high performance code, kernels for operating systems etc. A detailed explanation of low-level languages is given below:

- a. **Machine Language:** Machine language is also known as **Binary Language**. It is considered as the first generation of computer programming languages. Machine language is the fundamental language for computer systems because this language is directly understood by the computer hardware. Unlike high level programming languages, there is no need for translation of machine language code to make it understandable by the computer. This language consists of only two binary digits - 0 and 1.

Every instruction in machine language consists of two parts: Opcode and operand, as shown in the diagram below:

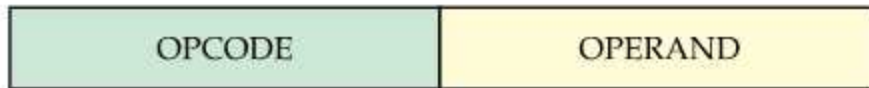


Fig. 3.3 Instruction format in machine Language

Here, Opcode is the Operation Code and Operand is the Operation Address. The first part - **Operation code** is the command which tells the computer what operation is going to be performed. The second part - **Operation Address** is the memory address which tells the computer where to find the data for the operation to be performed.

Because the instruction codes are written using binary digits 0 and 1 only, so it becomes difficult to remember the machine instruction codes in binary format. There are many advantages and disadvantages of using machine language some of which are explained below:

#### Advantages of Machine Language:

- Binary format instructions are **directly understood** by computer without any translation.
- Machine instructions are **fast in execution** because these are executed directly without any translation.

#### Disadvantages of Machine Language:

- It is **difficult to remember** the machine instruction codes as they are made up of complex combinations of binary digits i.e. 0 and 1.
- It is the most **difficult process to find errors** in the machine instruction codes.
- High level of knowledge of **low-level internal details** of hardware is required for programming in machine language.
- Programs developed in machine language are **machine dependent**, because machine instructions are written according to the underlying architecture of computer system. So, these instructions are machine specific which cannot be executed on the computer systems having different architecture.

**b. Assembly Language:** This language is also known as **Symbolic Language** because symbolic names of instructions are used instead of binary codes. This language is considered as second generation of computer programming languages. The major benefit of assembly language as compared to machine language is that it reduces coding time and the amount of information the programmer has to remember. The symbolic names of instructions can be easily remembered therefore, it also becomes easy to find errors in the program and to modify it as compared to machine language.

Despite of these benefits, programming in assembly language still requires in-depth technical knowledge of hardware. So, programmer must be aware of the machine architecture for programming in assembly language. Due to this hurdle, programs written in assembly language are still machine-dependent. These programs cannot be executed on other machines having different architectures.



Symbolic names used for operation codes in Assembly Language are called **Mnemonic Codes**. For example: the codes for addition, subtraction, multiplication, and division operation are ADD, SUB, MUL and DIV respectively in Assembly language.

Now the question arises how do computers execute the assembly language code because computers can execute instructions only in binary format. In order to execute an assembly language code on a computer, it must be translated into equivalent machine understandable code. For this translation, a translator program, named **Assembler**, is used. Assembler is a language translator program which translates the assembly language code into equivalent machine code. In the following section, we will study in detail about the assemblers.

#### **Advantages of Assembly Language:**

- It is easy to learn and remember the codes of assembly language because it uses codes like english language instead of binary digits as compared to low level (machine) language.
- Finding and correcting errors in the assembly language program is easy as compared to machine (binary) language.
- Assembly language programs have the equivalent efficiency as the machine language programs.

#### **Disadvantages of Assembly Language:**

- Knowledge of low-level internal details of hardware is required for programming in assembly language. Therefore, hardware technical skills are required for the programmer to do programming in assembly language.
- Programs developed in Assembly language are machine dependent because assembly instructions are written according to the underlying architecture of computer system. So, these instructions are machine specific which cannot be executed on the computer systems having different architecture.

### **B. High Level Languages:**

The primary objective of developing high level languages is that these languages facilitate a large number of people to build programs or software without the need to know the internal low-level details of computer system hardware. These languages are designed to be machine-independent.

**High level languages** are English like languages. These languages use simple & special characters and numbers for programming. Therefore, these languages make it easy for common people to learn and write computer programs. An instruction written in high level language is usually called a **Statement**. Each high-level language has its own rules for writing program instructions. These rules are called **Syntax** of the language. Some of the commonly used High Level Languages are: PYTHON, BASIC, COBOL, FORTRAN, PASCAL, C, C++, JAVA, CSHARP etc.



Similar to Assembly Language, high level languages cannot be directly understood by computer systems. Language translators are required for translating them into machine understandable format. There are two approaches for translating high level languages into machine code: first is via **Compiler** and other is via **Interpreter**. Each high-level language has its own translator program. We cannot translate a program written in one specific high-level language with the compiler of some other specific language. For example, we cannot compile C Program using COBOL Compiler or vice-versa.

Some of the common categories of high-level languages are discussed below:

- **Procedural or Procedure Oriented Languages:** Procedural languages are considered as the **Third Generation of Programming Languages (3GLs)**. In procedural languages, a program can be written by dividing it into small procedures or subroutines. Each procedure contains a series of instructions for performing a specific task. Procedures can be re-used in the program at different places as required. These languages are designed to express the logic of a problem to be solved. The order of program instructions is very important in these languages. Some popular Procedural languages are FORTRAN, COBOL, Pascal, C language etc.
- **Problem-Oriented or Non-Procedural Languages:** Problem oriented languages are also known as Non-Procedural languages. These languages are considered as the **Fourth Generation of Programming Languages (4GL)**. These languages have simple, English-like syntax rules and they are commonly used to access databases. It allows the users to specify what the output should be instead of specifying each step one after another to perform a task. It means there is no need to describe all the details of how the data should be manipulated to produce the result. This is one step ahead from third generation programming languages. These languages provide the user-friendly program development tools to write instructions. Using these languages, user writes the program using application generator that allows data to be entered into the database. The program prompts the user to enter the needed data and then it checks the data for its validity. Examples of problem-oriented languages are: SQL (Structure Query Languages), Visual Basic, C# etc. The objectives of these languages are to increase the speed of developing programs and reduce errors while writing programs.
- **Object-Oriented Programming Languages:** The Object-Oriented programming concept was introduced in the late 1960s, but now it has become the most popular approach to develop software. In these programming languages, a problem can be solved by dividing it into a number of objects. Object-Oriented languages support the concept of object, class, encapsulation, data hiding, inheritance and polymorphism etc. Now-a-days, most popular and commonly used Object-Oriented programming (OOPs) languages are C++ and Java.

- **Logic-Oriented Languages:** These languages use logic programming paradigms as the design approach for solving various computational problems. Any program written in a logic programming language is a set of sentences in logical form. These sentences express facts and rules about some problem domain. Major logic programming language families include Prolog, Answer Set Programming (ASP) and Datalog. In all of these languages, rules are written in the form of clauses. Such languages are very beneficial in the field of Artificial Intelligence and Robotics.

### Advantages of High-Level Languages

Some of the common advantages of high-level languages are given below:

- High Level languages are easy to learn and understand as compared to low level languages. It is because the programs written in these languages are similar to English statements.
- The errors in a high-level language program can be easily detected and removed. All the syntax errors are detected and removed during the compilation process of the program.
- These languages provide a large number of built-in functions that can be used to perform specific task during programming which results in huge time saving i.e. much faster development.
- Programs written in high level language are machine independent. A program written for one type of computer architecture can be executed on another type of computer architecture with little or no changes.

### Disadvantages of High-Level Languages

Some of the common disadvantages of high-level languages are given below:

- A program written in high level languages has lower efficiency and speed as compared to equivalent programs written in low level languages.
- Programs written in high level languages require more time and memory space for execution.
- High level languages are less flexible than low level languages because normally these languages do not have direct interaction with computer hardware such as CPU, memory and registers.

## 3.4 LANGUAGE TRANSLATORS

Language translators are also called Language Processors. These are the system programs which are helpful to develop programs. Language translators are designed primarily to perform two main functions as described below:

- These are designed to translate source programs into machine's object code. **Source programs** may be written in Assembly Language or High-Level languages while **object code** is a code that a computer CPU can understand without any translation.



These translator programs are also designed to detect any **syntax errors** in the source program. Successful translation of source program into object program takes place only if the source program does not have any syntax errors in it.

Each language has its own translator program which can translate the program written only in that specific language. Assembler is a translator program which can translate the source program written in assembly language only. Similarly, each high-level language has its own translator program, known as Compiler and Interpreter. Some High-level languages use Compiler (for example: C/C++ Language) while some other uses Interpreter (for example: BASIC language). But there also exist some languages which have both compiler and interpreter for different levels of translation. For example: JAVA is a language which has both compiler and interpreter. All these types of translator programs are discussed below:

### 3.4.1 Assembler:

It is a language translator which converts assembly language program into machine-understandable format. The program written in assembly language is called Source Program. This source program cannot be directly understood by the computer system. That is why it must be translated into machine understandable format for the execution. It is the assembler which translates this assembly language source program into machine understandable form. The source program after translation (in machine understandable form) is called Object Program (Code).

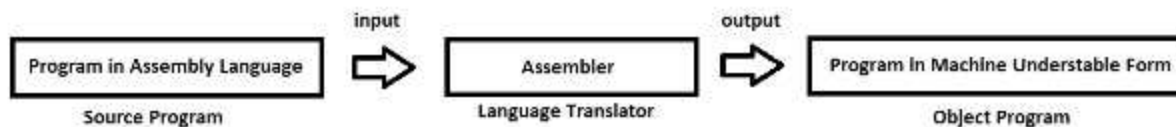


Fig. 3.4 Working of Assembler

### 3.4.2 Interpreter and Compiler:

There are two types of language translators for High level languages i.e. Interpreters and Compilers. These translators are used for translating source programs written in High Level languages into machine understandable form.

In the first approach, i.e. Interpreter, one statement of high-level language program is taken at a time and it is translated into machine instruction which is executed immediately by the processor. It means no object program is saved in this approach of translation. Whenever we want to execute the program we have to translate the source program every time.

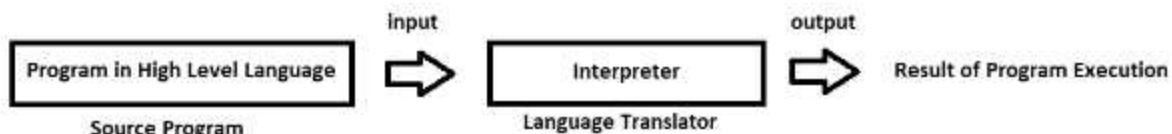
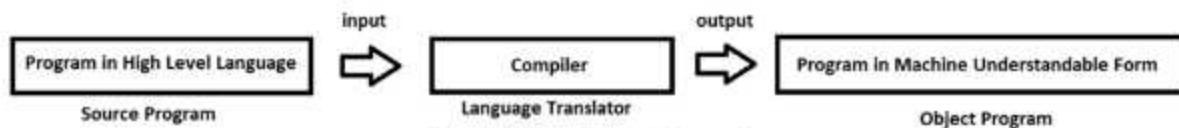


Fig. 3.5 Working of Interpreter



Interpreters do not require large memory space to translate and execute programs. The main disadvantage of interpreters is that they require same amount of time whenever we execute programs on a computer system because every statement of source program must be translated every time.

In the other approach, i.e. Compiler, all statements of the high-level language program are taken at a time and they are translated into machine understandable form which is stored as Object Program in Memory. This object program is provided to computer system whenever program is executed. Compilers take more time to translate source program as compared to interpreter. But compiled object program runs much faster than the interpreted program.



*Fig. 3.6 Working of Compiler*

Each High-Level language has its own compiler. We cannot compile the source code of one language with the compiler of another language. For example, FORTRAN compiler cannot compile the source code written in COBOL language and vice-versa.

The difference between an interpreter and a compiler may be understood with the help of following analogy. Suppose we want to translate a speech from Tamil to Hindi. We can use two approaches to do this translation. In first approach, translator listens to a sentence in Tamil and immediately translates it into Hindi. In the second approach, the translator listens to the whole passage in Tamil and then gives the equivalent Hindi passage. An interpreter is similar to first approach of translation where sentence-by-sentence translation is carried out. On the other hand, compiler is similar to second approach where whole passage is translated in a single step.

Some programming languages, such as JAVA, use both Compiler and Interpreter for the translation and execution of programs. Here the compiler is used to translate the source code into machine code that is understood by any type of machine, while the interpreter is used to translate and execute the compiled machine neutral code according to underlying machine on which it is being executed.

### 3.5 PROGRAMMING PROCESS

We know that a computer needs a program to tell it, "what to do". Instructions in the program guide the computer how to solve the given problem. But developing a program is not a simple and easy task. A programmer has to go through a specific process for developing a program successfully. The steps involve in the programming process are listed below and are required to be followed in the same sequence:

1. Defining the problem to be solved
2. Plan the solution of the problem
3. Coding the solution in the high-level language

4. Compile the program
5. Test and Debug the program
6. Documenting the program

These steps can be explained in detail as follows:

### **3.5.1 Defining the problem:**

It is the first step in the programming process. Before a programmer begins his task, he must need to know the extensive details of the problem to be solved through programming. The details of the problem should be provided to the programmer so that he gets a clear understanding of it. Analysis of the problem shows, what the required inputs and outputs will be there for the solution of problem. After having a clear understanding of the problem, programmer starts thinking about how to solve the given problem.

### **3.5.2 Planning the Solution:**

The next step after defining the problem is to prepare a detailed list of steps required to be carried out for solving the problem. We will take an example which shows why planning is required for solving the problem. A teacher asks the student to solve a specific mathematical problem and the student is not familiar with the steps involved in solving the problem. Thus, he would not be able to solve it. The same principle applies to writing computer programs also. A programmer cannot write the instructions for any program unless he understands how to solve the problem manually.

If a programmer knows the steps for solving the problem, but, while programming, if he applies the steps in the wrong sequence or he forgets to apply any of these steps, he will get a wrong output or even no output at all. Hence, to write an effective program, a programmer has to write all instructions in the correct sequence. Therefore, to ensure that the instructions of the program are appropriate and are in the correct sequence, the programmer must plan the program before start writing. For planning and defining the steps for the programs, programmers normally use Algorithms and Flow-Charts.

### **3.5.3 Coding the Solution:**

The sequence of operations defined in flow-chart can be converted into instruction using some High-Level programming language, such as C, C++, and PASCAL etc. Coding is the process of writing the instructions using programming language to make a program. This program file is known as source program or source code. This code is stored in a program file. This program contains the logic or steps for solving the problem.

### **3.5.4 Compile the program:**

After writing program code in High level language, we have to translate it into such a form that computer can understand and execute it, because, computer can understand



instructions only in binary format. A compiler is a small program that translates the source program into a machine understandable form (i.e. object code). This conversion of source code into object code is known as **Compilation**. During compilation, compiler also scans the source program for syntax errors. If there are syntax errors in the program, compiler generates error messages. These errors must be corrected to generate the object code. Then the object code will be stored in a file. Whenever we want to execute the program, this object code is supplied to computer for execution.

### 3.5.5 Testing and Debugging the Program:

Testing and debugging are important steps in the software development. Testing is a process which makes it sure that program (software) performs the intended task. Testing is a time-consuming task. As long as human beings make programs, the programs will have errors. These program errors are called **Bugs**. The process of detecting and correcting such bugs is called **Debugging**.

### 3.5.6 Documenting the Program:

The final stage in the development process of a program is documentation. The term documentation means specifying the important information regarding the approach and logic applied in the program by the programmer. The documentation enables other programmers to understand the logic and purpose of the program. It is also helpful in the maintenance of the program.

## 3.6 TYPES OF ERRORS IN THE PROGRAM

Errors are the faults that occur in the program. These errors make the behaviour of the program abnormal. Generally, two types of errors are found in the programs:

- **Syntax Errors:** These errors are also known as Compilation Errors. Such errors occur when we do not follow the rules or syntax of programming language being used. These types of errors are automatically detected by compilers during compilation process. A program cannot be successfully compiled until all of the syntax errors in the program are removed. Some examples of syntax errors in Java language are: missing semicolon, variable not declared, un-terminated string, compound statement missing etc.
- **Logical Errors:** These errors occur when there are errors in the logic of the program. If our program has logic errors, though it will compile successfully but it may produce wrong result/output. Such types of errors cannot be detected by the compilers. These errors are either traced out manually by the programmer or some debugging tools may be used to detect such errors. Programmer can detect any faulty logic by examining the output.



- **Runtime Errors:** A runtime error is an error that occurs during the execution-time of a program even after the successful compilation. Sometimes, when a program is running, it is not able to perform the operation. It may be due to the run-time error. These errors are very difficult to find, as the compiler does not point to the location of these errors. Examples of the most common types of runtime errors are:
  - ❖ IO error
  - ❖ Division by zero errors
  - ❖ Out of range errors



## Points to Remember

1. A Program is a set of instructions while a Software is a set of programs.
2. A Programmer is the person who writes the program code.
3. The Process of writing a program is called Programming
4. Machine language is directly understood by computer systems and consists of binary digits i.e. 0 and 1.
5. Assembly Language uses mnemonic codes to write instructions in the program.
6. High Level languages use alphanumeric codes to write instructions in the program.
7. Assembler is a language translator which translates Assembly language source program into machine understandable format which is called Object Program Code.
8. Compiler is a language translator which translates High Level language source program into machine understandable format which is called Object code.
9. Writing the instructions to make a program using some computer language is called coding.
10. Errors are the faults that occur in the program.
11. Finding and correcting errors in the program is called Debugging.
12. Syntax errors are also known as Compilation Errors.
13. Logical and Runtime errors cannot be detected by the compilers.

## Exercise

### Que: 1 Multiple Choice Questions

- I. Set of programs is called \_\_\_\_\_.
  - a. Group
  - b. Software
  - c. Program
  - d. None of these
- II. Which language is directly understood by computer without any translation?
  - a. Procedure Oriented Language
  - b. Machine Language
  - c. Assembly Language
  - d. High Level Language
- III. Mnemonic codes and symbolic addresses are used in which programming language?
  - a. Object Oriented Language
  - b. Non-Procedural Language
  - c. Assembly Language
  - d. Machine Language
- IV. Which translator does not save object code after translation of source program written in high level language?
  - a. Interpreter
  - b. Compiler
  - c. Assembler
  - d. None of these
- V. Process of finding and correcting errors in a program is called \_\_\_\_\_.
  - a. Compilation
  - b. Coding
  - c. Debugging
  - d. Documentation

### Que: 2 Fill in the Blanks:

- I. A person who writes the program is called \_\_\_\_\_.
- II. Low level internal details of hardware are required for programming in \_\_\_\_\_.
- III. Process of translating source program written in high level language into object code file is called \_\_\_\_\_.
- V. Those errors which are detected by the compilers are called \_\_\_\_\_ errors.

### Que: 3 Write the Full form of following:

- I. Opcode
- II. Operand
- III. 4GL
- IV. OOP

### Que: 4 Short Answer Type Questions.

- I. Define Programming.

- II. What are Low Level Languages?
- III. What do you know about Object-Oriented Programming Languages?
- IV. Write the steps used in Programming Process.
- V. What are Syntax Errors?
- VI. What is Assembler?
- VII. Write the instruction format in Machine Language.
- VIII. Which translators can be used for translating High Level Programming Languages?

**Que: 5 Long Answer Type Questions.**

- I. What are High Level Programming Languages? Explain.
- II. What are Language Translators? Explain Compiler and Interpreters.
- III. What is Error? Explain different types of errors found in the computer programs.
- IV. What is Machine Language? Write its advantages and disadvantages.
- V. Explain the steps involved in programming process.





# Introduction to OOP with Java



## OBJECTIVES OF THIS CHAPTER

- 4.1 Introduction to Object Oriented Programming
- 4.2 Principles of Object-Oriented Programming - abstraction, encapsulation, inheritance, and polymorphism
- 4.3 Introduction to Java
- 4.4 History of Java
- 4.5 Features of Java
- 4.6 Basic Structure of Java Program
- 4.7 Creating an Executing a Simple Java Program
- 4.8 Basic Elements for Java Programming: character set, tokens, comments

### 4.1 INTRODUCTION TO OBJECT ORIENTED PROGRAMMING

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects. Thus, we can say that Object-Oriented Programming Languages are based on the concept of "objects". Objects are the real-world entities such as a student, person, pen, table, television, computer etc.

Anything in the world can be defined as an object. In the Object-Oriented Programming, it can be defined in terms of its properties and behaviour. Consider an Example of Person. It is an object. Its properties will be: name, age, height, weight, etc. and the behaviour (actions) can be - walk, talk, sleep etc. Each object has its own properties that describe what it is or it does.

In the Object-Oriented Programming methodology, a program is designed using classes and objects. It simplifies software development and maintenance by providing some concepts such as: Object, Class, Inheritance, Polymorphism, Abstraction and Encapsulation. The programming paradigm where everything is represented as an object is known as a Truly Object-Oriented Programming Language.

Simula is considered as the first Object-Oriented Programming Language while Smalltalk is considered as the first Truly Object-Oriented Programming Language. Significant Object-Oriented Programming Languages include: Java, C++, C#, Python, R, PHP, Visual Basic.NET, JavaScript, Ruby, Perl, Object Pascal, Dart, Swift, Scala, Kotlin, Common Lisp, MATLAB, and Smalltalk etc.

## 4.2 PRINCIPLES OF OBJECT-ORIENTED PROGRAMMING

Object-Oriented Programming is the most widely used programming paradigm. It helps in developing large-scale applications in the real world using the modules in the form of classes and objects. These Modules allow multiple developers to work together with such that, it can make the complete system. Object-Oriented Programming allows the analysis and design of an application in terms of entities or objects. It means, the application has to implement the entities as they are seen in the real life and associate actions and attributes with each.

In OOP, code and data are merged into a single individual block – an object. When we approach a programming problem in an object-oriented language, we no longer ask how the problem will be divided into functions or procedures, but how it will be divided into objects. Every operation that is going to be functional is considered in terms of classes and objects. Using OOP, we can create classes defining the functionality and call that function by creating an object of that class.

These are the four main principles of the Object-Oriented Programming paradigm.

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

These four principles are considered as the four pillars of the any Object-Oriented Programming. Understanding of all these is essential to become a successful programmer. But before discussing these concepts, it is essential to know about, "what classes and objects are?". Let's get started using these concepts for better understanding of OOP:

**4.2.1 Objects :** Any real-world entity that has state and behaviour (action) is known as an object, for example: a chair, pen, table, keyboard, bike, etc. In the Object-Oriented Programming:

- ❖ We use data about the object to define its state. This data/state of an object is stored in the form of fields (variables). Fields are also known as attributes or properties.
- ❖ To define the behaviour of object, we use code written in the form of methods (functions). These methods can access and modify the object's data-fields.

Each object has its own properties that describe what it is or it does. Objects are created at runtime from templates, which are also known as classes. An object takes up some space in memory. Following diagram shows the concept of an object with real-life example of entity:



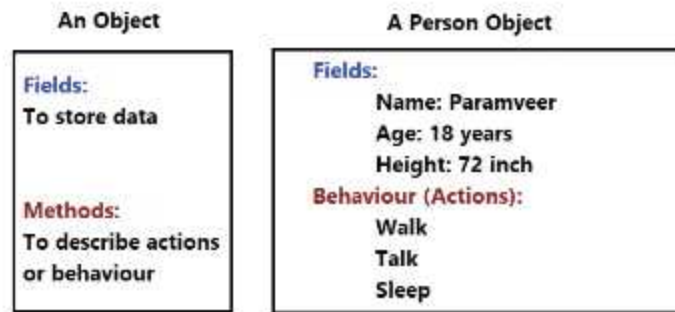
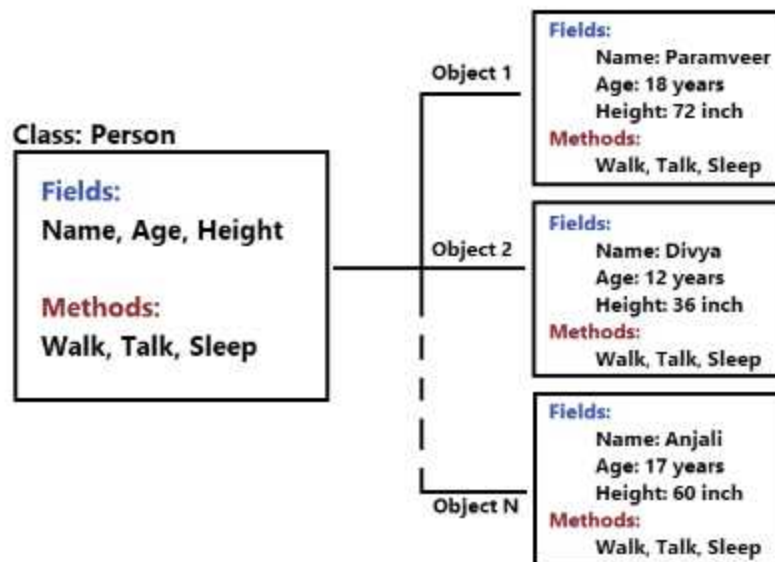


Fig: 4.1 Object and Its Example (Person Object)

**4.2.2 Class :** A class is a grouping of objects that have the same properties and common behaviour. A class can be considered as a blueprint from which we can create an individual object. When the individual objects are created, they inherit all the variables and methods from the class.

The class defines the characteristics of the objects. However, values can be assigned only after an object is created. Each object is said to be an instance of its class. In a class of Person, specific individuals such as: Paramveer, Divya, Anjali etc. are the objects of Person class.

A Class doesn't consume any memory space. It is only a logical representation of data. In simple words, we can say that a class is a template for objects, and an object is an instance of a class. Following figure shows the concept of a class and its objects:



**4.2.3 Abstraction :** It is one of the most important concepts of Object-Oriented programming. It defines how a real word entity can be represented into the programming entity. It is a process in which we select data from a larger pool to show only relevant details of the object to the user. It helps in reducing programming complexity and efforts.



In simple words, abstraction can be defined as hiding internal implementation of object and provides only the required features to the user. Let's understand with the help of an example:

**Real Life Example of Abstraction :** Consider, we are driving a bike. Here we are only concerned about driving the bike, like start/stop the bike, accelerate/break, etc. We are not concerned about how the actual start/stop mechanism or accelerate/brake process works internally. We are just not interested in those details.

**4.2.4 Encapsulation :** Encapsulation is a process of wrapping the data and methods together in a single unit. This single unit is known as Class. We can assume it as a protective wrapper that prevents random access of code defined within the wrapper from outside. It is just like a capsule that contains a mixture of several medicines.

Encapsulation enhances more security of the data, because everything related to a single task can be grouped together. Using encapsulation, access to the data is provided as per the requirement and this can be achieved using the concept of Data Hiding. In simple terms, encapsulation can be stated as:

**Encapsulation = Data Hiding + Abstraction**

Thus, Encapsulation allows selective hiding of properties and methods by wrapping them into a single unit called class. Data Hiding can be achieved by using private access modifiers. Private data and methods can't be accessed by anything outside the class.

Let's understand with the help of an example-

**Real Life Example of Encapsulation:** In the market, Capsules are available to cure different medical problems. In the capsule, different compositions are grouped to make a complete medicine that cures a particular medical problem. So, the grouping of that composition into a single unit in the capsule is a form of encapsulation.

**Features of Encapsulation:**

1. It provides extra security to class members using the concept of Data Hiding.
2. It groups data and methods into a single unit.
3. It attaches an extra security to the data that restricts to access data to unauthorized ones.

**4.2.5 Inheritance :** Inheritance is a mechanism in which one object acquires all the states and behaviours of a parent object. For example, a child inherits the traits of his/her parents.

In Object Oriented Programming, Inheritance is the process of creating new classes using the existing classes. In this process, new class acquires the features of the existing class. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance provides the facility of Reusability and is an important concept of OOPs.

Suppose there is a class, considered as the parent class, that has some methods associated with it. And we create a new class, considered as the child class, that has its own methods. So, when a child class is inherited from the parent class then all the methods

associated with parent class are available in the child class along with the child class own methods. Such process of creating new enhanced classes from existing classes is called Inheritance.

There are two terms involved in the inheritance – Base Class and Derived Class.

- ❖ Base Class – It is also termed as parent class. It is the main class that has the basic properties and methods which are defined.
- ❖ Derived Class – This is the extension of the base class and it is also called the child class. It has the properties and methods that are there in the base class with its own features.

**Real Life Example of Inheritance :** We all have seen the hardware updations for mobile phones. In the beginning, mobile phones were used only to talk and then started to be used for media access, Internet, Camera, etc. These are the evolution that arrived by adding some extra feature and making a new product, without rebuilding the complete functionality. So it is an example of inheritance.

**4.2.6 Polymorphism :** Polymorphism is the ability of an object to take on many forms. It is the most essential concept of the Object-Oriented Programming principle. The term Polymorphism consists of two words: Poly + morph. Here 'poly' means 'many' and 'morph' means 'forms'. So, polymorphism means many forms. In other words, "Many forms of a single object is called Polymorphism."

In Object-Oriented Programming, any object or method has more than one name associated with it. That is nothing but polymorphism. Let us understand the concept of Polymorphism with a real-life example.

Real Life Examples of Polymorphism:

- ❖ A teacher behaves students.
- ❖ A teacher behaves his/her seniors.

Here teacher is an object but the attitude is different in different situations.

Types of Polymorphism:

There are two types of Polymorphism in the Object-Oriented Programming:

1. **Compile-Time Polymorphism :** It is also known as static polymorphism or early binding. The compiler examines the method signature at compile time and determines which method to invoke for a given method call. In other words, we can say that in this type of polymorphism an object is bound with its functionality at the compile time. Compile Time Polymorphism is achieved using Method Overloading.

Method Overloading – When more than one method is declared with the same name but with a different number of parameters or different data-type of parameter, that is called method overloading.

2. **Run-Time Polymorphism :** It is also known as dynamic polymorphism or late



binding. In Runtime polymorphism, information to call a method is collected during runtime. In other words, we can say that in this type of polymorphism an object is bound with the functionality at run time. Run-Time Polymorphism is achieved using Method Overriding.

**Method Overriding** – When more than one methods are declared with the same name and same signature but in a different class. Then the derived class method will override the base class method. This means the Base class method will be shadowed by the derived class method. When the method is called then it can be called based on the overridden method.

### 4.3 INTRODUCTION TO JAVA

JAVA was developed by James Gosling at Sun Microsystems Inc in the year 1995. Later it was acquired by Oracle Corporation. Java is a simple programming language. Java is a general-purpose, class-based, object-oriented programming language, which works on different operating systems such as Windows, Mac, and Linux.

Java makes writing, compiling, and debugging programming easy. It helps to create reusable code and modular programs. Java allows developers to write once run anywhere. Simply, we can say that compiled Java code can run on all platforms that support Java. Java applications are compiled to byte code that can run on any Java Virtual Machine. The syntax of Java is similar to C/C++ programming languages.

We can use Java to develop:

- ❖ Desktop applications
- ❖ Web applications
- ❖ Mobile applications (especially Android apps)
- ❖ Web and application servers
- ❖ Big data processing
- ❖ Embedded systems

And much more

#### 4.3.1 Java Environment

Unlike many other programming languages including C and C++, when Java program is compiled, it is not compiled into platform specific machine; rather it is compiled into platform independent byte-code. This byte-code is then interpreted (executed) by the Java Virtual Machine (JVM). This byte-code can be executed on any platform, like Windows, Linux, or macOS. It means if we compile a program on Windows, then we can run it on Linux and vice versa. Each operating system has a different JVM, but the output produced by all the Operating System is the same after the execution of byte-code. That is why Java is considered as a platform-independent language.



Below are the environment settings for both Linux and Windows. JVM, JRE, and JDK all these are the essential components of Java and all these are platform-dependent because the configuration of each Operating System is different. But, Java is platform-independent. Few things of java environment must be clear before proceeding further which can better be perceived from the image provided as follows:



Fig: 4.3 Java Environment Structure (JDK, JRE and JVM)

- ❖ **JDK(Java Development Kit)** : The JDK is a key platform component for building Java applications. At its heart is the Java compiler. It contains JRE + Development Tools. Development Tools include Java compiler (javac), Javadoc, a debugger etc. JDK is intended for software developers.
- ❖ **JRE(Java Runtime Environment)** : JRE contains the parts of the Java libraries required to run Java programs. The JRE can be used as a standalone component to simply run Java programs, but it is also a part of the JDK. The JDK requires a JRE because running Java programs is part of developing them. This component is intended for end-users.
- ❖ **JVM (Java Virtual Machine)** : JVM is an abstract machine. It is a specification that provides a runtime environment in which java bytecode can be executed. JVMs are available for many hardware and software platforms. Simply, we can say that JVM acts as a run-time engine to run Java applications. JVM is the one that actually calls the main method present in a java code. JVM is a part of JRE(Java Runtime Environment). Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.

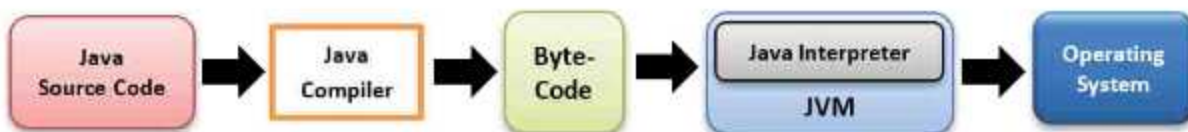


Fig: 4.4 Function of JVM

#### 4.4 HISTORY OF JAVA:

Java was started as a project called "Oak" by James Gosling and his team (Patrick Naughton, Chris Warth, Mike Sheridan, and Ed Frank) in June 1991. The idea was to develop a platform-independent language and create embedded software for consumer electronic devices. Gosling's goals were to implement a virtual machine and a language that had a familiar C-language notation but with greater uniformity and simplicity than C/C++. It took 18 months to develop and had an initial name as Oak. But, due to copyright issues. Thus, Oak was renamed to Java in 1995. The first public implementation was Java 1.0 in 1995. Java was later acquired by the Oracle Corporation. It was fairly secure.

Java is the name of an island in Indonesia where the first coffee (named java coffee) was produced. And this name was chosen by James Gosling while having coffee near his office. James Gosling is known as the Father of Java.

#### 4.5 FEATURES OF JAVA:

Java is one of the most popular programming languages in the world. It is easy to learn and simple to use. It has many powerful features which make it a powerful and a very popular programming language. Following are some of the most important features of Java:

- ❖ **Compiled and Interpreter :** Most languages are designed with purpose either they are compiled language or they are interpreted language. Java has both Compiled and Interpreter features. Java compiler compiles the source code to bytecode and JVM executes this bytecode to machine OS-dependent executable code.
- ❖ **Platform Independent :** Java Language is Platform Independent. It means program of java is easily transferable from one machine type to other. It is because Compiler converts source code to bytecode and then the JVM executes the bytecode generated by the compiler. This bytecode can run on any platform like Windows, Linux, or macOS. Which means if we compile a program on Windows, then we can run it on Linux and vice versa.
- ❖ **Object-Oriented :** Java is purely OOP Language. Whole the Code of the java Language is Written into the classes and Objects. Java allows the implementation of all the four main concepts of OOP: Abstraction, Encapsulation, Inheritance, and Polymorphism.



- ❖ **Robust** : Java language is robust, which means reliable. It is developed in such a way that it puts a lot of effort into checking errors as early as possible, that is why the java compiler is able to detect even those errors that are not easy to detect by another programming language. The main features of java that make it robust are garbage collection, Exception Handling and memory allocation.
- ❖ **Secure** : Java is secure due to many reasons. Java programs run inside a virtual machine which is known as a sandbox. Java does not support explicit pointer. Byte-code verifier checks the code fragments for illegal code that can violate access right to object. Java has multiple class libraries which provides secure APIs and hence each internal communication is verified for authentication by these APIs.
- ❖ **Distributed** : We can create distributed applications using the java programming language. The java programs can be easily distributed on one or more systems that are connected to each other through an internet connection. So, Java is specially designed for Internet-Users which uses the Remote Computers for executing their programs.
- ❖ **Simple, Small and Familiar** : Java is a simple Language Because it contains many features of other Languages like C and C++.Java removes complex features of these languages such as pointers, Storage Classes and Go to Statements etc. It also doesn't support Multiple Inheritance, operator overloading, Explicit memory allocation etc. features that are supported by C++.
- ❖ **Multithreaded and Interactive** : Java supports multithreading. It is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of the CPU.
- ❖ **Portable** : As we know, Java code written on one machine can be run on another machine (Write Once Run Anywhere-WORA). The platform-independent feature of java in which its platform-independent bytecode can be taken to any platform for execution makes java portable.
- ❖ **Dynamic and Extensible Code** : Java being completely object-oriented gives us the flexibility to add classes, new methods to existing classes and even create new classes through sub-classes. Java even supports functions written in other languages such as C, C++ which are referred to as native methods.

#### 4.6 BASIC STRUCTURE OF JAVA PROGRAM

Every programming language has some pre-defined code structure and a set of rules known as syntax. If these rules are violated while writing the code, it'll produce an error. It is very important to follow the syntax. Before we start learning how to write a program in Java, let us first understand how a basic structure of a Java program look likes.



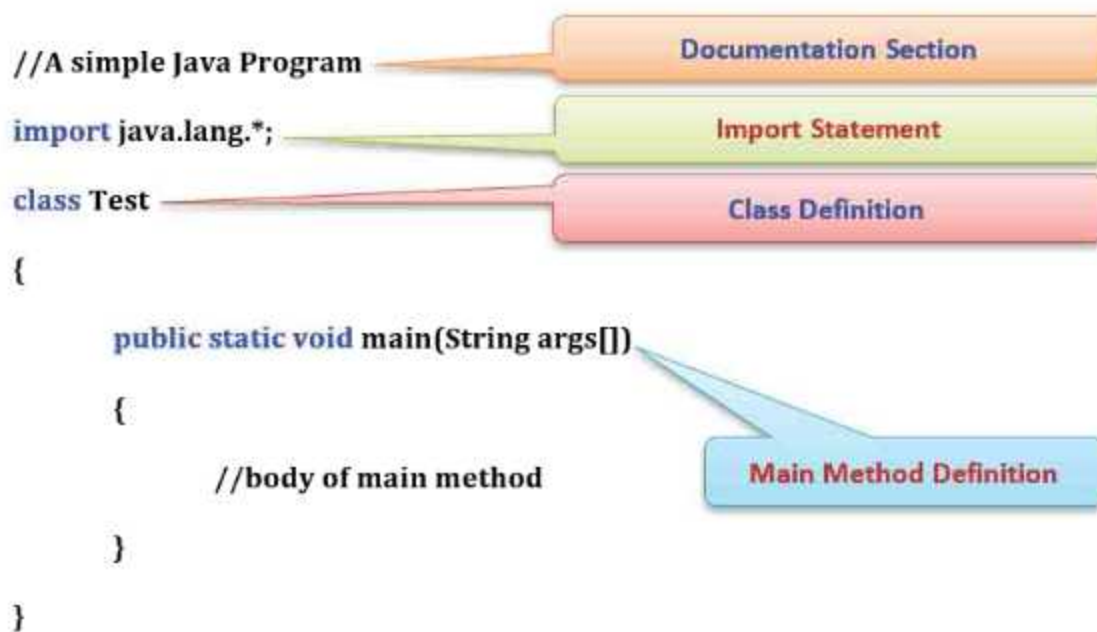


Fig: 4.5 A Simple Java Program Structure

This figure shows the very basic structure of a Java program. A Java program can have more elements in it, such as package statement, interface section etc. We will not discuss about these sections because these topics are beyond the scope of this book. Now, let us have a look on the description of program elements shown in the program-structure:

- ❖ **Documentation Section:** It is used to improve the readability of the program. It consists of comments in Java which make it easier for the programmer to understand it while reviewing or debugging the code. This statement is optional.
- ❖ **Import Statement:** In Java, there are many predefined classes that are organized in packages. If we want to use classes defined in other packages, we have to use import statement in our program.
- ❖ **Class Definition:** Classes are an essential part of any Java program. A Java program may contain several class definitions. Every program in Java will have at least one class with the main method. The class that contains the main method should be declared as public.
  - ❖ **class Test** - This creates a class called Test. As a standard convention, Java uses TitleCase for class names. So, we should make sure that the class name should be in TitleCase (without blank space in the name).
  - ❖ **Braces (Both Opening and Closing Brace)** - The curly braces are used to group the statements together. Curly braces are used to make sure that the statements belong to a class or a method.

- ❖ **main() Method Definition:** In Java, the main method is treated as the entry point of the program. The main method is called by the operating system when the user runs the program.

For Example: `public static void main(String args[])`

- ❖ **public** - When the main method is declared public, it means that it can be accessed outside the declared class as well.
- ❖ **static** - The word static means that we want to access a method without making object of the class within which the method is declared. During program execution we call the main method without creating any objects.
- ❖ **void** - The word void indicates that it does not return any value. The main method is declared as void because it does not return any value.
- ❖ **main** - The main is the user defined method, which is an essential part of any Java program.
- ❖ **String args[]** - It is an array where each element is a string, which is named as args. If we run the Java code through a console, we can pass the input through command line parameter. The main() takes it as an input.

#### 4.7 CREATING AND EXECUTING A SIMPLE JAVA PROGRAM

We need the following two softwares to create our first Java Program

1. The Java Development Kit (JDK): The JDK is a key platform component for building Java applications. Java compiler is the core of Java. It contains JRE + Development Tools.
2. A Text Editor: Any text editor such as Notepad, Notepad++ etc. can be used to write the Source Code for Java Program. In this following example program, we'll use Notepad++. We can use a different text editor like Notepad (a simple text-editor included with the Windows Operating System) or use online java compiler.

For creating and executing a java program, following steps can be used:

**Step 1:** Open Notepad++ (Start button → Notepad++) and type a java program in it and save it (Ctrl+S) with .java extension (for example, Test.java)

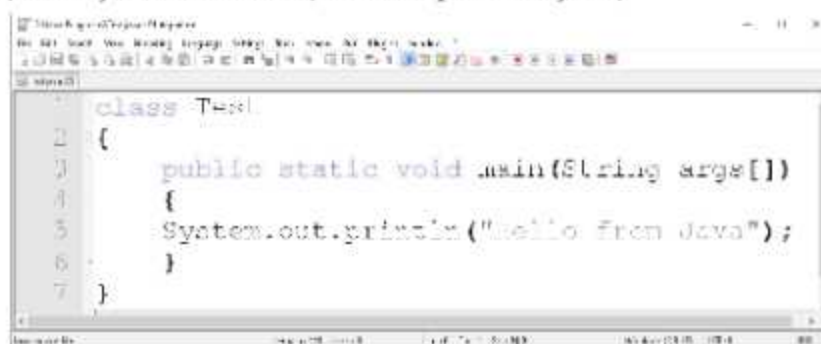


Fig 4.6: Example Java Program in Notepad++

**Step 2 :** Open Command Prompt for current file location by Right Clicking on the Current File Tab of Notepad++ and then click on the "Open Containing Folder in CMD" as shown in the following figure:

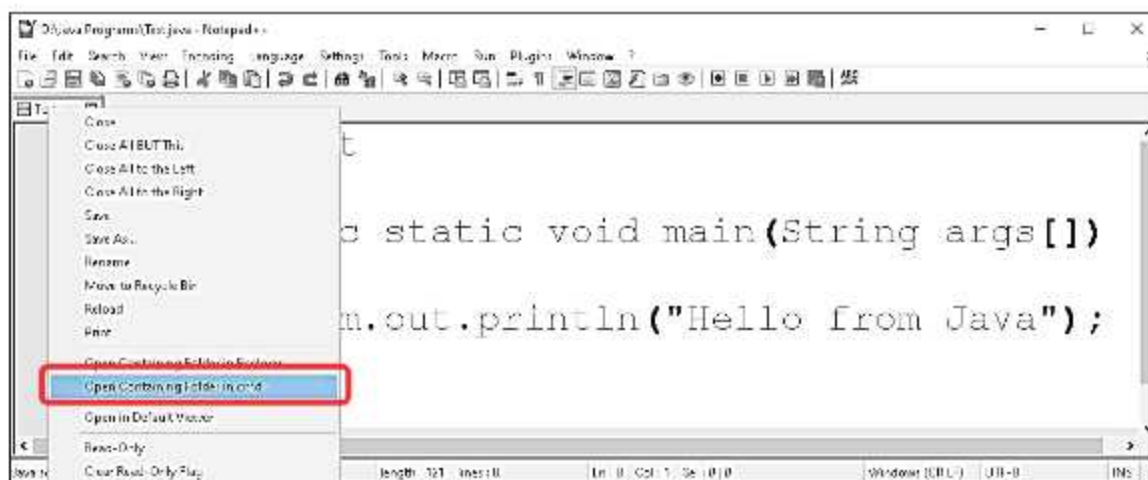


Fig: 4.7 Open Command Prompt for Current file through Notepad++



Fig: 4.8 Command Prompt with location of Current Java File

**Step 3:** Use the following command syntax to compile the Java program:

Syntax:

**Javac filename.java**

Compile the Test.java file as shown below:

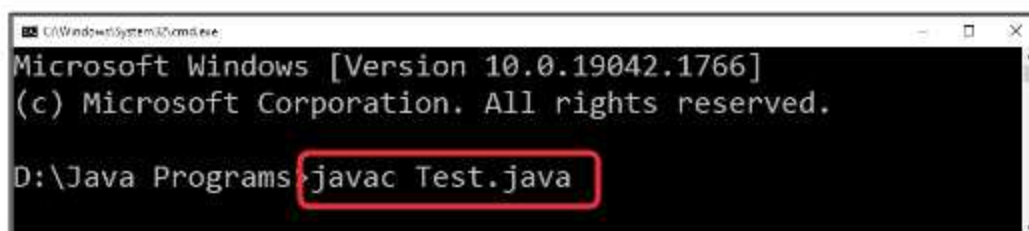


Fig 4.9: Compiling Java Program – Test.java

After typing javac Test.java, press enter key. It will compile the source code of our java program stored in the Test.java file. If program doesn't have any errors, it will show the command prompt in next line as shown in the following figure and generate a .class file



in the same folder. This .class file contains the bytecode of our program that will be executed by the java interpreter in the JVM.

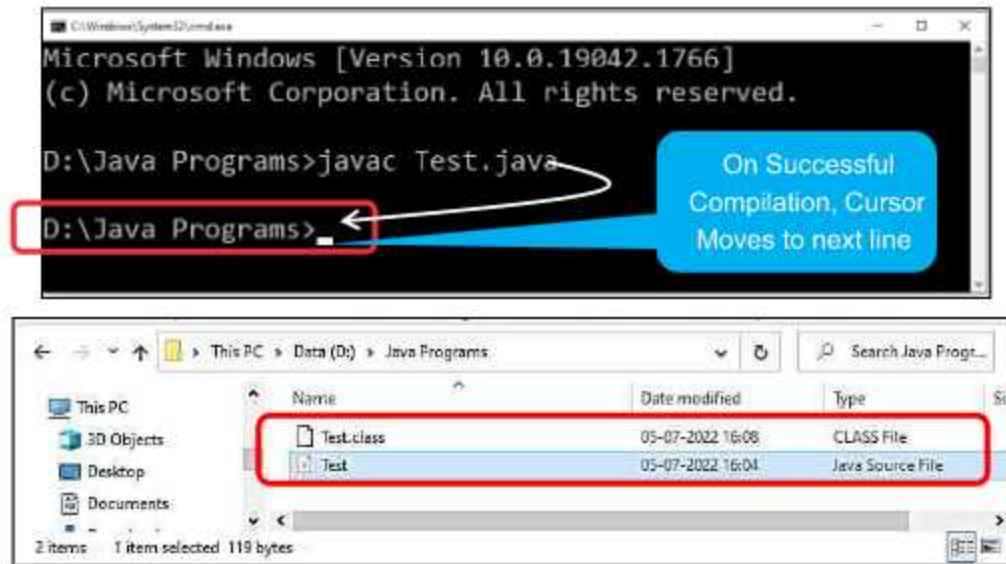


Fig 4.10: Generation of Test.class file (bytecode) after compilation of Test.java file

But, if our program contains any syntax errors, it will show error message/messages with line numbers in the source code file, as shown below:

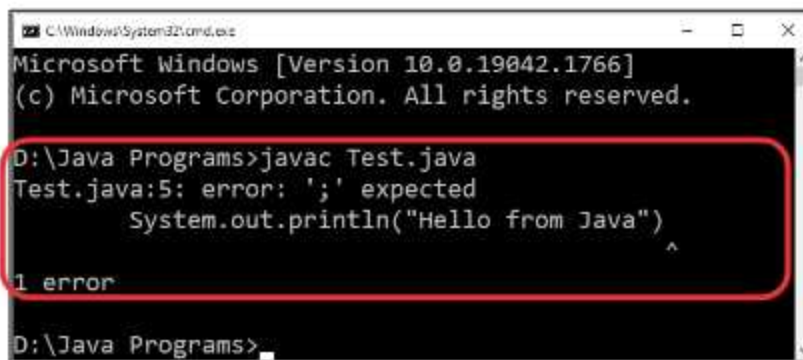


Fig 4.11 Compilation Errors (if exists in the Program)

If we are shown the following error message, we have to modify the path settings of our system.

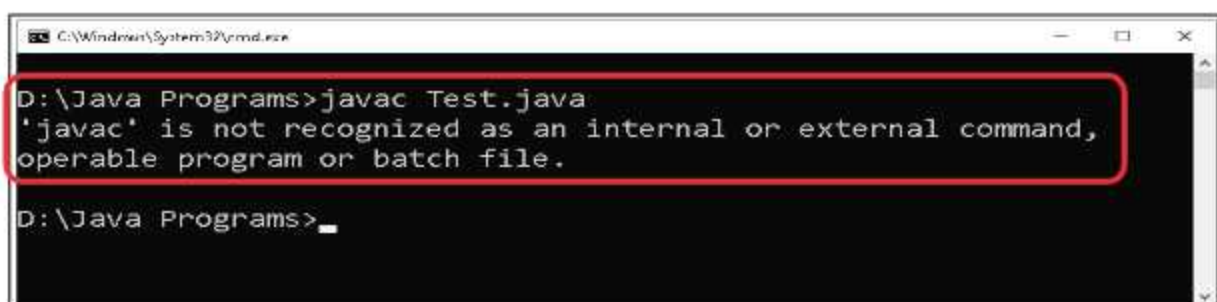


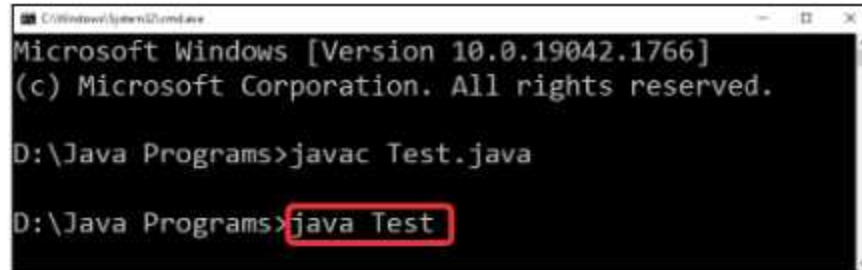
Fig 4.12: Path Setting issue of Java Installation

**Step 4:** Use the following command syntax to run the Java program:

Syntax:

java classname (Note: use the same casing as given in the class name)

Following figure shows how to execute Test.java program after successful compilation.



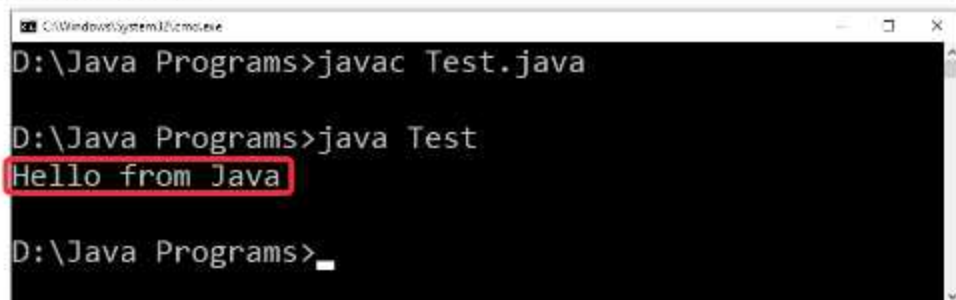
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.1766]
(c) Microsoft Corporation. All rights reserved.

D:\Java Programs>javac Test.java

D:\Java Programs>java Test
```

Fig 4.13: Executing java program after successful compilation

After Typing java Test, press Enter key from Keyboard and it will display the program output as shown below:



```
C:\Windows\System32\cmd.exe
D:\Java Programs>javac Test.java

D:\Java Programs>java Test
Hello from Java

D:\Java Programs>
```

Fig: 4.14: Output of Test.java program after execution

The process of compiling and executing a java program is shown in the following figure:

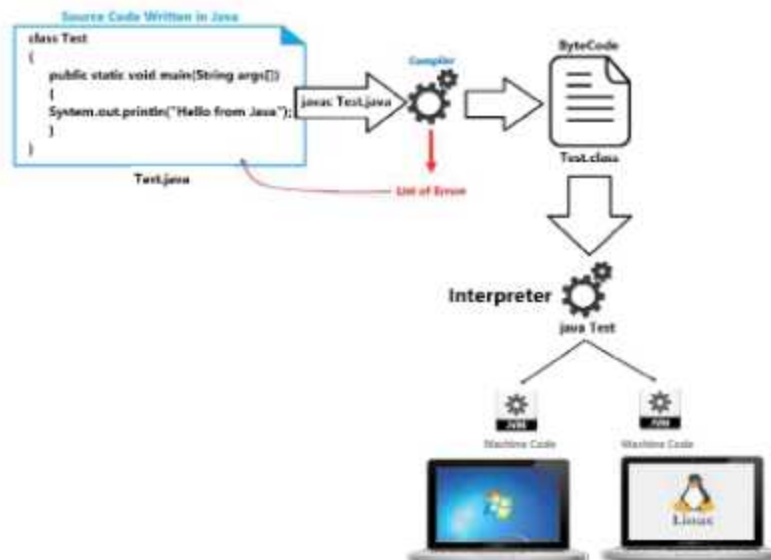


Fig: 4.15 Compilation and Execution Process of a Java Program

## 4.8 BASIC ELEMENTS OF JAVA PROGRAMMING

For communication among human beings, we use natural languages like English, Hindi, and Punjabi etc. But to communication with computers, we can use only those languages which a computer system can understand. These languages are called Programming Languages. Java is a programming language. As we have to learn any natural language before using it, similarly we must also have to learn Java programming language for communication with the computers. Learning programming languages are very similar to learn any natural language like Hindi, Punjabi etc. Instead of learning directly, how to write programs, we must first know what characters, numbers, symbols are used in Java, then using these, how different tokens (like words in English) are constructed, finally we learn how these tokens are combined to form instructions (like sentences in natural languages). A group of instructions would be combined later on to form a program (like a paragraph). Now, let's begin our journey to learn Java Programming language by understanding its character set.

**4.8.1 Character Set :** It is the first step for learning any language whether it is Natural Language or a Computer Programming Language. We can learn any language only if we know which characters and symbols are allowed to use in that language. So, before learning Java, we must be familiar with the characters and symbols used in the Java language. Unlike other programming languages like C and C++, Java uses a standard known as Unicode to define characters. Unicode is a 16-bit character code set which defines all of the characters available in all human languages like: English, Hindi, French, German, Chinese, Japanese etc. This makes Java a worldwide programming language.

**4.8.2 Tokens :** Tokens are like words and punctuation-marks in English language. In any programming language, like Java, a program is made up of tokens. Tokens are the smallest individual units in a program. When a program is compiled, the compiler scans the source code and parses it into tokens to find the syntax errors. Java tokens are broadly classified into keywords, identifiers, literals (constants), operators and punctuators (special symbols):

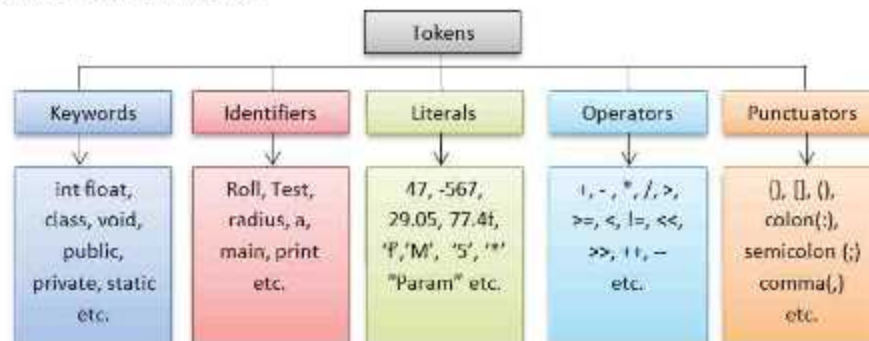


Fig: 4.16 Types of Tokens in Java



**4.8.2.1 Keywords:** Keywords are also called the Reserve Words. These words are predefined in Java Compiler. Meaning of these words is predefined. They are used for special purposes for which they have been defined and hence cannot be used as user-defined names (identifier). We cannot change their meaning. These keywords can be used wherever they are required in the program. All the keywords in Java programs must be written in lower-case only. As Java is case-sensitive language, so if we write these keywords in upper-case in a program, it will display compile errors. Because they would be considered as Identifier (A case sensitive language is a language which considers lower case and upper-case alphabets as different elements.) there are various keywords in Java, which are listed in the following table:

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

**4.8.2.2 Identifiers:** Identifiers are the name given to elements of program such as variables, constants, arrays, methods, classes, interfaces, etc. Every program element must be named to distinguish it from other elements. The name assigned to the elements should be meaningful because it facilitates easy understanding of the program elements. After naming the program elements, they can be identified by their name. For defining names of program elements, some naming rules must be followed. These naming rules are given below:

- Identifiers can consist of upper case and lowercase letters, digits, dollar sign (\$) and an underscore (\_).
- An identifier must begin with a letter, dollar sign or an underscore, i.e. it must not begin with a digit.
- Identifiers are case-sensitive, i.e. an identifier written in uppercase is different from that is in lowercase.
- A keyword cannot be used as an identifier since it is a reserved word and has some special meaning.
- Within a given section of program or scope, each user defined item must have a unique identifier
- Identifiers can be of any length.
- An identifier must not contain white space and other characters such as \*, ; etc.

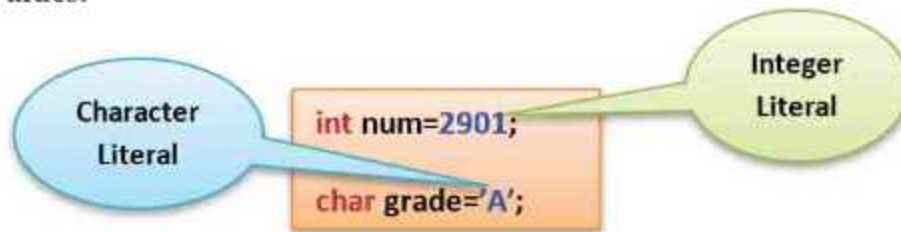
Following are some examples of legal and illegal identifiers:

- ❖ Legal identifiers: MinNumber, total, vk49, hello\_world, \$amount, \_under\_value
- ❖ Illegal identifiers: 49vk, -amount, roll no, house#

According to the convention, all letters should be in lowercase for variables. The underscore is not recommended for the names of variables. Constants (static, final, attributes and enums) should be in all Uppercase letters.

#### 4.8.2.3 Literals:

Literals are also called constant values. These are the fixed values that are normally assigned to program elements. For example: 2901, 47.29f, "Param", 'A' etc. are all constant values.



Based on the type of value, java literals/constants can be categorized broadly into four categories: Numeric constants, Character constants, String constants and Boolean constants:

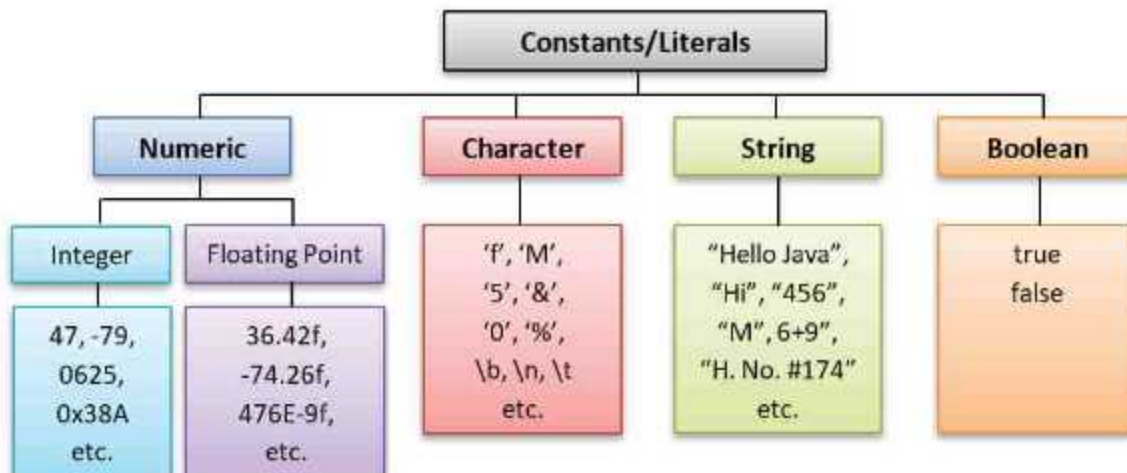


Fig: 4.17 Types of Literals/Constants in Java

##### a. Numeric Literals:

Numeric literals are the numeric values which can be used for numeric calculations. These values consist of sequence of digits (with or without decimal point) that can be either positive or negative. By default, numeric literals are positive. Numeric literals can further be divided into following two types:



**i. Integer Literals :** These are the literals without fractional (decimal) part. These literals consist of digits from 0 to 9 along with positive (+) or negative (-) sign. For example: 56, +26, -96 etc. are the examples of integer literals. To represent the type as long integer, we use L as a suffix with the value, i.e. 3456L. Integer Literals can be represented by three different number systems:

- ❖ **Decimal (base 10):** To indicate a decimal format of a number, put the left most digit as non-zero. For example: 56, 5689 etc.
- ❖ **Octal (base 8):** We can indicate the octal format by a zero-digit followed by the digits 0 to 7. For example: 0625, 034 etc.
- ❖ **Hexadecimal (base 16):** Hexadecimal constants are preceded by 0x or 0X. For example: 0x38A, 0XC4 etc.

**ii. Floating Point Literals :** These are also referred as real numbers. These are the numbers with fractional (decimal) part. These literals consist of digits from 0 to 9 along with positive (+) or negative (-) sign. It also has a decimal point (.) which separates the integral and fractional part of the real number. If integral part does not have any sign, it is considered as the positive real literal. For example: 3.14, +256.5896, -96.14, 36.00 etc. Java has two kinds of floating-point numbers: float and double. We also add a suffix to the floating-point literal as D, d, F or f. The suffix d or D is used to represent the Double-precision floating-point literal and suffix f or F is used to represent Floating-point literal. The type of a floating-point literal defaults to double-precision floating-point.

**For Example:**

65563.554	Double-precision floating-point literal (by default)
34578.343D	Double-precision floating-point literal
45344.01f	Floating-point literal

Floating-point literals can be expressed in two forms: Standard Notation and Scientific Notation.

- ❖ **Standard Notation:** In the standard notation, the floating-point numbers have an integer part and a fractional part, with a decimal point in between the two parts. For Example: 130.35, 41., 0.56, .67, -.67, -7.71 etc.
- ❖ **Scientific Notation:** In the scientific notation, a floating-point literal has a mantissa and an exponent. The mantissa part represents the floating-point number in standard notation and the exponent part specifies a power of 10 by which the number is to be multiplied. The mantissa part and exponent part should be separated by letter E (uppercase or lowercase). For example: A number 231.54 can be represented in exponential form as 2.3154e2 (i.e.



$2.3154 \times 10^3$ ). Here, 2.3154 represents mantissa and the part after letter e represents exponent part which has a base value 10. Exponential form is useful for representing very large numbers as it avoids writing large number of zero's in a number. For Example: 6000000 can be written as 6.0e6

#### **b. Character Literals:**

These are the single character values which usually do not involve in the calculations. These literals are enclosed in single quotes. More than one printable character is not allowed in these literals. Non-printable characters also come in this category of constant literals though two symbols are used in these characters, for example: new line character (`\n` - backslash and a letter n), but yet they are considered to be a single character. Examples of single character literals are: `'A'`, `'g'`, `'7'`, `'+'`, `'$'`, `'\n'`, `'\t'` etc. Values `'AB'`, `'45'` are the invalid examples of single character literals because more than one characters are enclosed in single quotes which are not allowed in the single character constant.

#### **c. String Literals:**

These are the literals which have sequence of any number of characters enclosed in the double quotes. These literals may have the combination of letters, digits, special symbols, and blank space. Examples of these literals are: `"Paramveer"`, `"A"`, `"House#196"`, `"1829"` etc.

#### **d. Boolean Literals:**

The values true and false are also treated as literals in Java programming. When we assign a value to a Boolean variable, we can only use these two values. Unlike C, we can't presume that the value of 1 is equivalent to true and 0 is equivalent to false in Java. We have to use the values true and false to represent a Boolean value.

**Boolean flag = true;**

#### **4.8.2.4 Operators:**

Operators are the symbols which are used to perform some mathematical or logical operations. Operators are used to manipulate values/variables in the program. These values/variables are called operands. Java supports a rich set of built-in operators. All these operators can be classified into three broad categories: unary, binary, and ternary operators. Detailed explanation of these operators has been given in the next chapter.

#### **4.8.2.5 Punctuators or Delemeters**

These are the symbols used as punctuation marks. Each symbol has its different speciality in program. Each symbol is used to denote something special in the program. For example: semicolon (`;`) is used to terminate the statement, comma (`,`) is used as a separator, parenthesis (`()`) are used to represent the methods and constructors, square brackets [`]`] are used to denote arrays, Braces { `}` are used for grouping the statements etc.

### 4.8.3 Comments

Comments are used in the program only for documentation purpose. They are used to describe the code within the program. If comments are added in program then it becomes easier to understand the code. The compiler ignores the comments during compilation. Following styles can be used in java for comments:

- ❖ Block style comments begin with `/*` and terminate with `*/` that spans multiple lines.
- ❖ Line style comments begin with `//` and are used for single line comments.



## Points to Remember

1. Objects are the real-world entities such as a student, person, pen, table, television, computer etc
2. The programming paradigm where everything is represented as an object is known as a Truly Object-Oriented Programming Language.
3. In OOP, code and data are merged into a single individual block called an object.
4. These are the four main principles of the Object-Oriented Programming paradigm: Abstraction, Encapsulation, Inheritance and Polymorphism
5. A class is a grouping of objects that have the same properties and common behaviour.
6. Abstraction defines how a real word entity can be represented into the programming entity.
7. Encapsulation is a process of wrapping the data and methods together in a single unit. This single unit is known as Class
8. Encapsulation allows selective hiding of properties and methods by wrapping them into a single unit called class.
9. Inheritance is a mechanism in which one object acquires all the non-private states and behaviours of a parent object.
10. Many forms of a single object is called Polymorphism
11. There are two types of Polymorphism in the Object-Oriented Programming: Compile-Time Polymorphism and Run-Time Polymorphism
12. Compile Time Polymorphism is achieved using Method Overloading.



13. Run-Time Polymorphism is achieved using Method Overriding.
14. JAVA was developed by James Gosling at Sun Microsystems Inc in the year 1995.
15. When Java program is compiled, it is not compiled into platform specific machine; rather it is compiled into platform independent byte-code.
16. The JDK is a key platform component for building Java applications.
17. JRE contains the parts of the Java libraries required to run Java programs.
18. JVM is an abstract machine. It is a specification that provides a runtime environment in which java bytecode can be executed.
19. Java applications are called WORA (Write Once Run Anywhere).
20. Java Language is Platform Independent. It means program of java is easily transferable from one machine type to other.
21. Tokens are the smallest individual units in a program.
22. Keywords are also called the Reserve Words. These words are predefined in Java Compiler.
23. Identifiers are the name given to elements of program such as variables, constants, arrays, methods, classes, interfaces, etc.
24. Literals are also called constant values. These are the fixed values that are normally assigned to program elements.
25. Operators are the symbols which are used to perform some mathematical or logical operations.
26. They are used to describe the code within the program. The compiler ignores the comments during compilation.

## **Exercise**

### **Q:1 Multiple Choice Questions:**

- i. In OOP, code and data are merged into a single individual block called \_\_\_\_\_.
  - a. Program
  - b. class
  - c. object
  - d. Unit
- ii. \_\_\_\_\_ defines how a real word entity can be represented into the programming entity.
  - a. Abstraction
  - b. Encapsulation
  - c. Polymorphism
  - d. Inheritance



- iii. \_\_\_\_\_ is a mechanism in which one object acquires all the states and behaviours of a parent object.
  - a. Abstraction
  - b. Encapsulation
  - c. Polymorphism
  - d. Inheritance
- iv. \_\_\_\_\_ Polymorphism is achieved using Method Overloading.
  - a. Run-Time
  - b. Execution Time
  - c. Compile-Time
  - d. Loading Time
- v. When Java program is compiled, it is compiled into platform independent \_\_\_\_\_.
  - a. Source Code
  - b. Object Code
  - c. Bit Code
  - d. Byte Code
- vi. \_\_\_\_\_ are the smallest individual units in a program.
  - a. Objects
  - b. Tokens
  - c. Literals
  - d. Classes
- vii. \_\_\_\_\_ are the symbols which are used to perform some mathematical or logical operations.
  - a. Digits
  - b. Operands
  - c. Expressions
  - d. Operators

#### Q:2 Fill in the Blanks:

- i. \_\_\_\_\_ allows selective hiding of properties and methods by wrapping them into a single unit called class.
- ii. Many forms of a single object is called \_\_\_\_\_.
- iii. JAVA was developed by \_\_\_\_\_ at Sun Microsystems Inc in the year 1995.
- iv. \_\_\_\_\_ contains the parts of the Java libraries required to run Java programs
- v. \_\_\_\_\_ is an abstract machine that provides a runtime environment in which java bytecode can be executed.
- vi. \_\_\_\_\_ are the name given to elements of program such as variables, constants, arrays, methods, classes, interfaces, etc.
- vii. The compiler ignores the \_\_\_\_\_ during compilation.

#### Q:3 Short Answer Type Questions:

- i. What are the four main principles of Object-Oriented Programming Paradigm?
- ii. Defines Object.
- iii. What is Inheritance?
- iv. What is Encapsulation?
- v. What do you know about Java?
- vi. What is JVM?

- vii. What are Keywords?
- viii. Define Comments.
- ix. What are Operators?
- x. Why Java programs are known for Platform Independence?

**Q:4 Long Answer Type Questions:**

- i. What is Polymorphism? Write the name of different types of Polymorphism.
- ii. Explain any five features of Java.
- iii. Write the basic structure of Java Program.
- iv. What are Tokens? Write an overview of different types of Tokens.
- v. What is Identifier? Write the naming rules for identifiers.
- vi. What are Literals? Explain different types of Literals.



# Data Types and Operators in Java



## OBJECTIVES OF THIS CHAPTER

- 5.1 Data Types
- 5.2 Variables
- 5.3 Operators
- 5.4 Expressions
- 5.5 Precedence of Operators

### 5.1 DATA TYPES

Data type defines which type of data will be stored in the program elements, such as variables, arrays etc. Data types determine a specific type, range of values and the operations that can be performed on data. Java is a strongly typed language therefore data type of all the variables must be declared during declaration.

Java provides various data types and each data type is represented differently within computer's memory. The Data types provided by Java can be categorized broadly into two types:

- ❖ Primitive Data Types
- ❖ Non-Primitive Data Types

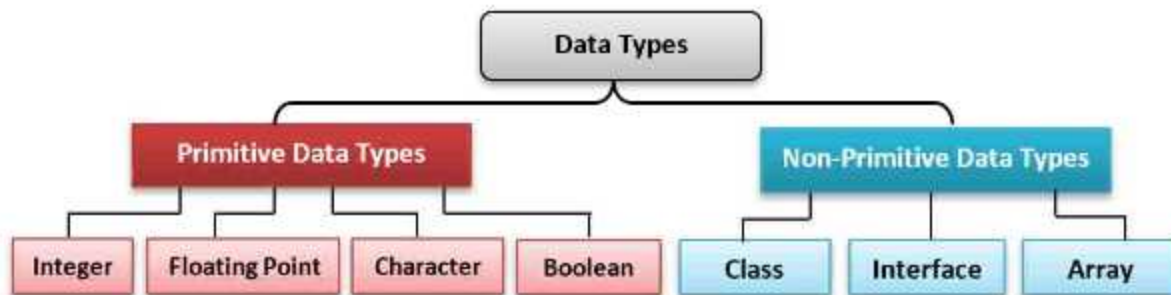


Fig 5.1: Classification of Data Types in Java

**5.1.1 Primitive Data Types :** Primitive data types are also known as built-in data types. Java has four main primitive data types built into the language.

1. **Integer:** byte, short, int and long



- II. **Floating Point:** float and double
- III. **Character:** char
- IV. **Boolean:** boolean variable with a value of true or false

These data types have been described as following in detail.

**I. Integer Type:** Integer data type is used to store integer values (numbers without fractional part) like 49, 174, +2901, -54896 etc. Java support four types of integer data type: byte, short, int and long. All of these data types are signed as positive or negative. There is no concept of unsigned integer in java. The default value for these types is 0. These types have different storage capacities.

Following table shows the description about the memory requirements and range of values for the integer type data types:

Data Type	Storage Size	Range of Data	
byte	1 byte	$-2^7$ to $+2^7 - 1$	OR (-127 to +128)
short	2 bytes	$-2^{15}$ to $+2^{15} - 1$	OR (-32,768 to +32,767)
int	4 bytes	$-2^{31}$ to $+2^{31} - 1$	OR (-2,147,483,648 to +2,147,483,647)
long	8 bytes	$-2^{63}$ to $+2^{63} - 1$	OR (-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807)

Table: 5.1 Summary of Integer Data Types

We can select an appropriate data type for a variable depending on the type and range of integer value required.

**II. Floating-Point Type :** Floating-point data type is used to store real numbers such as 3.14, 74.5884569, +29.05, -524836.458 etc. Java supports two floating point data types: float and double. Following table shows the description about the memory requirements and range of values for the floating-point type data types:

डेटा टाइप	स्टोरेज साइज	डेटा सी रेंज
float	4 bytes	3. 4E-38 to 3. 4E+38
double	8 bytes	1.7E-308 to 1. 7E+308

Table: 5.2 Summary of Floating-Point Data Types

Data type float represents the a single-precision number while double type represents the double-precision number. Single-precision number occupies lesser space than double precision. Single-precision value becomes inaccurate when values are large. Therefore, double type floating-point is the best choice when we need to store large values. For example: If we want to store the value of marks obtained by students, float data type can be used. Similarly, when we are working with mathematical functions like sin(), cos(), sqrt() etc, we should use double data type. The default value of float data type is 0.0f and double data type is 0.0d.

**III. Character Data Type :** Character data type is used to store a single Unicode

character enclosed in single quotes. To represent the character data type, char keyword is used. It takes 2 bytes (16 bits) of memory space. The range of char data type is 0 to 65535. The default value of char data type is '\u0000'.

Data Type	Storage Size	Range of Data
char	2 bytes	0 to 65535

Table: 5.3 Summary of Character Type Data Types

**IV. Boolean Data Type :** Boolean data type is used to store boolean values for the program variables. This data type accepts only two values: true or false. The keyword boolean is used to denote the boolean data type. It specifies 1 bit of information but its size can't be defined precisely. The default value of boolean data type is false.

**5.1.2 Non-Primitive Data Types :** Non-Primitive data types are also known as user-defined data types or reference types. These data types are derived from the primitive data types. Classes, Interfaces and Arrays are the Non-Primitive data types in Java.

## 5.2 VARIABLES

A Variable is an identifier that represents a memory location. This memory location is used to store data/value. Data/value stored in these locations can be accessed using the variable name. Variables allow us to change their values during execution. Every variable must have a data type, which specifies the size and type of value that can be stored in the variable. To use the variables in the program, they must be declared.

**5.2.1 Variable Declaration :** Java is a strongly typed language. It means we must declare variables before using them in the program. If we use variables without declaring them, compiler will generate a syntax error. A variable declaration consists of two parts: data type and variable name (identifier). Following syntax is used to declare variable in the java program:

**data\_type variable\_name;**

Here, **data\_type** tells the compiler what type of value is going to be stored in the variable, and **variable\_name** is the valid identifier which tells the compiler about the name of the variable which will be used to refer to the value stored in the variable. Consider the following example of variable declaration:

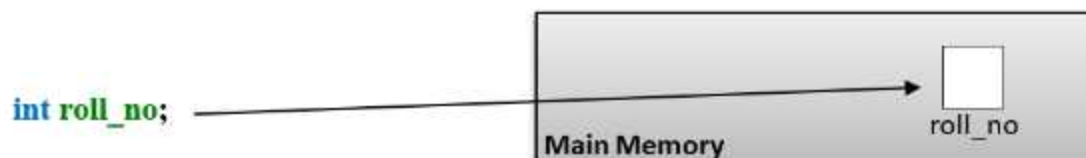


Fig:5.2 Memory allocation to the Variable

Here, **int** represents the integer data type which allocated 4 bytes of memory to identifier **roll\_no**. This identifier name is used to access the value stored in the variable,



whenever required. Here, the variable **roll\_no** can hold only integer value. We can also declare multiple variables of same type by using the comma separator. For example:

```
int a, b, c;
```

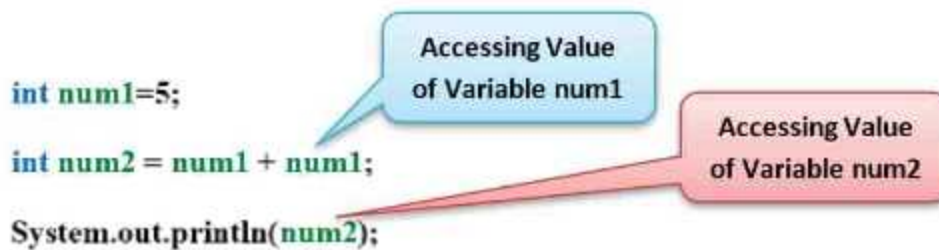
**5.2.2 Variable Initialization:** Declaration of a variable only allocates memory to it but it does not store any data at the time of declaration. We can assign a value to a variable during its declaration, which is called variable initialization. For example:



Table: 5.3 Value stored in the Variable

This assigned value can be changed later at any time because a variable allows us to change its values at any time during execution.

**5.2.3 Accessing Value of a Variable:** To access/use the value stored in the variable, we have to use its name. For example:



In above example, we declared and initialized two variables `num1` and `num2`. Value of `num1` variable has been used for calculating the value of `num2`, i.e. `int num2=num1+num1`. To show the value of `num2`, we use `println()` method of java.

## 5.3 OPERATORS

**Operators** are the symbols which are used to perform some specific type of operation on data. For example: `+` symbol is used to perform addition, `*` is used to perform multiplication, `>=` is used to perform comparison etc. Here `+`, `*` and `>=` are the operators to perform different types of operations. After performing the operations, all operators produce a value.

To perform any type of operation, we require Operands. **Operands** are the data items on which operators can perform operations. These operands can be either variables or constant values. Consider the following example:

```
a + 5 * 10
```

In this example, `+` and `*` are the operators which perform the operation on variable 'a' and constant values 5 and 10. Here the variable 'a' and constant values 5 and 10 are called the Operands. A valid combination of operators and operands is known as **Expression**.



Depending on the function performed by the operator, Java operators can be classified into various categories: Arithmetic Operators, Relational Operators, Logical Operators, Assignment Operators, Bitwise Operators, Increment & Decrement Operators, Conditional Operators, and Special Operators.

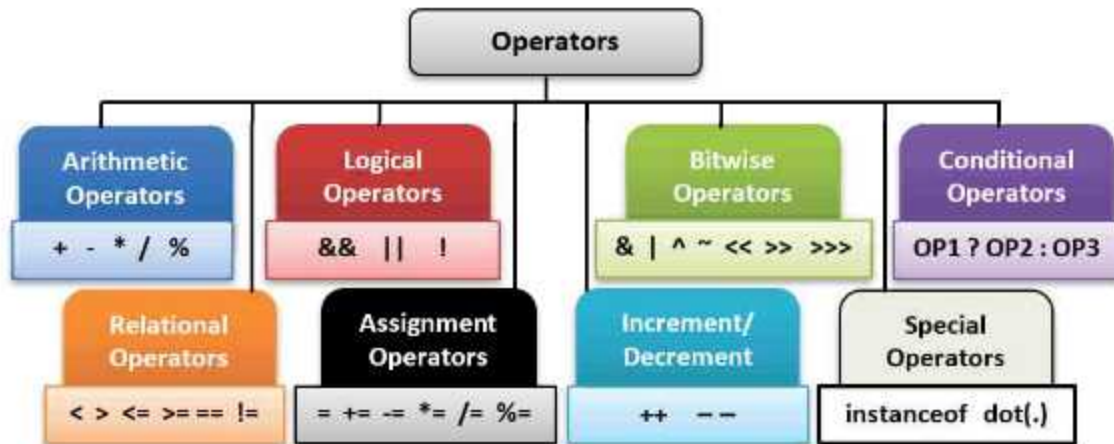


Fig: 5.4 Classification of Operators in Java

### 5.3.1 Arithmetic Operators:

Arithmetic operators are used to perform arithmetic operations such as: addition, subtraction, multiplication, division etc. There are five arithmetic operators in 'Java'. All these operators are binary operators because all these operators require two operands to perform their operations. Following table shows the list and working of all these operators:

Name	Operator	Description	Examples of Operators on	
			Integer Values	Real Values
Add	+	Used to perform Addition of numbers.	2+4 → 6	2.0+4.0 → 6.0
Subtract	-	Used to perform subtraction or used as any unary minus.	6-2 → 4	6.0-4.0 → 2.0
Multiply	*	Used to perform multiplication of numbers.	7*2 → 14	7.0*2.0 → 14.0
Divide	/	Used to perform division of numbers.	5/2 → 2	5.0/2.0 → 2.5
Modulus	%	Used to get remainder value after division of numbers.	7%4 → 3	5.0%2.0 → 1.0

Fig: 5.4 Arithmetic Operators

Following program shows how to use arithmetic operators in Java Programming:

#### Program 5.1: Use of Arithmetic Operators in Java

```
1 class Test1
2 {
3     public static void main(String args[])
4     {
5         float num1=5.0f, num2=2.0f, result;
6         result=num1+num2;
7         System.out.println("Result of Addition is "+result);
8         result=num1-num2;
9         System.out.println("Result of Subtraction is "+result);
10        result=num1*num2;
11        System.out.println("Result of Multiplication is "+result);
12        result=num1/num2;
13        System.out.println("Result of Division is "+result);
14        result=num1%num2;
15        System.out.println("Result of Modulus is "+result);
16    }
17 }
```

#### Compilation, Execution and Output of Program 5.1:

```
C:\Windows\System32\cmd.exe
D:\Java Programs>javac Test1.java

D:\Java Programs>java Test1
Result of Addition is 7.0
Result of Subtraction is 3.0
Result of Multiplication is 10.0
Result of Division is 2.5
Result of Modulus is 1.0

D:\Java Programs>
```

In the program 5.1, we declare three variables: num1, num2, and result of floating type. Variables num1 and num2 are initialized with float values 5.0f and 2.0f respectively. Then the calculation result of all arithmetic operators is stored in the result variable which has been displayed using the System.out.println() method of java. We can apply these operators on integer and floating-point operands.

#### 5.3.2 Relational Operators

Relational operators are also called comparison operators. These operators are used to test the relationship between operands. In other words, these operators are used to compare values. After comparison, these operators return boolean value either **true** or **false**. All the relational operators present in Java language are of binary type. It means these operators require two operands to perform their operation. There are 6 relational operators in Java which are given below in the table with examples:

Name	Operator	Description	Example	Result
Equals to	<code>==</code>	Used to check whether two values are equal	<code>4==5</code> <code>5==5</code>	False True
Not Equal to	<code>!=</code>	Used to check whether two values are not equal	<code>4!=5</code> <code>4!=4</code>	True False
Greater than	<code>&gt;</code>	Used to check whether the first value is greater than second	<code>4&gt;5</code> <code>5&gt;4</code>	False True
Less than	<code>&lt;</code>	Used to check whether the first value is less than second	<code>4&lt;5</code> <code>5&lt;4</code>	True False
Greater than or equal to	<code>&gt;=</code>	Used to check whether first value is greater than or equal to second value	<code>5&gt;=5</code> <code>6&gt;=8</code> <code>10&gt;=5</code>	True False True
Less than or equal to	<code>&lt;=</code>	Used to check whether first value is lesser than or equal to second value	<code>4&lt;=5</code> <code>4&lt;=2</code> <code>4&lt;=4</code>	True False True

Table 5.5 Relational Operators

### Program 5.2: Use of Relational Operators in Java

```

1 class Test2
2 {
3     public static void main(String args[])
4     {
5         int num1=5, num2=2;
6         boolean result;
7         System.out.println("Value of num1 is "+num1);
8         System.out.println("Value of num2 is "+num2);
9         result=num1==num2;
10        System.out.println("Result of num1==num2 is "+result);
11        result=num1!=num2;
12        System.out.println("Result of num1!=num2 is "+result);
13        result=num1>num2;
14        System.out.println("Result of num1>num2 is "+result);
15        result=num1<num2;
16        System.out.println("Result of num1<num2 is "+result);
17        result=num1>=num2;
18        System.out.println("Result of num1>=num2 is "+result);
19        result=num1<=num2;
20        System.out.println("Result of num1<=num2 is "+result);
21    }
22 }

```

### Compilation, Execution and Output of Program 5.2:

```

D:\Java Programs>javac Test2.java

D:\Java Programs>java Test2
Value of num1 is 5
Value of num2 is 2
Result of num1==num2 is false
Result of num1!=num2 is true
Result of num1>num2 is true
Result of num1<num2 is false
Result of num1>=num2 is true
Result of num1<=num2 is false

D:\Java Programs>

```



Usually relational operators are used in the expression that consists of control statements such as branching, looping and jumping statements. These statements have been explained in the next chapter of this book.

### 5.3.3 Logical Operators:

Logical operators are also called Boolean Operators. These operators are used to make compound relational expressions. In other words, we can say that these operators are used when we want to test more than one condition at a time. There are 3 Logical operators in Java: 'Logical AND', 'Logical OR' and 'Logical NOT'. Here, 'Logical AND' and 'Logical OR' are the binary operators whereas 'Logical NOT' is unary operator. Two operands are required for 'Logical AND' and 'Logical OR' to perform their operations while one operand is required for 'Logical NOT' to perform its operation.

All Logical Operators return boolean value either true or false. The result of a logical AND operator will return true only if both operands are true, whereas the result of a logical OR operator will be true if either operand is true or if both operands are true. Logical NOT operator returns true only when its operand is false. Following table shows the list and working of all Logical Operators used in Java language with appropriate examples:

Name	Operator	Description	Example	Explanation and Result
AND	&&	Returns true only if both operands are true otherwise it returns false	3>5 && 4>5 3>5 && 4<5 3<5 && 4>5 3<5 && 4<5	False && False → False False && True → False True && False → False True && True → True
OR		Returns true if at least one of its operands is true otherwise it returns false	3>5    4>5 3>5    4<5 3<5    4>5 3<5    4<5	False    False → False False    True → True True    False → True True    True → True
NOT	!	Returns true only when its operand is false otherwise it returns false	!(3<5) !(3>5)	!(True) → False !(False) → True

Table 5.6 Logical Operators

### Program 5.3 Use of Logical Operators in java

```

1 class Test3
2 {
3     public static void main(String args[])
4     {
5         int num1=15, num2=10, num3=5;
6         boolean result;
7         System.out.println("Value of num1 is "+num1);
8         System.out.println("Value of num2 is "+num2);
9         System.out.println("Value of num3 is "+num3);
10        result=num1>num2 && num2<=num3;
11        System.out.println("Result of AND (num1>num2 && num2<=num3) is "+result);
12        result=num1>num2 || num2<=num3;
13        System.out.println("Result of OR (num1>num2 || num2<=num3) is "+result);
14        result=! (num1==num2);
15        System.out.println("Result of NOT !(num1==num2) is "+result);
16    }
17 }

```

### Compilation, Execution and Output of Program 5.3:

```
C:\Windows\System32\cmd.exe
D:\Java Programs>javac Test3.java

D:\Java Programs>java Test3
Value of num1 is 15
Value of num2 is 10
Value of num3 is 5
Result of AND (num1>num2 && num2<=num3) is false
Result of OR (num1>num2 || num2<=num3) is true
Result of NOT (!(num1==num2)) is true

D:\Java Programs>
```

#### 5.3.4 Assignment operators:

These Operators are used to assign or store values in variables etc. The symbol of assignment operator is =. Consider the following examples which show how to use assignment operator in Java programs:

```
int a = -2;           // assigns -ve value (-2) to the variable.
int b = 5;            // assigns value (5) to the variable.
int c = a + b;        // assigns the result of expression to the variable.
int a = a + 10;       // self-assignment of a variable.
```

Left hand side of assignment operator must be a valid identifier which represents a memory location. We cannot put the expression on the left side of the assignment operator. For example, following assignment statement would be invalid:

```
a + b = c;           // Invalid assignment statement
```

With the help of assignment operator, several variables can be assigned a common value. Consider the following example:

```
a = b = c = 5;       // value 5 will be assigned to three variables a, b and c
```

Assignment operators can also be used as **compound or shorthand assignment operators**. Shorthand assignment operators are useful for self-assignment statements. Following table shows the examples of shorthand assignment with arithmetic operators:

Let's assume            `int a=5;`

Shorthand Operator	Example for Shorthand Assignment	Equivalent Self-Assignment	Result
<code>+=</code>	<code>a+=2</code>	<code>a = a + 2</code>	<code>a=7</code>
<code>-=</code>	<code>a-=2</code>	<code>a = a - 2</code>	<code>a=3</code>
<code>*=</code>	<code>a*=2</code>	<code>a = a * 2</code>	<code>a=10</code>
<code>/=</code>	<code>a/=2</code>	<code>a = a / 2</code>	<code>a=2</code>
<code>%=</code>	<code>a%=2</code>	<code>a = a % 2</code>	<code>a=1</code>

Table 5.7: Shorthand Assignment for Arithmetic Operators



Assignment operator = and the equality (equal to) operator == both are different types of operators. The assignment operator is used to assign a value to an identifier, whereas the equality operator is used to determine if two operands have the same value. These two operators cannot be used in place of each other.

### 5.3.5 Bitwise Operators:

Bitwise operators are used for performing bit level operations. They can be used with any integral type (char, short, int, etc.). Following table shows a summary of bitwise operators:

Assume **int A=60** and **int B=13** then:

Operator	Description	Example
& (Bitwise AND)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
(Bitwise OR)	Binary OR Operator copies a bit if it exists in either operand.	(A   B) will give 61 which is 0011 1101
^ (Bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (Bitwise Compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A ) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (Left Shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (Right Shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (Zero Fill Right Shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Table 5.8: Bitwise Operators

### 5.3.6 Increment and Decrement Operators:

These are the unary operators. The symbol ++ is used for increment operator while symbol - - used for decrement operator. The increment operator causes its operand to increase by one whereas the decrement operator causes its operand to decrease by one. The operand used with each of these operators must be a single variable. These operators cannot be applied directly on the constant values.

For example:

int x=10;      (x is an integer variable that has been assigned a value of 10)

Following expression causes the value of x to be increased to 11



`++x;` (which is equivalent to `x = x+1`)

Similarly, following expression causes the original value of `x` to be decreased to 9:

`--x;` (which is equivalent to `x = x-1`)

If we use `10++` or `--10`, it will be a wrong statement as we have already studied that these cannot be applied directly on the constant values.

The increment and decrement operators can each be utilized in two different ways. It depends on whether the operator is written before or after the operand:

- ❖ Prefix increment and decrement
- ❖ Postfix increment and decrement

If the operator precedes the operand, then the value of operand will be altered before it is used for its intended purpose within the program. This is called **pre-increment/decrement**. If, however the operator follows the operand then the value of the operand will be changed after it is used. This is called **post-increment/decrement**.

For example:

If the value of `x` is initially 10, it can be increased by two methods:

`y = ++x;` (pre-increment)

Here, at first, the value of `x` will be incremented to 11 and then this incremented value of `x` will be assigned to variable `y`, i.e. `y` will also get value 11 (i.e. `x=11` and `y=11`)

`y = x++;` (post-increment)

Here, at first, the value of `x` will be assigned to variable `y` (i.e. `y` will get value 10) then value of `x` will be incremented to 11 (i.e. `y=10` and `x=11`)

Similarly, decrement operator can be used. For example:

`y = --x;` (pre-decrement)

Here, first of all, the value of `x` will be decremented to 9 and then this decremented value of `x` will be assigned to variable `y`, i.e. `y` will also get value 9 (i.e. `x=9` and `y=9`)

`y = x--;` (post-decrement)

Here, first of all, the value of `x` will be assigned to variable `y` (i.e. `y` will get value 10) then value of `x` will be decremented to 9 (i.e. `y=10` and `x=9`)

### 5.3.7 Conditional Operator (?:)

It is a ternary operator. There is one and only one ternary operator in 'Java' language. This operator requires three operands to perform its operation. Ternary operator in Java is represented using `?:` symbols. The syntax for using this operator is given below:

`exp1?exp2:exp3;`

Here, `exp1` must be a conditional expression which produces a result either true or false. If the value of `exp1` is true then the `exp2` will perform its function otherwise `exp3` will perform its function. Consider the following example:

```
a=5;
b=10;
c=a>b ? a : b;
```

Here, the expression  $a > b$  will produce false (0) result (Operand1/exp1), therefore value of b (Operand3/exp3) will be stored in variable c. Variable a (Operand2/exp2) will not do any function because it will perform its function only if exp1 is true (1).

#### Program 5.4 Use of Conditional Operator in java

```
1 class Test4
2 {
3     public static void main(String args[])
4     {
5         int num1=15, num2=10;
6         int result;
7         System.out.println("Value of num1 is "+num1);
8         System.out.println("Value of num2 is "+num2);
9         result = num1>num2 ? num1 : num2;
10        System.out.println("Largest Number is "+result);
11    }
12 }
```

#### Compilation, Execution and Output of Program 5.4:

```
D:\Java Programs>javac Test4.java

D:\Java Programs>javac Test4.java

D:\Java Programs>java Test4
Value of num1 is 15
Value of num2 is 10
Largest Number is 15

D:\Java Programs>
```

#### 5.3.8 Special Operators:

Java provides some special operators such as instance of and member selection operators:

- ❖ **instanceof operator:** This operator is used to check whether the object belongs to a particular class or not. It returns either true or false value depending on whether the object on left side of expression is an instance of the object on the right side. For example:  
mango instanceof fruit  
This statement returns true if the object mango belongs to class fruit, otherwise it returns false.
- ❖ **Member Selection Operator:** It is also known as dot(.) operator. This operator is used to access the variables and methods of a class using its objects:

```
Object.variable;           // accessing variable of a class through object
Object.method();           // accessing method of a class through object
```

## 5.4 EXPRESSIONS

An expression is like a formula in mathematics. An **expression** can be any valid combination of operators and operands. A valid combination is such a combination that confirms to the syntax rules of Java language. A valid expression is also known as well-formed-expression. An expression after evaluation always returns a single value. This result can be used in the Java programs. Expressions can be as simple as a single value and as complex as a large calculation. Consider the following examples:

```
x=2.9;
```

It is a simple expression where = operator is used with operands x and 2.9

```
x=2.9*y+3.6>z-(3.4/z);
```

It is a somewhat complex expression which consists of many operators and operands. Here, =, \*, +, >, - and / are the operators and x, 2.9, y, 3.6, 3.4 and z are the operands.

Now, consider the following combination of operators and operands which do not form a valid combination to be an expression:

```
x+y=z;
```

The above combination of operators and operands do not form a valid expression, although we are using valid operators and operands in the above example. But this combination of operators and operands do not follow the syntax rules of Java language to be a valid expression. Left side of = operator must represent a valid memory location (identifier) to store value of z. Here, x+y cannot be a valid identifier, because a valid identifier cannot have special character other than underscore (as we learnt in the previous chapter).

All Java expressions can be categorized broadly into following types:

### 5.4.1 Numerical Expressions:

These expressions are used to perform numerical calculations. These expressions always return a numerical value after evaluating operators and operands. Consider the following examples:

```
4+3*2
```

```
3.2-7.8
```

After evaluation of above given numerical expression, a numerical value 10 and -4.6 will be produced.

### 5.4.2 Logical or Conditional Expressions:

These expressions are used to perform logical or conditional operations. These expressions always return one of two possible values: either true or false. Consider the following examples:

```
14>6
```

```
15<=6
```



After evaluating above conditional expressions, we receive true for the first expression and false for the second expression.

## 5.5 PRECEDENCE/HIERARCHY AND ASSOCIATIVITY OF OPERATORS

The order or priority in which various operators in an expression are evaluated is known as **precedence**. Operators with a higher precedence are carried out before operators having a lower precedence. In simple words, the sequence of evaluation of operators in which they are applied on the operands in an expression is called the **Precedence of Operators**. For example, division is performed before the subtraction as division operator has higher precedence than the subtraction operator. The natural order can be altered by making use of parentheses ().

Precedence of commonly used operators in decreasing order and their Associativity is given below:

S.N.	Common Operators	Associativity
1.	.      ( )    [ ]	Left to Right
2.	-      ++    --    !      ~      (type) casting	Right to Left
3.	*      /      %	Left to Right
4.	+      -	Left to Right
5.	<      <=    >      >=    instanceof	Left to Right
6.	==      !=	Left to Right
7.	&&	Left to Right
8.		Left to Right
9.	? :	Right to Left
10.	=      *=      /=      %=-    +=      -=	Right to Left

Consider the following example which illustrates how arithmetic expressions are evaluated using operator's precedence:

a = 5 * 4 / 4 + 8 - 9 / 3;	(* is evaluated)
a = 20 / 4 + 8 - 9 / 3;	(/ is evaluated)
a = 5 + 8 - 9 / 3;	(/ is evaluated)
a = 5 + 8 - 3;	(+ is evaluated)
a = 13 - 3;	(- is evaluated)
a = 10;	result of expression

The order in which operators of the same precedence are evaluated is known as **Associativity**. For example: addition and subtraction operators have the same precedence. However, addition and subtraction may be performed on the expression depending upon the order of their occurrence. The associativity of an operator can be

either from left to right or from right to left. The operators with left to right associativity are evaluated from the left end of the expression while the operators with right to left associativity are evaluated from right end of the expression. Consider the following examples:

For example:

```
a = b = c = 8;
```

The assignment operator has right to left associativity. It means that the value 8 is assigned to c, then c is assigned to b, and at last b is assigned to a.

## 5.5 TYPE CONVERSION

The value of an expression can be converted to a different data type in JAVA, if desired. When value of one type is converted into some other type, it is called Type Conversion. Operands that differ in type may undergo type conversion before the expression takes on its final value. There are two ways of type conversions in JAVA:

1. Implicit Conversion or Widening Conversion
2. Explicit Conversion or Narrowing Conversion

### 5.5.1 Implicit Conversion or Widening Conversion:

This type of conversion is **automatic**. For this type of conversion, we use assignment (=) operator. This type of conversion is used when operand having lower data type is converted into higher data type. If both the data types are compatible and the data type of target variable is large enough to store the value of the source variable, java automatically converts the source type into target type. There is no loss of information in this type of conversion. For example, value of int data type can be assigned to a variable of double data type since double type is larger than int type. Consider the following example for automatic conversion:

```
double n;  
n = 5;
```

In this example, implicit conversion takes place, as integer type value (5) will automatically be converted into double type value so that it can be stored in the double type variable n i.e. value of variable n will become 5.0d

### 5.5.2 Explicit Conversion or Narrowing Conversion or Type Casting:

If the target type is smaller than the source type, conversion cannot be performed automatically. For example, value of double type cannot be stored in the int type variable. For such conversion, java provides a mechanism known as **type casting**. This is forceful conversion. For this type of conversion, we use **caste operator**. There may or may not be any loss of information in this type of conversion.

The syntax for this type of casting is:

```
(data type) variable or expression
```

The name of data type into which the conversion is to be made is enclosed in parentheses and placed directly to the left of the value to be converted.



For example:

```
double n=45.5;
```

```
int a=(int) n;
```

In this example, type casting is performed to convert double data type to int data type. Hence, value of integer type variable 'a' becomes 45, i.e. a=45



## Points to Remember

1. Data types determine a specific type, range of values and the operations that can be performed on data.
2. Primitive data types are also known as built-in data types.
3. Java support four types of integer data type: byte, short, int and long.
4. Floating-point data type is used to store real numbers such as 3.14, 74.5884569, +29.05, -524836.458 etc.
5. Character data type is used to store a single Unicode character enclosed in single quotes.
6. Boolean data type is used to store boolean values for the program variables. This data type accepts only two values: true or false.
7. Non-Primitive data types are also known as user-defined data types or reference types.
8. A Variable is an identifier that represents a memory location.
9. We can assign a value to a variable during its declaration, which is called variable initialization.
10. **Operands** are the data items on which operators can perform operations.
11. A valid combination of operators and operands is known as **Expression**.
12. Arithmetic operators are used to perform arithmetic operations such as: addition, subtraction, multiplication, division etc.
13. Relational operators are also called comparison operators. These operators are used to test the relationship between operands.
14. Logical operators are also called Boolean Operators.
15. Shorthand assignment operators are useful for self-assignment statements.
16. Increment (++) and Decrement (--) Operators are the unary operators.
17. Conditional Operator is a ternary operator.



18. The sequence of evaluation of operators in which they are applied on the operands in an expression is called the **Precedence of Operators**.
19. The order in which operators of the same precedence are evaluated is known as **Associativity**.
20. When value of one type is converted into some other type, it is called Type Conversion.

## Exercise

### Q:1 Multiple Choice Questions

- i. \_\_\_\_\_ determine a specific type, range of values and the operations that can be performed on data.
  - a. Variable
  - b. Expression
  - c. Operand
  - d. Data Type
- ii. Floating-point data type is used to store \_\_\_\_\_.
  - a. Integers
  - b. Character
  - b. Real numbers
  - d. Boolean
- iii. A \_\_\_\_\_ is an identifier that represents a memory location.
  - a. Variable
  - b. Literal
  - c. Integer
  - c. Operator
- iv. \_\_\_\_\_ are the data items on which operators can perform operations
  - a. Operands
  - b. Literal
  - c. Data Types
  - c. Operator
- v. \_\_\_\_\_ operators are used to test the relationship between operands.
  - a. Arithmetic
  - b. Unary
  - c. Relational
  - c. Ternary
- vi. Increment (++) and Decrement (--) Operators are the \_\_\_\_\_ operators.
  - a. Unary
  - b. Binary
  - c. Ternary
  - c. None of these
- vii. When value of one type is converted into some other type, it is called \_\_\_\_\_.
  - a. Data Conversion
  - b. Type Conversion
  - c. Value Conversion
  - c. Operator Conversion

### Q:2 Fill in the Blanks

- i. Java supports four types of integer data type: byte, short, int and \_\_\_\_\_.
- ii. Character data type is used to store a single Unicode character enclosed in \_\_\_\_\_ quotes.
- iii. A valid combination of operators and operands is known as \_\_\_\_\_.

- iv. Logical operators are also called \_\_\_\_\_ Operators.
- v. Conditional Operator is a \_\_\_\_\_ operator.
- vi. The order in which operators of the same precedence are evaluated is known as \_\_\_\_\_.

### Q:3 Short Answer Type Questions

- i. Define Data types.
- ii. Write the name of different types of Primitive Data Types used in Java.
- iii. What is a Variable?
- iv. What are Operands?
- v. Write about the Arithmetic Operators of Java.
- vi. What are assignment Operators?
- vii. What do you know about Increment and Decrement Operators?
- viii. What do you mean by precedence of Operators in Java?
- ix. What is Type Conversion?

### Q:4 Long Answer Type Questions

- i. What is Expression? Explain different types of Expressions in Java.
- ii. What are Relational Operators? Explain with suitable examples.
- iii. Define Logical Operators. Explain with suitable examples.
- iv. Write a program in Java which shows the use of Arithmetic Operators in Java.
- v. How will we use Conditional Operator in Java? Explain with suitable example.
- vi. What is Type Conversion? Explain different types of Type Conversion.



# Control Statements in Java



## OBJECTIVES OF THIS CHAPTER

- 6.1 Introduction
- 6.2 Branching Statements: if else, switch case
- 6.3 Looping Statements: for, while, do while
- 6.4 Jumping Statements: break, continue
- 6.5 Type Conversion

### 6.1 INTRODUCTION

The statements inside our source code files are generally executed from top to bottom, in the order they appear. Control statements, however, break up the flow of execution by employing decision making, looping and jumping statements which enables our program to conditionally execute specific blocks of code. This chapter describes the decision-making statements (if, if-else, switch case etc.), the looping statements (for, while, do-while), and the jumping statements (break, continue, return) supported by Java programming language.

Whenever we want to change the execution flow of statements, we can use Control Statements in the program. These statements change the execution flow based on some test condition. Based on different ways of control flows, Control Statements can be divided into three categories, namely:

1. Conditional (Decision Making) Statements
2. Looping (Iterative) Statements
3. Jumping Statements

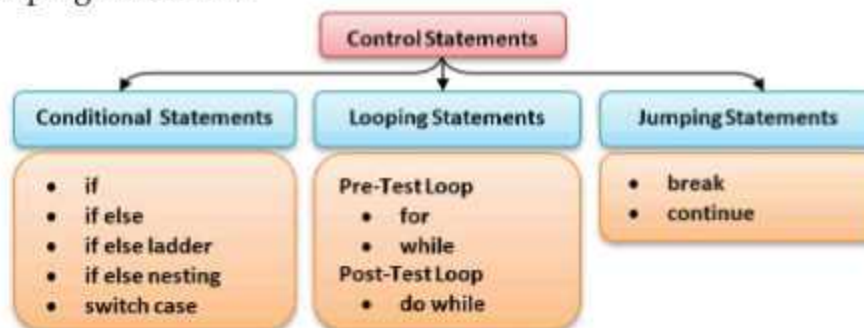


Fig: 6.1 Types of Control Statements

Now, let's begin explaining these statements in detail.



## 6.2 CONDITIONAL (DECISION MAKING) STATEMENTS

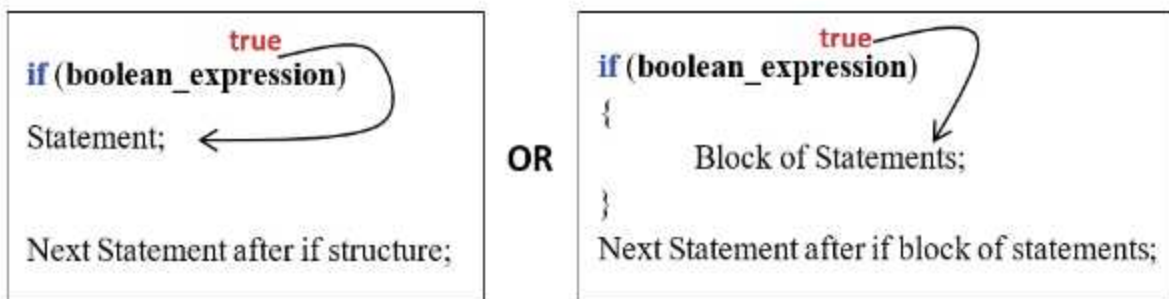
These statements can be used for decision – making purpose or for making multi-way selection. We can make decisions for executing the statements based upon the result of a condition. These statements evaluate the test-condition (Boolean Expression) to control the execution flow in the program. Result of condition determines which block of statements will be executed. That is why these statements are also known as Decision-Making statements. These statements are also called branching statements because the program chooses to follow one branch or another during execution. Various conditional flow statements in java are given following:

1. Only if statement
2. if-else statement
3. if-else ladder statement
4. if-else nesting statement
5. switch case statement

### 6.2.1 Only if Statement:

It is the simplest form of if else statement. The 'if statement' evaluates the `boolean_expression` (condition) and then proceed to carry out the set of actions only if the test condition is evaluated to true. It is terminated when the test condition evaluates to false. The syntax for using this statement is given below:

**Syntax:**



Here, the `boolean_expression` also referred so as condition must be enclosed in parenthesis, which causes the `boolean_expression` (test condition) to be evaluated first. Most often, the expression used to control the if, will involve the relational OR/ AND logical operators. If this expression is evaluated true, then the "Block of Statements" will be executed otherwise the "Block of Statements" connected with if statement will be ignored and control will be passed to the next statement after if block of statements. Consider the following program example:

**Program 6.1:** Write a program in Java to show "Excellent Score!!" message to the student only if marks scored by the student are above 80%.

```
1 class cs1
2 {
3     public static void main(String args[])
4     {
5         int math=85, sci=90;
6         int total=math+sci;
7         double per=total/200.0*100.0;
8
9         if(per>80)
10        {
11            System.out.println("Excellent Score!!");
12        }
13        System.out.println("Your Percentage Marks are: "+per);
14    }
15 }
```

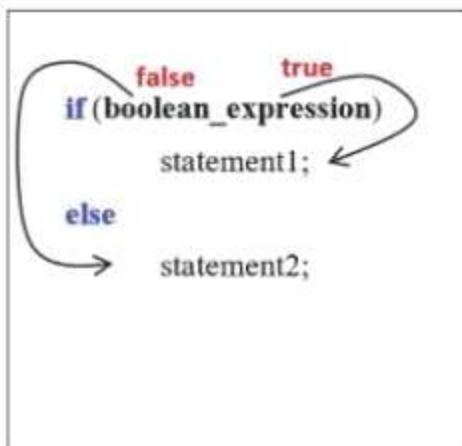
**Compilation, Execution and Output of Program 6.1:**

```
D:\Java Programs>javac cs1.java
D:\Java Programs>java cs1
Excellent Score!!
Your Percentage Marks are: 87.5
D:\Java Programs>
```

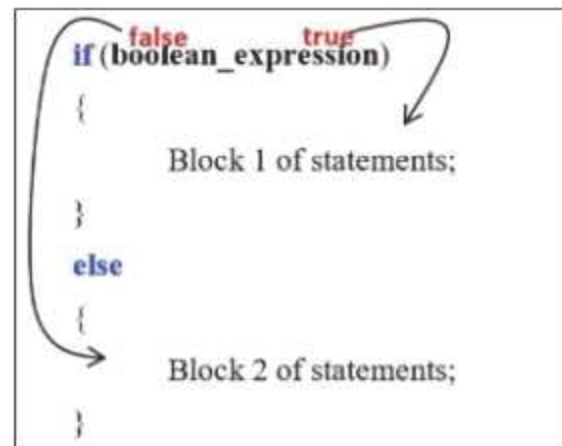
### 6.2.2 if else Statement:

In this conditional control statement, "statement" or "block of statements" associated with **if** statement gets executed only when the given **boolean\_expression** (test condition) is evaluated to **true** while the statements in **else** block gets executed only when the **boolean\_expression** (test condition) returns **false**. The syntax for using if else statement is given below:

**Syntax:**



OR



As shown in the above syntax, If the **boolean\_expression** (condition) is **true**, then "statement1" or "Block 1 of statements" is executed. Otherwise (If **boolean\_expression** is **false**), "statement2" or "Block 2 of statements" is executed. In no case, both statements will be executed.

**Program 6.2: Write a program to find whether the student is "Pass" or "Fail" according to the value of percentage marks.**

```
1 class cs2
2 {
3     public static void main(String args[])
4     {
5         int math=35, sci=50;
6         int total=math+sci;
7         double per=total/200.0*100.0;
8         if(per>35)
9         {
10            System.out.println("Congrates!! You are Pass");
11        }
12        else
13        {
14            System.out.println("Sorry!! You are Fail");
15        }
16        System.out.println("Your Percentage Marks are: "+per);
17    }
18 }
19 }
```

**Compilation, Execution and Output of Program 6.2:**

```
C:\Program Files\Java\jdk-1.8.0_101\bin>
D:\Java Programs>javac cs2.java
D:\Java Programs>java cs2
Congrates!! You are Pass
Your Percentage Marks are: 42.5
D:\Java Programs>_
```

### 6.2.3 if else ladder Statement:

It is a chain of multiple if-else statements. It is used to execute the if-else condition (**boolean\_expression**) where each else part has associated set of if-else statement. For each else part, we need to evaluate another set of if-else conditions. Here, a particular "statement" or "block of statements" get executed when the corresponding condition is true. Statements in final else block gets executed when all other conditions are false. We can use any number of else if blocks between if and else.



### Syntax:

```
if (boolean_expression1)
    statement 1;
else if (boolean_expression2)
    statement 2;
    .....
    .....
else
    statement n;
```

OR

```
if (boolean_expression1)
{
    Block 1 of statements;
}
else if (boolean_expression2)
{
    Block 2 of statements;
}
.....
.....
else
{
    Block n of statements;
}
```

**Program 6.3:** Write a program to find Grade of the student If marks  $\geq 80$  then Grade A, if marks  $\geq 60$  and marks  $< 80$  then Grade B, if marks  $\geq 40$  and marks  $< 60$  then Grade C, otherwise Grade D.

```
1 class cs3
2 {
3     public static void main(String ar[])
4     {
5         int math=35, sci=50;
6         int total=math+sci;
7         double per=total/200.0*100.0;
8         if(per>=80)
9             System.out.println("Grade A");
10        else if(per>=60)
11            System.out.println("Grade B");
12        else if(per>=40)
13            System.out.println("Grade C");
14        else
15            System.out.println("Grade D");
16
17        System.out.println("Your Percentage Marks are: "+per);
18    }
19 }
```

### Compilation, Execution and Output of Program 6.3:

```
C:\Windows\system32\cmd.exe
D:\Java Programs>javac cs3.java

D:\Java Programs>java cs3
Grade C
Your Percentage Marks are: 42.5
```

#### 6.2.4 if-else nesting statement:

if else nesting is an if or if-else statement that is the target of another if or else or if-else. The nested if-else statements include multiple if and/or if-else statements. In other words, we can say that writing the **if** or **if-else statement with-in another if or else or if-else** is called as if-else nesting statement. Inner if or if-else is executed after the evaluation of outer boolean\_expression (condition).

Syntax:

<pre>if(boolean_expression1) {     if(boolean_expression2)     {         Block 1 of Statements;     }     else     {         Block 2 of Statements;     } } else {     Block 3 of statements; }</pre>	<pre>if(boolean_expression1) {     Block 1 of Statements; } else {     if(boolean_expression2)     {         Block 2 of Statements;     }     else     {         Block 3 of Statements;     } }</pre>	<pre>if(boolean_expression1) {     if(boolean_expression2)     {         Block 1 of Statements;     }     else     {         Block 2 of Statements;     } } else {     if(boolean_expression3)     {         Block 3 of Statements;     }     else     {         Block 4 of Statements;     } }</pre>
---	---	---

#### Program 6.4: Write a program to find largest of three numbers.

```
1 class CS4 {
2     public static void main(String args[])
3     {
4         int a=15,b=56,c=34;
5         if(a>b)
6         {
7             if(a>c)
8                 System.out.println("Largest Number is:" + a);
9             else
10                System.out.println("Largest Number is:" + c);
11        }
12        else
13        {
14            if(b>c)
15                System.out.println("Largest Number is:" + b);
16            else
17                System.out.println("Largest Number is:" + c);
18        }
19    }
20 }
```

#### Compilation, Execution and Output of Program 6.4:

```
D:\Java Programs>javac cs4.java

D:\Java Programs>java cs4
Largest Number is:56

D:\Java Programs>_
```

#### 6.2.5 switch-case Statement:

It is a multi-way branching statement. The statement switch-case is similar to if-else ladder. It is a matter of preference which we use. Switch statement can be slightly more efficient and easier to read.

switch-case statement provides the most efficient method of passing control to one of the different labels of our code based on the value of an expression. The expression must be of type byte, short, int, String or char. The values specified in the case statements must be of a type compatible with the expression. Each case value must be a constant, not a variable. Duplicate case values are not allowed.

#### Syntax:

```
switch (expression)
{
    case constant1:
        statements1; break;
    case constant2:
        statements2; break;
    ...
    ...
    case constant n:
        statements n; break;
    default:
        statements;
}
```



The value of the expression is compared with each of the constant values in the case statements. If a match is found, the code sequence following that case statement is executed. If none of the constants matches the value of the expression, then the default statement is executed. However, the default statement is optional. If no case matches and no default is present, then no action is taken by switch statement.

The break statement is used inside the switch to terminate a statement sequence. When a break statement is encountered, execution branches to the first line of code that follows the entire switch statement.

**Program 6.5: Write a program using switch case to print the name of the day corresponding to the number of the day specified by the user.**

```
1 class cs5
2 {
3     public static void main(String args[])
4     {
5         int day=2;
6         switch(day)
7         {
8             case 1: System.out.println("1st Day is Monday"); break;
9             case 2: System.out.println("2nd Day is Tuesday"); break;
10            case 3: System.out.println("3rd Day is Wednesday"); break;
11            case 4: System.out.println("4th Day is Thursday"); break;
12            case 5: System.out.println("5th Day is Friday"); break;
13            case 6: System.out.println("6th Day is Saturday"); break;
14            case 7: System.out.println("7th Day is Sunday"); break;
15            default: System.out.println("Wrong value of the day");
16        }
17    }
18 }
```

### Compilation, Execution and Output of Program 6.5

```
C:\Windows\System32\cmd.exe
D:\Java Programs>javac cs5.java
D:\Java Programs>java cs5
2nd Day is Tuesday
D:\Java Programs>java cs5_
```

## 6.3 LOOPING CONTROL STATEMENTS

Looping statements are also called **Iterative** Statements. There may be situations when we need to execute a block of statements several number of times. In such situations, loops provide a way to repeat statements. A loop repeatedly executes the same set of instructions until a termination condition is met. Java loops can be categorized into two categories as given following:

### 6.3.1 Pre-Test Loops

- ❖ for loop
- ❖ while loop

### 6.3.2 Post-Test Loop

- ❖ do-while loop

Any loop usually consists of following three expressions along with the body of the loop that contains statements to be repeated:

- ❖ **Expression1** includes initialization of counter variable that controls the loop
- ❖ **Expression2** includes boolean\_expression that defines the termination condition of loop
- ❖ **Expression3** includes step value which can be represented by increasing or decreasing the value of counter variable.

#### 6.3.1 Pre-Test Loops:

Pre-Test loops are also called Entry-Controlled loops. In these loops, the control conditions are tested before execution of the body of loop. The 'for' and 'while' loops are the entry-controlled loops in Java. These loops are explained below:

**for loop:** A for loop is perhaps the most commonly used form of looping. This pre-test iterative control structure allows us to write a loop that needs to be executed for a specific number of times. The 'for loop' statements get executed as long as condition is true.

#### Syntax:

```
for(initialization; boolean_expression; step)
{
    Body of loop;
}
```

If only one statement is being repeated, there is no need for the curly braces.

#### Working of 'for' loop:

1. When the loop first starts, the initialization portion of the loop is executed. Generally, this is an expression that sets the value of the loop control variable, which acts as a counter that controls the loop. It is important to understand that the initialization expression is only executed once.
2. Next, condition is evaluated. This must be a Boolean expression. It usually tests the loop control variable against a target value. If this expression is true, then the body of the loop is executed. If it is false, the loop terminates.
3. Next, the step portion of the loop is executed. This is usually an expression that increments or decrements the loop control variable.

4. The loop then iterates, first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression with each pass. This process repeats until the controlling expression is false.

**Program 6.6: Write a program to print first n natural numbers using for loop.**

```
cs6.java
1 class cs6
2 {
3     public static void main(String args[])
4     {
5         int i,n;
6         n=7;
7         for(i=1;i<=n;i++)
8         {
9             System.out.println("i=" + i);
10        }
11    }
12 }
```

**Compilation, Execution and Output of Program 6.6:**

```
C:\Windows\System32\cmd.exe
D:\Java Programs>javac cs6.java

D:\Java Programs>java cs6
i=1
i=2
i=3
i=4
i=5
i=6
i=7

D:\Java Programs>
```

### While loop:

It is also a pre-test loop. This loop checks the boolean\_expression (condition) before the execution of its body. The body of the loop will be executed as long as the conditional expression is true. When condition becomes false, control passes to the next line of code immediately following the loop. if the conditional expression controlling a while loop is initially false, then the body of the loop will not be executed at all. So, minimum number

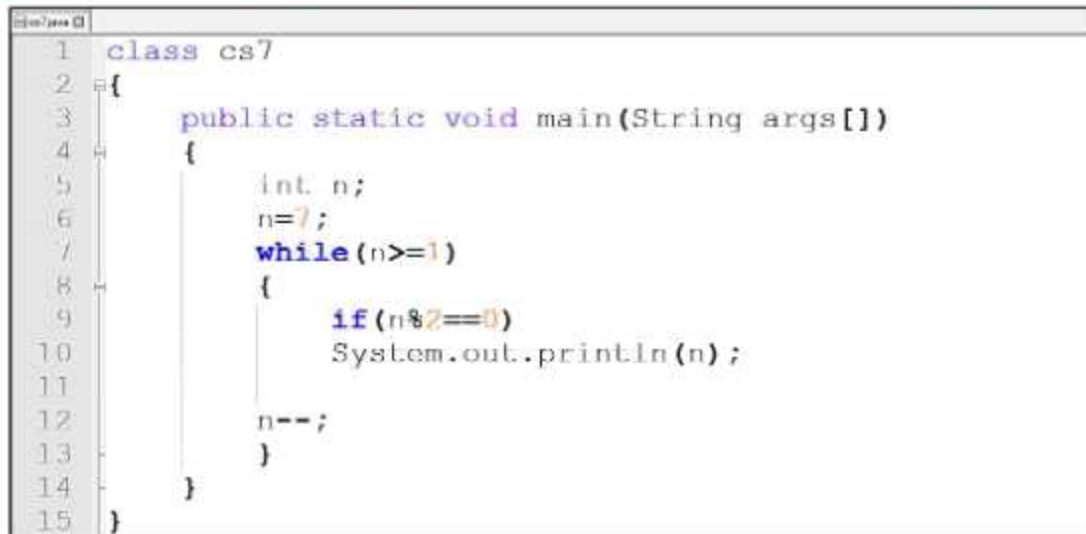


of executions of the body of while loop is zero. The general syntax of for loop is given below:

**Syntax:**

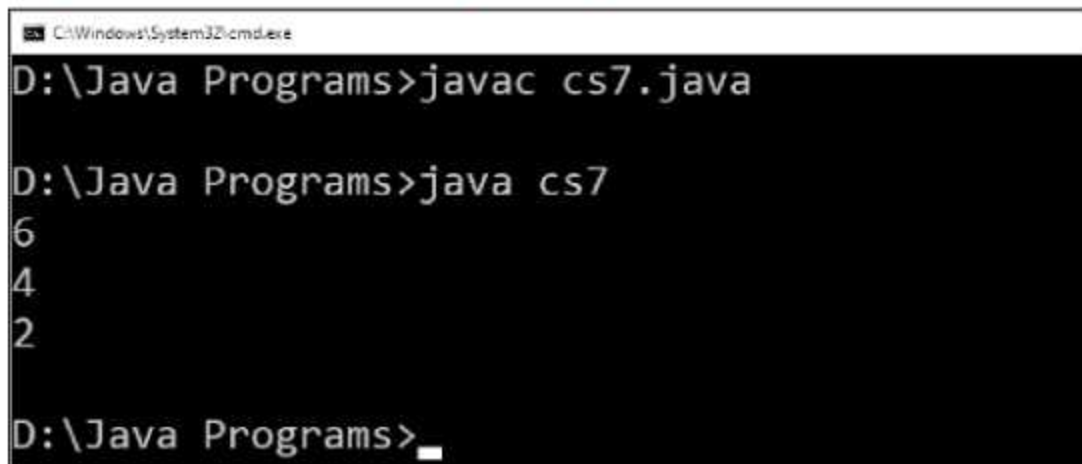
```
while(boolean_expression)
{
    Body of loop;
}
```

**Program 6.7: Write a program to print even numbers between n to 1 using while loop.**



```
1 class cs7
2 {
3     public static void main(String args[])
4     {
5         int n;
6         n=7;
7         while (n>=1)
8         {
9             if (n%2==0)
10                System.out.println(n);
11
12            n--;
13        }
14    }
15 }
```

**Compilation, Execution and Output of Program 6.7:**



```
C:\Windows\System32\cmd.exe
D:\Java Programs>javac cs7.java

D:\Java Programs>java cs7
6
4
2

D:\Java Programs>_
```

### 6.3.2 Post-Test Loop:

Sometimes it is desirable to execute the body of a loop at least once, even if the conditional expression is false. In other words, there are times when we would like to test the termination expression at the end of the loop rather than at the beginning. In such situations, Post Test loops are used. Post-Test loop is also known as Exit-Controlled loop. The 'do while' loop is the only one exit-controlled loop in Java.

### do while loop:

In do-while loop, statements get executed as long as condition is true. The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop. Therefore, in do while loop, the minimum number of executions for the body of the loop will be one. Each iteration of the do-while loop first executes the body of the loop and then evaluates the conditional expression. If this expression is true, the loop will repeat. Otherwise, the loop terminates. General syntax of do while is as follows:

**Syntax:**

```
do
{
    ....
    statements;
    ....
} while(boolean_expression);
```

The sole difference between while and do-while is that the statement of the do-while always executes at least once, even if the expression evaluates to false for the first time. In a while, if the boolean\_expression (condition) is false for the first time, the statement never executes. In practice, do-while is less common than while.

**Program 6.8: Write a program to print odd numbers from 1 to n using do while loop.**

```
1 class cs8
2 {
3     public static void main(String args[])
4     {
5         int i,n;
6         n=7;
7         i=1;
8         do{
9             if(i%2==1)
10                System.out.println(i);
11
12            i++;
13        }while(i<=n);
14    }
15 }
```

### Compilation, Execution and Output of Program 6.8:

```

D:\Java Programs>javac cs8.java

D:\Java Programs>java cs8
1
3
5
7

D:\Java Programs>
```

## 6.4 JUMPING CONTROL STATEMENTS

Jumping statements are also used for altering the normal flow of a program. Using these statements, we can transfer the execution control from one location to some other location in the program. Loops perform a set of repetitive tasks (statement) until condition becomes false but it is sometimes desirable to skip some statements inside loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used. The break statement is also used in switch statement to exit switch statement.

Following are the Jumping control statements built in Java to alter the normal flow of execution in the program:

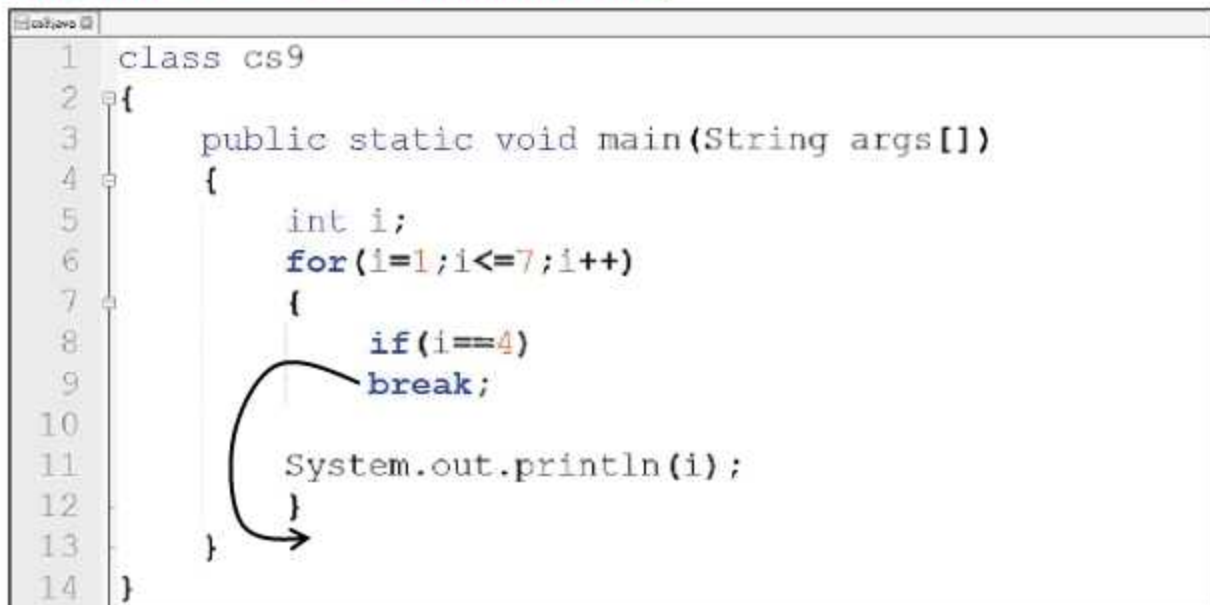
- ❖ break
- ❖ continue

### 6.4.1 break statement:

The break statement terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch statement. The break statement in Java programming language has the following two usages:

1. When the break statement is encountered inside a loop, the loop is immediately terminated and program execution resumes at the next statement following the loop.
2. It can be used to terminate a case in the switch statement.

#### Program 6.9: Use of break statement in the loop.



```
1 class cs9
2 {
3     public static void main(String args[])
4     {
5         int i;
6         for(i=1; i<=7; i++)
7         {
8             if(i==4)
9                 break;
10
11             System.out.println(i);
12         }
13     }
14 }
```

The screenshot shows a Java IDE with a file named 'cs9.java'. The code defines a class 'cs9' with a 'main' method. Inside the 'main' method, a 'for' loop is used to iterate from 1 to 7. Within the loop, there is an 'if' statement that checks if 'i' is equal to 4. If this condition is met, the 'break' statement is executed, which immediately terminates the loop. An arrow in the image points from the 'break' statement to the closing brace of the 'for' loop, indicating the exit point. After the loop, the program prints the value of 'i' using 'System.out.println(i);'.



### Compilation, Execution and Output of Program 6.9:

```
C:\Windows\System32\cmd.exe
D:\Java Programs>javac cs9.java
D:\Java Programs>java cs9
1
2
3
D:\Java Programs>
```

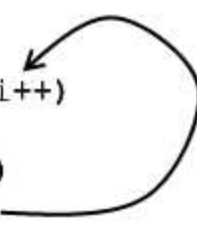
As shown in the program 6.9, Using **break**, we can force immediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop. Here, when value of *i* becomes 4, **break** statement terminates the execution of loop and it will take the execution control out of the loop.

#### 6.4.2 continue statement:

It is sometimes desirable to skip some statements inside the loop without exiting the loop. The **continue** statement stops the execution of the current iteration and goes back to the beginning of the loop to begin the next iteration.

**Program 6.10: Use of continue statement in the loop.**

```
1 class cs10
2 {
3     public static void main(String args[])
4     {
5         int i;
6         for(i=1;i<=7;i++)
7         {
8             if(i%2==0)
9                 continue;
10
11             System.out.println(i);
12         }
13     }
14 }
```



### Compilation, Execution and Output of Program 6.10:

```
C:\Windows\System32\cmd.exe
D:\Java Programs>javac cs10.java

D:\Java Programs>java cs10
1
3
5
7

D:\Java Programs>_
```

In the program 6.10, When the value of 'i' is divisible by 2 (i.e. when i=2, 4 and 6), the continue statement plays its role and skip their execution but for other values of 'i', the loop will run smoothly.

## Points to Remember

1. Whenever we want to change the execution flow of statements, we can use Control Statements in the program.
2. Conditional statements can be used for decision – making purpose or for making multi-way selection.
3. Conditional statements are also called branching statements because the program chooses to follow one branch or another during execution.
4. Writing the **if** or **if-else** statement **with-in** another **if** or **else** or **if-else** is called as if-else nesting statement.
5. switch-case is a multi-way branching statement.
6. switch-case statement provides the most efficient method of passing control to one of the different labels of our code based on the value of an expression.

7. Looping statements are also called **Iterative** Statements.
8. Pre-Test loops are also called Entry-Controlled loops. In these loops, the control conditions are tested before execution of the body of loop.
9. The 'for' and 'while' loops are the entry-controlled loops in Java.
10. for loop is a pre-test iterative control structure that allows us to write a loop that needs to be executed for a specific number of times.
11. Minimum number of executions for the body of while loop is zero.
12. Post-Test loop is also known as Exit-Controlled loop.
13. The 'do while' loop is the only exit-controlled loop in Java.
14. The do-while loop always executes its body at least once.
15. Using Jumping statements, we can transfer the execution control from one location to some other location in the program.
16. break and continue are the jumping statements in Java.
17. The break statement terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch statement.
18. The **continue** statement stops the execution of the current iteration and goes back to the beginning of the loop to begin the next iteration.

## Exercise

### Q:I Multiple Choice Questions

- i. \_\_\_\_\_ statements are used to control the execution flow of a program.
  - a. Compound
  - b. Control
  - c. Comparative
  - d. None of these
- ii. \_\_\_\_\_ statements can be used for decision-making purpose or for multi-way selection.
  - a. Looping
  - b. Iterative
  - c. Conditional
  - d. Jumping
- iii. \_\_\_\_\_ is a multi-way branching statement.
  - a. switch-case
  - b. if-else
  - c. while
  - d. do-while



- iv. Looping statements are also called \_\_\_\_\_ Statements.
- |                    |              |
|--------------------|--------------|
| a. Iterative       | b. Selection |
| c. Decision-Making | d. Jumping   |
- v. Which of the following is an example of Post-Test loop in Java?
- |             |                  |
|-------------|------------------|
| a. for loop | b. while loop    |
| c. do-while | d. None of these |
- vi. The 'for' and 'while' loops are the \_\_\_\_\_ loops in Java.
- |                     |                    |
|---------------------|--------------------|
| a. entry-controlled | b. exit-controlled |
| c. Post-Test        | d. None of these   |
- vii. break and continue are the \_\_\_\_\_ statements in Java.
- |                    |              |
|--------------------|--------------|
| a. Iterative       | b. Selection |
| c. Decision-Making | d. Jumping   |

#### Q: II Fill in the Blanks

- i. Pre-Test loops are also called \_\_\_\_\_ loops.
- ii. Writing the if or if-else statement with-in another if-else statement is called as \_\_\_\_\_ statement.
- iii. Minimum number of executions for the body of while loop is \_\_\_\_\_.
- iv. The \_\_\_\_\_ statement terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch statement.
- v. The \_\_\_\_\_ loop always executes its body at least once.
- vi. \_\_\_\_\_ statement is an example of Jumping Statement.

#### Q: III Short Answer Type Questions

- i. Define Control Statements.
- ii. Write the name of different types of control statements used in Java.
- iii. What are conditional controls statements in Java?
- iv. Write the syntax of if-else statement in Java.
- v. What do you know about switch-case statement of java.
- vi. What are Iterative Statements?
- vii. What are Pre-Test loops? Give Examples.
- viii. Compare while and do-while loops.
- ix. What are Jumping Statements?

#### Q: IV Long Answer Type Questions

- i. What are Control Statements? Give an overview of different types of controls statements used in java.
- ii. What are Decision-Making Statements? Explain
- iii. Write a program in Java to find the largest of three values.
- iv. What are looping statements? Explain different categories of looping statements.
- v. Write a program in java to display all even values between 1 to n.
- vi. What are Jumping Statements? Explain the use of break statement in loops.



# Classes and Objects



## OBJECTIVES OF THIS CHAPTER

- 7.1 Class
- 7.2 Objects
- 7.3 Static Members
- 7.4 Constructors
- 7.5 Overloading in Java (Methods and Constructors)

### INTRODUCTION:

Languages like BASIC, C are considered to be Procedural Languages where each program is nothing more than a crisscross of functions. There is a lack of data binding among function and data members. Such a programming type fails to design a robust application. To deal with this drawback, java offers Object Oriented Programming paradigm which is also known as OOP. Here, “**Object**” means a real-world entity such as a pen, chair, table, computer, watch, etc. An Object-Oriented Programming is a methodology to design a program using classes and objects. This programming technique simplifies software development and maintenance by providing some concepts like Inheritance, Polymorphism, Abstraction etc. Let's learn more about OOPs by understanding class and object.

### 7.1 CLASS:

A **class** in java is a user defined prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, a class can be defined as a template/blueprint that describes the behavior and/or state that the object of its type supports. Thus, we can simply say, “class in Java determines how an object will behave and what the object will contain”. Usually a class contains fields and methods related to an object to be used in a program. We can explain the concept of Class and Object in the form of a diagram as follows:



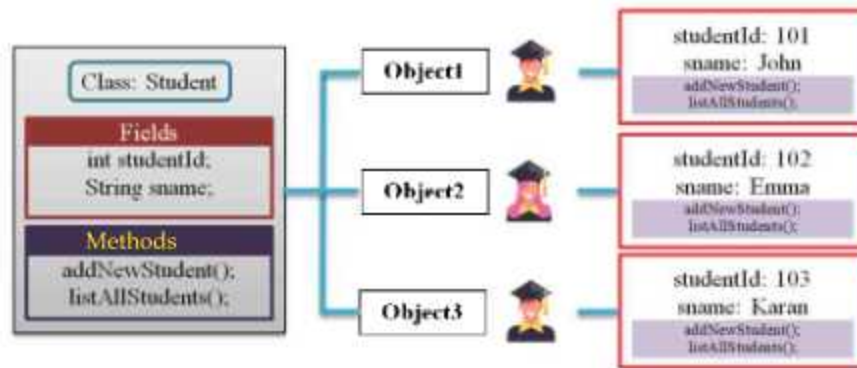


Fig 7.1 Concept of Class and Objects

In Java, a class declaration can include some of the components given below.

### 7.1.1 Basic Components of Java Class:

- **Modifiers** : Modifiers are the keywords which describes the access rights of members of a class. In java, a class can be public or has default access i.e. friendly.
- **Class Keyword** : class keyword is used to create a class.
- **Class Name** : This is an identifier of class. This name of class should begin with letter. As per java naming conventions, first letter of each class name should be capital.
- **Body** : The class body is surrounded by braces, i.e. { }. Body part of java class can contain two type of elements.
  - o Fields
  - o Methods

Each of these components are having their own importance. A general syntax of creating a class is as under:

```
[Modifier] class ClassName
{
  Body of Java Class
  // Fields
  // Methods
}
```

**Note :** Modifier shown in square brackets specifies that it is optional part of class.

Example Program 7.1:

```
public class CompApp
{
    int ch1,ch2;           //Instance Variables
    static String title;   // Class Variables

    void showTitle()
    {
        int a=10;         // Local Variables
        System.out.println("Title of Chap. is "+title);
        -----
    }
}
```

### 7.1.2 Fields (Member Variables) in Class:

Variables declared inside the class are called Fields. We can declare fields with different access specifiers such as private, public, protected etc. We can categorize the member variable in following types:

- **Instance Variables** : These are the variables that are inherent to an object and that can be accessed from inside any method, constructor or block. They are destroyed when the object is destroyed.
- **Class Variables** : Class variables or static variables are declared with the static keyword in a class. They are similar to instance variables, but they are created when the program starts and destroyed when the program stops. The main difference with instance variables is in what scope they are available. A class variable can be accessed with class name, while an instance variable is accessible only by class object.

### 7.1.3 Methods in Class:

A method in Java is a group of instructions that performs a specific task. It provides the reusability of code. We can divide a complex problem into smaller parts known as modules which makes our program easy to understand and reusable. In Java, there are two basic types of methods:

- **Standard Library Methods**: These type of methods are also known as Pre-defined Methods. These built-in methods in Java are already defined in Java library. Examples of this type of method can be print() method that comes that under java.io. PrintStream which prints the string that is written within the quotation. sqrt() is another method of Math class which returns the square root of specific number given as an argument.
- **User-Defined Methods**: We can create our own method based on our requirements in java. All methods designed by user itself are called "User

Defined Methods". We can create any number of methods in a java class to define behavior of our Object.

User can design the method according to the requirement by using data fields or local variables and statements to preform requisite task. We can define local variable of a method as under:

- **Local variables:** These are the temporary variables defined inside methods. They are declared and initialized within that method, and will be made eligible for garbage collection once the execution of method is completed. These variables are not accessible outside the methods they are declared in.

#### 7.1.4 Access Modifiers

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of these elements by applying the access modifier on them. There are four access modifiers in java.

- **Private:** Accessible within same class only.
- **Default:** Accessible within same class and same package only.
- **Protected:** Accessible within same class, same package and outside the package using subclass only.
- **Public:** Accessible anywhere outside the class and package.

We can easily understand the difference between all these access modifiers with the help of a table given following:

Access Modifier	Within Class	Within Package	Outside Package by Subclass only	Outside Package
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

Table 7.1 Access Modifiers in Java

**Note :** Java package is a group of similar type of classes, interfaces and subpackages. Think of it as a folder in a file system.

Following program shows the basic program of Java class with its component

```
class FindArea
{
    float radius, area;           //Fields
    void getRadius(float r)       //Method
    {
```



```

        radius=r;
    }
    void showArea ()                //Method
    {
        final float PIE=(float)3.143;
        area=PIE*radius*radius;
        System.out.println("The Area of Circle is: "+area);
    }
}

```

## 7.2 OBJECTS

A Java object is a member (also called an instance) of a Java class. Each object has an identity, a behavior and a state. The identity is an internal ID assigned to each object, state of an object is stored in fields (variables), while methods (functions) display the object's behavior. Objects are created at runtime from templates, also known as classes. It is a basic unit of Object-Oriented Programming and represents real life entities. A typical Java program creates many objects, which as we know, interact by invoking methods.

If we consider the real-world, we can find many objects around us like cars, dogs, humans, etc. All these objects have a state and a behavior. In the case of a dog as an example of Object, its states can be - name, breed, color, and the behavior is - barking, wagging the tail, running. Similarly, in software development, methods operate on the internal state of an object and the object-to-object communication is also done via methods. An object has two characteristics:

- **State:** represents the data (value) stored in every field of an object.
- **Behavior:** represents the functionality of an object given in the form of method, such as read Value, display Value etc.

We shall have a look on all these terms in the form of a program later, and try to understand the importance of each one.

### 7.2.1 Use of new Keyword in Java

The Java new keyword is used to create an instance of the class. In other words, it instantiates a class by allocating memory for a new object and returning a reference to that memory. We can also use the new keyword to create the array object. Some of the main characteristics of new keyword are as under:

- It is used to create the object.
- It allocates the memory at runtime.
- All objects occupy memory in the heap area.
- It invokes the object constructor.

### 7.2.2 Creating Object in Java:

The object is a basic building block of JAVA Program. We cannot use the attributes or methods of any class without creating an object. There are various ways to create an object in Java. Usually, we use **new** keyword to create an object in java. It allocates memory (heap) for the newly created object and also returns the reference of that object to allocated memory. The syntax for creating an object is:

Syntax of creating object using new Keyword:

```
ClassName Object_Name=new ClassName ([argument list-if any]);  
Or  
ClassName Object_Name;  
Object_Name =new ClassName ([argument list-if any]);
```

#### For Example:

```
FindArea obj=new FindArea();  
Or  
FindArea obj;  
obj=new FindArea();
```

Now we have understood the concept of Class and Objects. Lets make a simple Java programs to demonstrate Class and Objects in JAVA.

#### Program 7.2 (CAProg1.java) Program for creation and use of a class in Java

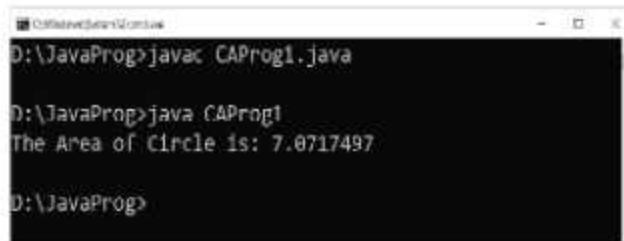
```
class FindArea  
{  
    float radius, area;                //Fields  
    void getRadius(float r)            //Method  
    {  
        radius=r;  
    }  
    void showArea ()                   //Method  
    {  
        final float PIE=(float)3.143; //using type casting  
        area=PIE*radius*radius;  
        System.out.println("The Area of Circle is: "+area);  
    }  
} //end of Find Area Class
```

```

class CAProg1
{
    public static void main(String arg[])    {
        FindArea obj=new FindArea();        //creating object
        obj.get Radius(1.5f);
        obj.showArea();
    }
}

```

### Compilation, Execution and Output of Program 7.2 (CAProg1.java)



```

D:\JavaProg>javac CAProg1.java

D:\JavaProg>java CAProg1
The Area of Circle is: 7.0717497

D:\JavaProg>

```

### 7.3 STATIC MEMBERS (CLASS VARIABLES)

The static modifier means that the entity to which it is applied is available outside any particular instance of the class. It also means that the static methods or attributes are a part of the class and not an object. The memory is allocated to such an attribute or method at the time of class loading. The use of a static modifier makes the program more efficient by saving memory. A static field exists across all the class instances, and can be called without creating an object of the class. The use of a static modifier can be understood in the form of a program given below:

#### Program 7.3 (CAProg2.java) Program for static variable/class variable

```

class StaticDataMember
{
    int Variable=10;
    static int Static_Member=10;
    void showValue()
    {
        System.out.println("Variable: "+Variable);
        System.out.println("Static_Member: "+Static_Member);
    }
}

```



```

        void incValue()
        {
            Variable++;
            Static_Member++;
        }
    }
}

class CAProg2 {
    public static void main(String arg[]) {
        StaticDataMember obj1=new StaticDataMember();
        obj1.showValue();
        obj1.incValue();
        obj1.showValue();
        obj1.incValue();
        StaticDataMember obj2=new StaticDataMember();
        obj2.showValue();
        obj2.incValue();
        obj1.showValue();
    }
}

```

### Compilation, Execution and Output of Program 7.3 (CAProg2.java)

```

D:\JavaProg>javac CAProg2.java
D:\JavaProg>java CAProg2
Non_Static: 10
Static_Member: 10
Non_Static: 11
Static_Member: 11
Non_Static: 10
Static_Member: 12
Non_Static: 12
Static_Member: 13
D:\JavaProg>

```

In the given example, we can clearly see the importance of static and instance data fields in a class. We have declared two different variables named Variable and Static\_Member in the given class Static Data Member. We created two methods, showValue(); to display the values of both these fields whereas incValue(); to increase the value of both these variables with one. We created two different objects named obj1 and obj2. We called the given methods with both these objects in certain order. We can notice that the static variable of the class is free from the object with which it is called. The value of the static variable

is being shared among all the objects. On the other hand, Variable (instance variable) is associated with object. It gives different values when called with different objects. This is the main function of static variable which makes the field shared among all the objects of the same class.

## 7.4 CONSTRUCTORS IN JAVA

A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes. In Java, a constructor is a block of codes similar to the method but having some special properties. Constructor is called automatically when an instance of the class is created. At the time of calling the constructor, memory for the object is allocated. It is a special type of method which is mainly used to initialize the object. We can explain the difference between method and constructor as under:

Method	Constructor
Method can be named as per the operation performed by it.	Constructors must have the same name as the class within which it is defined.
Method can return upto one value of any given type. However any number of arguments can be passed to a method.	Constructors do not return any value of any type. However any number of arguments can be passed to a constructor similar to the method.
A Method can be called explicitly any number of time when required.	Constructors are called only once at the time of Object creation.

### 7.4.1 Rules to be followed while defining constructors

There are some rules for creating constructors in java class. We can explain some of them as follows:

- Constructor of a class must have the same name as the class name in which it is declared.
- Access modifiers can be used in constructor declaration to control its access.
- A constructor in Java cannot be abstract, final, static, or Synchronized.
- A Constructor must have no explicit return type, even not void type.

### 7.4.2 Types of Constructors in Java

Now is the correct time to discuss the types of the constructor, so primarily there are two types of constructors in java:

1. **Default Constructor (No-Argument Constructor)** : A constructor that has no parameter is known as the default constructor. If we don't define a constructor in a class, then the compiler automatically creates a default constructor(with no arguments) for the class. And if we write a constructor

with arguments or no arguments then the compiler does not automatically create a default constructor. The default constructor initializes any uninitialized instance variables with default values as per given bellow:

Type	Default Value
boolean	false
byte	0
short	0
int	0
long	0L
char	\u0000
float	0.0f
double	0.0d
object	Reference null

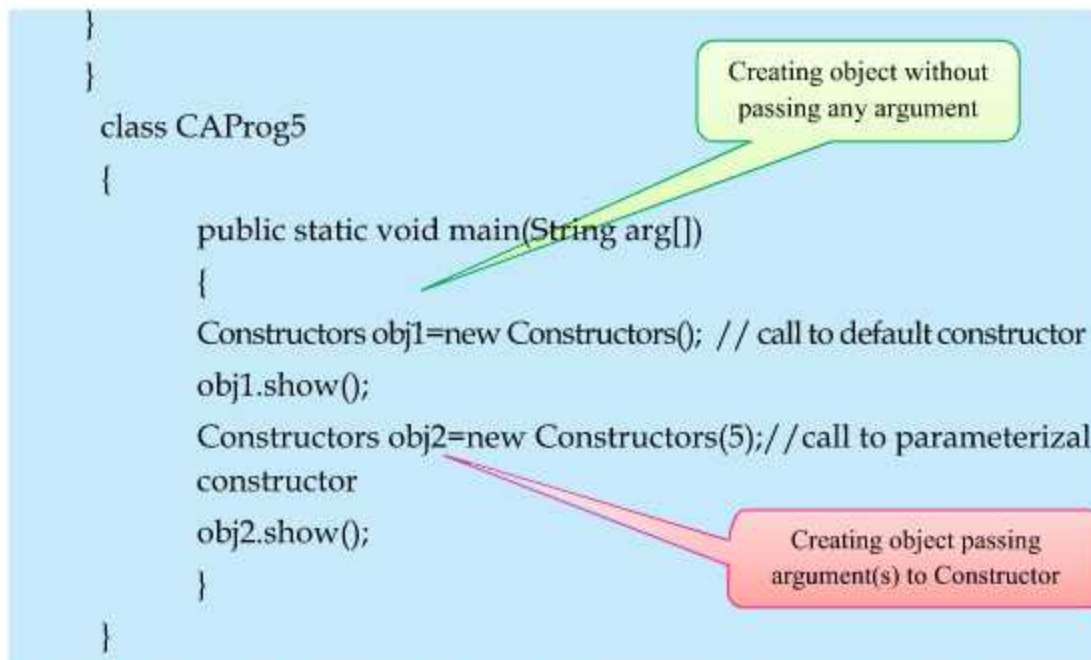
Table: Default values of instance variables

2. **Parameterized Constructor:** A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor. For example:

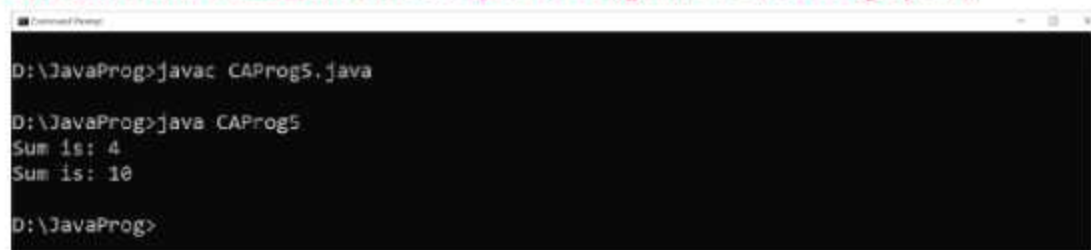
**Program 7.4 (CAProg5.java) Java Program for Constructors**

```
class Constructors
{
    int no1,no2,total;// fields
    Constructors() // default constructor
    {
        no1=no2=2;
    }
    Constructors(int inp1) // Parameterized constructor
    {
        no1=no2=inp1;
    }
    void show()//method
    {
        total=no1+no2;
        System.out.println("Sum is: "+total);
    }
}
```





### Compilation, Execution and Output of Program 7.4 (CAProg5.java)



We can pass an object of a class as an argument to a constructor also. Such a constructor is known as **Copy Constructor**. Such kind of constructors are beneficial in the case when we want to create a copy of an existing object in JAVA program. For Example:

```
Constructors(Constructor obj)                //Copy Constructor  
{  
    no1=obj.no1;  
}
```

## 7.5 OVERLOADING IN JAVA

Overloading allows different methods or constructors to have the same name, but different signatures where the signature can differ by the number of parameters or data type of parameters or both. Overloading is related to compile-time (or static) polymorphism. Overloading can never be based on the return type of a method. We can classify the overload in following two types:

1. Method Overloading
2. Constructor Overloading

### 7.5.1 Method Overloading in Java

Sometime, we may need to define multiple operations associated with one common behavior of a class based on number of arguments, then we can use method overloading. In this case, multiple methods are having same name but different in number of parameters or data type of parameters. Such a way of creating methods is known as Method Overloading. It increases the readability of the program. Suppose you have to perform addition of the given numbers but there can be any number of arguments. In the given problem, if you write the method such as sum Two (int, int) for two parameters, and sum Three (int, int, int) for three parameters and so on, then it may be difficult for us as well as other programmers to efficiently use the behavior of the method because of differences in their names. We can overcome this problem using method overload.

**Method Overloading can further be used in different ways :**

1. **Method Overloading by changing number of arguments :** In this type of method overloading, we can have two or more methods with same name but different number of arguments. For example: we can declare first set Value() method to assign any default value to the field of a class and then another method with same name but with argument to assign the given value to the class field. Lets see the implementation in a program as follows.

**Program 7.5 (CAProg6.java) Java Program for Method Overloading:**

```
class CAProg6 {
    int marks;
    void setValues()    {
        marks=0;
    }
    void setValues(int inp1)    //Method overloading
        marks=inp1;
    }
    void show()        {
        System.out.println("Marks: "+marks);
    }
    public static void main(String arg[])    {
        CAProg6 obj1=new CAProg6();
        CAProg6 obj2=new CAProg6();
    }
}
```

```

        obj1.setValues();
        obj2.setValues(450);
        System.out.println("Member of Object 1");
        obj1.show();
        System.out.println("Member of Object 2");
        obj2.show();
    }
}

```

### Compilation, Execution and Output of Program 7.5 (CAProg6.java)



```

D:\JavaProg>javac CAProg6.java
D:\JavaProg>java CAProg6
Member of Object 1
Marks: 0
Member of Object 2
Marks: 450
D:\JavaProg>

```

As we can see in the output, when we passed no argument with setValues methods then method without argument selected for execution and marks variable assigned 0 value. When we passed 450 value as an argument to the setValues method then method with argument selected for execution and passed value assigned to the marks variable. This is a main conception of method overload by changing number of arguments.

2. **Method Overloading by changing data type of arguments** : In this type of method overload, we can have multiple methods with same name but different data type of argument. For example: we can use setValues method to assign different values to different fields of a class. Such as:

### Program 7.6 (CAProg7.java) Java Program for Method Overloading :

```

class CAProg7
{
    int marks;
    String name;
    void setValues(int inp1)    {
        marks=inp1;
        name="";
    }
    void setValues(String inp2) { //Method overloading
        marks=0;
        name=inp2;
    }
}

```



```

    }
    void show() {
        System.out.println("Marks: "+marks);
        System.out.println("Name: "+name);
    }
    public static void main(String arg[])
    {
        CAProg7 obj1=new CAProg7();
        CAProg7 obj2=new CAProg7();
        obj1.setValues("Shivpreet");
        obj2.setValues(450);
        obj1.show();
        obj2.show();
    }
} //end of class

```

### Compilation, Execution and Output of Program 7.6 (CAProg7.java)



```

D:\JavaProg>javac CAProg7.java
D:\JavaProg>java CAProg7
Number of Object 1
Marks: 0
Name: Shivpreet
Number of Object 2
Marks: 450
Name:
D:\JavaProg>

```

- 3. Method Overloading by changing both, number of arguments and data type of arguments :** As its name implies, we can overload the methods by declaring multiple methods with same name but different number of arguments as well as different data types. We can implement this type of Method Overloading according to the application requirement. We can create methods as in previous examples by making changes like:

```

    void setValues(String inp1)
    {
        .....
    }
    void setValues(int inp1,String inp2)
    {
        .....
    }

```

- 4. Based on the sequence of data types in parameters:** The method overloading also depends on the ordering of data types of parameters within the method. We can use this type of Method Overloading by creating methods in previous examples with different arguments like:

```

void setValues(String inp1, int inp2)
{
    .....
}
void setValues(int inp1, String inp2)
{
    .....
}

```

### 7.5.2 Constructor Overloading in Java

We can also overload constructors as we do in method overloading. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task. Constructors are used to initialize the class fields. Sometimes, we need to pass different variables in different combination of arguments. In that case, constructor overloading becomes more convenient to do the needful. As like methods, we can also use constructor overloading in following types:

1. **Constructor Overloading by changing number of arguments :** In this type of Constructor overloading, we can have two or more Constructors with different number of arguments. For example: we can assign same value to all the fields of a class in one constructor and different values to different fields in other constructor.

#### Program 7.7 (CAProg8.java) Program for Constructor Overloading

```

class CAProg8
{
    int studentId;
    CAProg8() { //default consturtor
        studentId=101;
    }
    CAProg8(int sid) { // overloading constructor
        studentId=sid;
    }
    void show() //method
    {
        System.out.println("Your student ID is: "+studentId);
    }
    public static void main(String arg[])
    {
        CAProg8 obj1=new CAProg8();
        CAProg8 obj2=new CAProg8(2462);
    }
}

```

```

        System.out.println("Member of Object 1");
        obj1.show();
        System.out.println("-----\n Member of Object 2");
        obj2.show();
    }
}

```

### Compilation, Execution and Output of Program 7.7 (CAProg8.java)

```

D:\JavaProg>javac CAProg8.java

D:\JavaProg>java CAProg8
Member of Object 1
Your student ID is: 101
-----
Member of Object 2
Your student ID is: 2462
D:\JavaProg>

```

2. **Constructor Overloading by changing data type of arguments :** In this type of Constructor overloading, we can have multiple Constructors with different data type of arguments. For example: we can declare two Constructors to assign integer value in one constructor and string value in other one. As we have studied in Method Overloading, we can use this type of constructor overloading using following constructors:

```

CAProg8(int inp1)
{
    .....
}

CAProg8 (String inp2) //overloading constructor with different type
of arguments
{
    .....
}

```

3. **Constructor Overloading by changing both, number of arguments and data type of arguments :** As we can understand by its name, constructors can overload by giving different number of arguments as well as different data type: For example: we can declare multiple constructor with only one integer argument, both integer and string argument and/or only string argument. As we have studied in Method Overloading, we can use this type of constructor overloading using following constructors:



```

CAProg8(int inp1)
{
.....
}
CAProg8 (int inp1,String inp2)
{
.....
}

```

4. **Based on the sequence of data types in parameters :** The constructor overloading also depends on the ordering of data types of parameters within the constructor. As we have studied in Method Overloading, we can use this type of constructor overloading using following constructors:

```

CAProg8(String inp1, int inp2)
{
.....
}
CAProg8 (int inp1,String inp2)
{
.....
}

```

### 7.5.3 Advantages of Overloading in Java:

There are several advantages of Overloading in JAVA

1. Overloading in java improves code re-usability and readability.
2. Overloading in java offers flexibility to call similar methods with different types of data.
3. By using overloading in java, it is easier to remember one method name instead of multiple names. We can also create objects in different ways to define the default values of fields in different cases.
4. Consistency in naming method in java can be achieved using Overloading in methods of a class.
5. We can pass different amount of data to an object during instantiation process using constructor overloading.



## Points to Remember

1. A class can be defined as a template/blueprint that describes the behavior and/or state that the object of its type supports.
2. Modifiers are the keywords which describes the access rights of members of a class.
3. Variables declared inside the class are called Fields.
4. Instance Variable are variables that are inherent to an object and that can be accessed from inside any method, constructor or block of a class.
5. Class variables or static variables are declared with the static keyword in a class. A class variable is accessible from an object instance, while an instance variable is not accessible from a static method.
6. A method in Java is a group of instructions that performs a specific task. It provides the reusability of code.
7. A Java object is a member (also called an instance) of a Java class. Each object has an identity, a behavior and a state.
8. The Java new keyword is used to create an instance of the class.
9. A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created.
10. Constructor of a class must have the same name as the class name in which it is declared.
11. A constructor that has no parameter is known as the default constructor.
12. Method Overloading means multiple methods are having same name but different in number of parameters or data type of parameters.
13. We can also overload constructors as we do in method overloading.

## Exercise

### Que:1 Multiple Choice Questions:

- i. An instance of a Java class is known as:  
A. Method  
B. Field  
C. Object  
D. Constructor
- ii. Declaring multiple methods with same name but different number of arguments is called.  
A. Method Overloading  
B. Declaration  
C. Inheritance  
D. None of These

- iii. \_\_\_\_\_ is invoked automatically when we create an object of a class.
 

A. Field	B. Method
C. Static Member	D. Constructor
- iv. A constructor that has no parameter is known as :
 

A. Default Constructor	B. Constructor Overloading
C. Parameterized Constructor	D. Weak Constructor
- v. A Constructor can not be \_\_\_\_\_.
 

A. Abstract	B. Final
C. Static	D. All of the above

**Que:2 Fill in the blanks:**

- i. A class can be defined as a \_\_\_\_\_ of all its objects.
- ii. \_\_\_\_\_ are the variables declared inside the class.
- iii. \_\_\_\_\_ keyword is used to create an object in Java.
- iv. Method Overloading can be achieved based on \_\_\_\_\_ or \_\_\_\_\_.
- v. Declaring multiple constructor in a class is called \_\_\_\_\_.

**Que:3 Short Answer type Questions:**

- i. Define class in JAVA.
- ii. What are the basic components of java class?
- iii. Explain Fields in a class.
- iv. What do you mean by Object?
- v. Define Instance variable?
- vi. What do you mean by class variable?
- vii. What are methods in JAVA?
- viii. Explain Private and Protected modifier.
- ix. How an object of a class can be created?
- x. What is constructor?
- xi. Explain the difference between Methods and Constructors.

**Que:4 Long Answer type Questions:**

- i. What is class? Explain Fields and methods in class.
- ii. Define Constructors. Explain different types of constructors with suitable example.
- iii. What do you mean by Method overloading? Explain any two ways of method loading.
- iv. How constructor Overloading take place? Write a program to explain different types of constructor overloading.





# Strings and Wrapper Classes



## OBJECTIVES OF THIS CHAPTER

- 8.1 Strings
- 8.2 Performing operations on Strings
- 8.3 Wrapper Classes

### INTRODUCTION

Strings are an essential part of every field as we can store our data in the form of strings to process it digitally. So we can say, String is a widely used data type which allow us to store anything and everything that has to do with words, sentences, phone numbers, or even just regular numbers. In this chapter we will learn several concepts of using Strings and several methods also to handle strings effectively. Let's understand the use of Strings in detail.

### 8.1 STRINGS

Strings are used to represent a sequence of characters. In Java programming language, a String is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the `length()` method. We can also use strings using array of `char` data type. Since arrays are mutable, which means that the value of an array can be changed during execution. On the other hand, Strings are immutable which means that their values can never be changed in JAVA. When we change the value of a string (e.g.: adding an extra character, making it lowercase, swapping two characters, etc.), we are actually creating a new and separate copy of string in java heap:

**There are two ways to create a String object:**

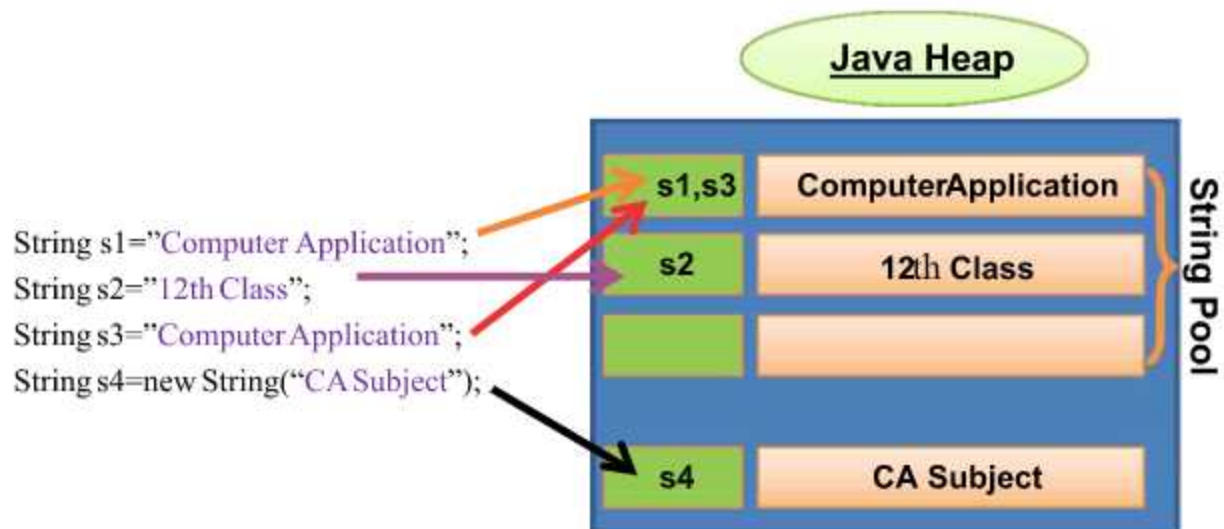
- **By string literal** : Java String literal is created by using double quotes.

**For Example:** `String s1="Computer Application";`

- **By new keyword** : Java String is created by using a keyword "new".

**For example:** `String s2=new String("CA Subject");`

Let's understand the difference between both these type of strings in JAVA in the form of a Diagram:



### 8.1.1 Advantages of Strings :

- ❖ String provides us a string library to create string objects which will allow strings to be dynamically allocated in memory. String boundary issues are also handled inside class library automatically.
- ❖ String provides us large library of inbuilt String methods which make the implementation of string very easy.
- ❖ String helps as a base for many data structures such as tree and arrays.
- ❖ Strings provide us very helpful string algorithms for solving very complex problems with less time complexity.
- ❖ Strings have close relationship with StringBuffer which provides an efficient mechanism for growing, inserting, adding, change, and other types of string manipulation.

### 8.1.2 Limitations of Strings :

- ❖ Strings are immutable in JAVA i.e. they cannot be modified or changed.
- ❖ Strings are generally slow in performing operations like input and output of String value.
- ❖ We cannot inherit string class which means overriding methods in string class is not possible.

## 8.2 PERFORMING OPERATIONS ON STRINGS

JAVA provides various methods to perform different operations on strings. We can create new String using different ways and implement a given string with the help of several string methods. Let's learn the basic use of String in the form of a program as given bellow.

**Following program shows the different ways to create and use String variable in JAVA**

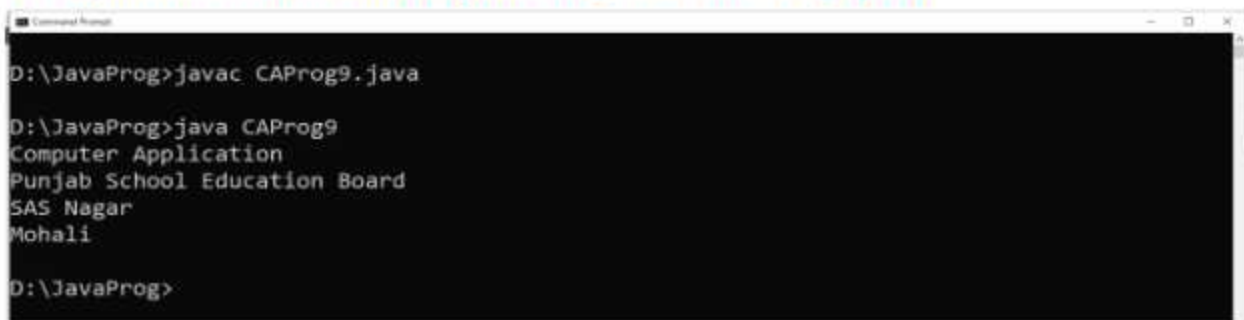
```
class CAProg9
{
    public static void main(String arg[])
    {
```

```

//First way of using String
String str1 = "Computer Application";
System.out.println(str1);
//Second way of using String
String str2=new String("Punjab School Education Board");
System.out.println(str2);
//Third way of using String
String str3;
str3=new String("SAS Nagar");
System.out.println(str3);
//Fourth way of using String
char[] strarray = { 'M','o','h','a','l','i' };
String str4 = new String(strarray);
System.out.println(str4);
}
}

```

#### Compilation, Execution and Output of Program 8.1 (CAProg9.java)



```

D:\JavaProg>javac CAProg9.java
D:\JavaProg>java CAProg9
Computer Application
Punjab School Education Board
SAS Nagar
Mohali
D:\JavaProg>

```

This java program represents the different ways of string declaration in java. We can perform some other operations on strings too using String Handling methods. Let's understand the importance and examples of string handling methods.

#### 8.2.1 String Handling Methods:

Java provides several methods to perform operations on Strings which are called String Methods. These methods perform string comparison, string search, string copy, extracting substring, and convert string to lowercase or uppercase. Methods such as compare(), concat(), equals(), split(), length(), replace(), compareTo() etc. are some examples of such type of methods. Some of the most widely used String methods are as follows:

1. **charAt():** The charAt() method returns the character at the specified index in a string. The index of the first character always begin with zero. The second character is 1 and so on. This method returns value of char data type. Following is the Syntax and example of this method:



### Syntax

```
Char_Variable= String_Identifier.charAt(Int_index);
```

Example :

```
String Str="Computer";  
char result= Str.charAt(1);  
System.out.println(result);
```

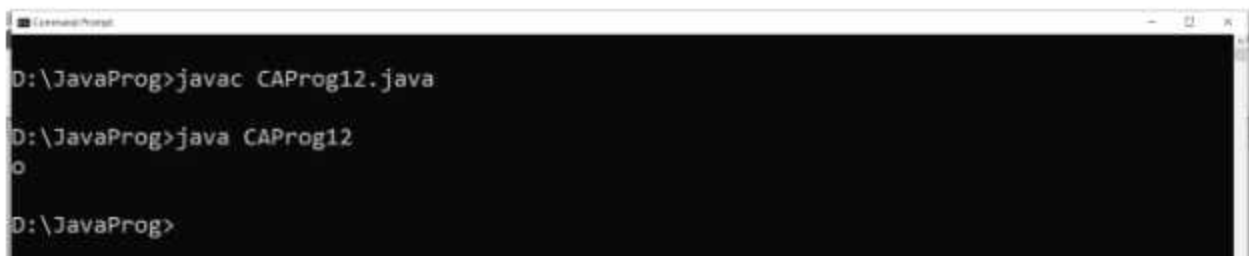
Output :

o

Following program shows the implementation of charAt() function in Java

```
class CAProg12  
{  
    public static void main(String arg[])  
    {  
        String Str="Computer";  
        char result= Str.charAt(1);  
        System.out.println(result);  
    }  
}
```

Compilation, Execution and Output of Program 8.2 (CAProg12.java)



```
D:\JavaProg>javac CAProg12.java  
D:\JavaProg>java CAProg12  
o  
D:\JavaProg>
```

Similarly, we can use all String methods in our java programs.

2. **indexOf():** This method does the opposite of charAt() function. It finds and returns the index of given character located within a string. The index of characters in a Java string always begin with zero. This method returns value of int data type. Following is the Syntax and example of this method :

**Syntax :**

```
Char_Variable= String_Identifier.indexOf(Char_Literal);
```

Example :

```
String Str = "Computer Application";  
int location = Str.indexOf('o');  
System.out.println(location);
```

Output :

1

3. **length()**: This method finds and returns the length of a string. The length of the Java string is same as the number of characters or symbols including blank spaces within a String. This method returns value of int data type. Following is the Syntax and example of this method :

**Syntax :**

```
Int_Variable= String_Identifier.length();
```

Example :

```
String Str = "Computer Application";  
int count = Str.length();  
System.out.println(count);
```

Output :

20

4. **toLowerCase()**: The Java method toLowerCase() converts and returns the string in lowercase letters. We can say, it converts all characters of the string into lower case letters. This method returns value of String type. Following is the Syntax and example of this method :

**Syntax :**

```
String_Variable= String_Identifier.toLowerCase();
```

Example :

```
String Str = "Computer Application";  
String str_result = Str.toLowerCase ();  
System.out.println(str_result);
```

Output :

computer application

5. **toUpperCase()**: The Java method toUpperCase() converts and returns the string in uppercase letter. We can say, it converts all characters of the string into upper case letters. This method returns value of String type. Following is the Syntax and example of this method :

**Syntax :**

```
String_Variable= String_Identifier.toUpperCase();
```

Example :

```
String Str = "Computer Application";  
String str_result = Str.toUpperCase ();  
System.out.println(str_result);
```

Output :

COMPUTERAPPLICATION

6. **trim():** The trim() method in Java string checks the space/spaces before and after the string, if it exists then the method removes the spaces and returns the omitted string. The string trim() method doesn't omit middle spaces. This method returns value of String type. Following is the Syntax and example of this method :

**Syntax :**

```
String_Variable= String_Identifier.trim();
```

Example :

```
String Str = " Computer Application ";  
String str_result = Str.trim();  
System.out.println("[ "+str_result+" ]");
```

Output :

[COMPUTERAPPLICATION]

7. **substring():** The Java substring() method returns a part of the string specified by given arguments of integer type. We can use this method in two different ways. If we pass only one value as a begin\_Index, the method returns all the characters from begin\_Index. But, if we pass two arguments as begin\_Index and end\_Index as an argument, it returns string between begin\_Index and end\_Index. The begin\_Index is always included in the returned string but end\_Index is excluded. This method returns a value of String type. Following is the Syntax and example of this method :

**Syntax :**

```
String_Variable= String_Identifier.substring(Int_Begin_Index);  
String_Variable= String_Identifier.substring(begin_Index,end_Index);
```

Example :

```
String Str = "Personality";  
String str_result1 = Str.substring(3);  
System.out.println(str_result1);  
String str_result2=Str.substring(3,8);  
System.out.println(str_result2);
```



Output :

```
sonality  
sonal
```

8. **concat():** The concat() method combines given string at the end of a string and returns a combined string. We can say, it appends an existing string with a given string. This method returns value of String type. Following is the Syntax and example of this method.

**Syntax :**

```
String_Variable= String_Identifier.concat(String_Value);
```

Example :

```
String Str = "Computer";  
String str_result = Str.concat("Application");  
System.out.println(str_result);
```

Output :

```
ComputerApplication
```

9. **replace():** The replace() method in Java returns a resultant string after replacing all occurrence of old\_string with new\_string. This String method is basically used to replace a sequence of char values in a given string. This method returns value of String type. Following is the Syntax and example of this method :

**Syntax :**

```
String_Variable= String_Identifier.replace(old_string,new string);
```

Example :

```
String Str = "Computer was a good and was a modern device";  
String str_result = Str.replace("was", "is");  
System.out.println(str_result);
```

Output :

```
Computer is a good and is a modern device
```

10. **String.valueOf():** The String.valueOf() method in Java converts different types of values into string value. We can convert data types like int, long, Boolean, character, float, double, object to string. This method returns value of String type. Following is the Syntax and example of this method :

**Syntax :**

```
String_Variable= String.valueOf(Any_DataType_Variable);
```

Note : ValueOf() is a static method of string class which can be called directly using class name String.

Example :

```
String str_result = String.valueOf(101);  
System.out.println(str_result.length());
```

Output :

3

**11. equals():** The equals() method compares two strings, and returns true if the strings are exactly equal, and false if not. This String comparison method is case sensitive. When letters are same but case are different then this methods returns false too. In most of the cases, this function is used in conditional control statement like if-else. This method returns value of Boolean type. i.e. true or false. Following is the Syntax and example of this method :

**Syntax :**

```
String_Variable.equals(Any_String_Variable or Constant);
```

Example :

```
String str1 = "Computer";  
String str2 = "COMPUTER";  
if(str1.equals(str2))  
System.out.println("Strings are same");  
else  
System.out.println("Strings are different");
```

Output :

Strings are different

**12. equalsIgnoreCase():** The equalsIgnoreCase() method compares two strings, and returns true if the strings are equal, and false if not. This String comparison method is not case sensitive. When letters are same but case are different then this methods returns true too. In most of the cases, this function is used in conditional control statement like if-else. This method returns value of Boolean type. i.e. true or false. Following is the Syntax and example of this method :

**Syntax :**

```
String_Variable.equalsIgnoreCase(Any_String_Variable or Constant);
```

Example :

```
String str1 = "Computer";
String str2 = "COMPUTER";
if(str1.equalsIgnoreCase(str2))
    System.out.println("Strings are same");
else
    System.out.println("Strings are different");
```

Output :

Strings are same

### 8.2.2 StringBuffer class in Java

Java StringBuffer class is used to create a mutable String object. As we have studied, String is immutable. Once a string is created, it can not be changed. On the other hand, StringBuffer class is modifiable to its contents point of view. Following is the Syntax and example of using StringBuffer Class in java :

#### Syntax:

```
StringBuffer str = new StringBuffer();
```

This type of StringBuffer class object reserves space for 16 characters without reallocation

```
StringBuffer str = new StringBuffer(20);
```

This type of StringBuffer class object accepts an integer argument that explicitly sets the size of the buffer used for storage of String.

```
StringBuffer str = new StringBuffer("ComputerApplication");
```

This type of StringBuffer class object accepts a string argument that sets the initial contents of the String Buffer object.

#### Program 8.3 (CAProg11.java) Java Program for using StringBuffer:

```
class CAProg11
{
    public static void main(String arg[])
    {
        StringBuffer str1=new StringBuffer();
        str1.insert(0,"Computer");
        System.out.println(str1);
        StringBuffer str2=new StringBuffer(25);
        str2.insert(0,"Application");
        System.out.println(str2);
        StringBuffer str3=new StringBuffer("PSEB");
        System.out.println(str3);
    }
}
```



## Compilation, Execution and Output of Program 8.3 (CAProg11.java)

```
Command Prompt
D:\JavaProg>javac CAProg11.java
D:\JavaProg>java CAProg11
Computer
Application
PSEB
D:\JavaProg>
```

### 8.2.3 String Buffer Class Methods:

All String methods discussed above can also be used on StringBuffer class object. We can use some other StringBuffer methods to perform certain operations on Strings stored in StringBuffer class object which are not applicable on String. Lets discuss such kind of methods in detail:

1. **reverse():** This method reverse the string within a String Buffer object. This method returns the character present at last index to lowest index, i.e. zero. It changes the value of object itself with which the reverse method is called. This method also returns value of StringBuffer type. Followig is the Syntax and example of this method :

#### Syntax:

String\_Variable= StringBuffer\_Object.reverse ();

#### Example

```
StringBuffer Str = new StringBuffer("Computer");
StringBuffer result = Str.reverse();
System.out.println(result);
System.out.println(Str);
```

#### Output :

```
retupmoC
retupmoC
```

2. **append():** It is used to append the specified string with given string as an argument. The append() method can be used in several types like append(char), append(boolean), append(int), append(float), append(double) etc. It changes the value of object itself with which the append() method is called. This method also returns a value of StringBuffer type. Following is the Syntax and example of this method :

#### Syntax :

String\_Variable= StringBuffer\_Object.append (value);

#### Example :

```
StringBuffer Str = new StringBuffer("Computer");
Str.append("Application");
System.out.println(Str);
```

Output :

ComputerApplication

- 3. insert():** This method inserts one string into another StringBuffer object. It accepts two values as an argument. The first parameter gives the index at which position the string will be inserted and Second argument represents the string to be inserted into StringBuffer object. It changes the value of object itself with which the insert() method is called. This method also returns value of StringBuffer type. Following is the Syntax and example of this method :

**Syntax:**

```
String_Variable= StringBuffer_Object.insert (Index,String_Value);
```

Example :

```
StringBuffer Str = new StringBuffer("Computer PSEB");  
Str.insert(9,"Application");  
System.out.println(Str);
```

Output :

ComputerApplicationPSEB

- 4. capacity():** This method returns the current capacity of StringBuffer object. We can specify the capacity of StringBuffer object. In the case, if not argument is given, an empty constructor reserves space for 16 characters. This method returns value of integer type. Following is the Syntax and example of this method :

**Syntax:**

```
Int_Variable= StringBuffer_Object.capacity ();
```

Example :

```
StringBuffer Str = new StringBuffer("Computer Application");  
System.out.println(Str.capacity);
```

Output :

36

Here, 20 Characters are given as an argument to the StringBuffer Constructor and 16 more spaces are reserved. So the output appeared 36.

- 5. delete():** The delete() method deletes a sequence of characters from the StringBuffer object. We provide two arguments to this method in which the first argument is the starting index of the String which we want to delete and the second index is the last index of the String up to which we want to delete. It changes the value of object itself with which the delete() method is called. This method also returns value of StringBuffer type. Following is the Syntax and example of this method :

**Syntax:**

```
String_Variable= StringBuffer_Object.delete(Start_index,End_Index);
```

Example :

```
StringBuffer Str = new StringBuffer("ComputerApplication");  
Str.delete(3,15);  
System.out.println(Str);
```

Output :

Comtion

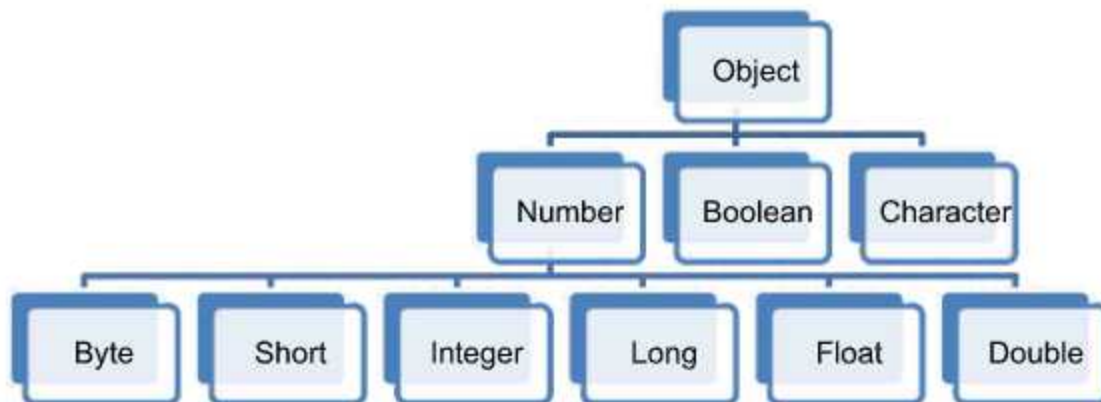
### 8.3 JAVA WRAPPER CLASSES

A Wrapper class in JAVA is a class whose object contains primitive data types. When we create an object of a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object. The automatic conversion of primitive into an object is known as autoboxing and vice-versa is called unboxing.

There are eight classes of the java.lang package which are known as wrapper classes in Java. These wrapper classes are as follows:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

We can represent these classes in a hierarchy like:





### 8.3.1 Need of Wrapper Classes

1. They convert primitive data types into objects. It is required some time when Objects are sent as an argument and needed to be modified.
2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as ArrayList and Vector, store only objects not primitive types.
4. An object is needed to support synchronization in multithreading.



## Points to Remember

1. String is a widely used data type which allow us to store anything and everything that has to do with words, sentences, phone numbers, or even just regular numbers.
2. Strings are used to represent a sequence of characters.
3. A String is actually an object, which contain methods that can perform certain operations on strings.
4. Java provides several methods to perform operations on Strings which are called String Methods.
5. String is immutable object.
6. An object of StringBuffer class is mutable by its implementation point of view.
7. A Wrapper class in JAVA is a class whose object contains primitive data types.
8. The classes in java.util package handles only objects.
9. Data structures in the Collection framework, such as ArrayList and Vector, store only objects not primitive types.

## Exercise



### Que:1 Multiple Choice Questions:

- i. A String can created by:  
A. String Literal  
B. By new Keyword  
C. Both A and B  
D. None of these
- ii. The \_\_\_\_\_ method returns the character at the specified index in a string.  
A. indexOf()  
B. charAt()  
C. length()  
D. trim()

- iii. \_\_\_\_\_ method compare two strings without case sensitivity.  
A. compare() B. equals()  
C. equalsIgnoreCase() D. All of above
- iv. Object of primitive data types are contained in \_\_\_\_\_ Classes.  
A. Default Class B. Wrapper Class  
C. Final Class D. Static Class
- v. Which one is not an example of primitive data type?  
A. char B. int  
C. short D. Double.

**Que:2 Fill in the blanks:**

- An object of \_\_\_\_\_ class is mutable by its implementation point of view.
- \_\_\_\_\_ method removes all the blank spaces from any end of the string.
- Sequence of characters can be represented as \_\_\_\_\_.
- To delete a sequence of characters from the StringBuffer object, \_\_\_\_\_ method can be used.
- \_\_\_\_\_ converts primitive data types into objects.

### Que:3 Short Answer type Questions:

- i. What do you mean by Strings?
- ii. Explain the way to create an object as a string literal.
- iii. What are the advantages of string?
- iv. Explain the limitations of string.
- v. Define toLowerCase() method with example.
- vi. What is difference between equals() and equalsIgnoreCase()?
- vii. How can we convert different types of values into string value?
- viii. What is StringBuffer class?
- ix. Explain capacity() method.
- x. What is wrapper class?

### Que:4 Long Answer type Questions:

- What do you mean by Strings? Explain different ways to create a string with example.
- Explain string handling methods. Write any two methods with program.
- Explain `replace()` and `substring()` methods with suitable examples.
- Why do we use `StringBuffer` class objects? Explain any two methods of `StringBuffer` class.



# Inheritance



## OBJECTIVES OF THIS CHAPTER

- 9.1 Inheritance and its types
- 9.2 Interfaces
- 9.3 Use of super method
- 9.4 Abstract Classes and Methods
- 9.5 Method Overriding
- 9.6 Final Class, Method and Variables

### 9.1 INTRODUCTION TO INHERITANCE

Inheritance is a feature or a process in which new classes are created from the existing classes. Here, Existing class is known as super class while new class is known as sub class. It refers to the ability of an object to take on one or more characteristics from other classes of objects. This process is same as a child inherits the traits of his/her parents. The purpose of inheritance is to consolidate and reuse an existing code. For example, if the objects "student", "teacher" and "office staff" are subclasses of Person super class. Code applying to all of them can be consolidated into a Person super class.

The characteristics inherited are usually instance variables or methods. The new class created is also called "derived class" or "child class" and the existing class is known as the "base class" or "parent class". The sub (derived) class now is said to be inherited from the super (base) class. We can understand inheritance in graphical form as shown below :

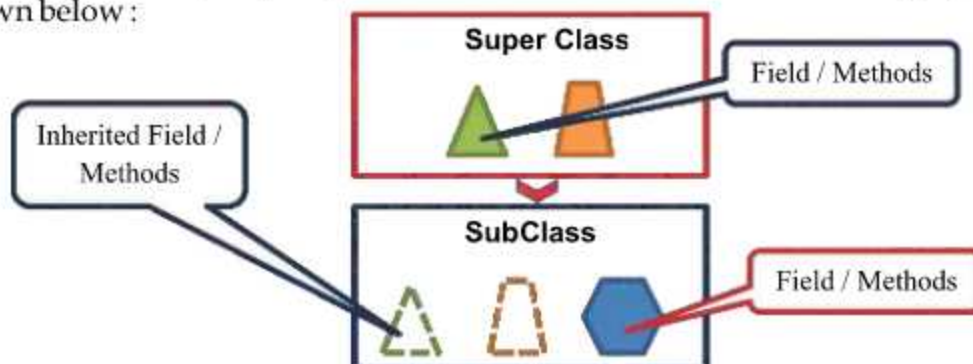


Fig. 9.1: Concept of Inheritance

#### Terminologies related to the Inheritance:

❖ **Class** : It is defined as a collection of objects sharing common properties. It is a kind of blueprint of objects created in a Java program.



❖ **Sub Class:** It is also known as derived class. This category of classes inherits the options from another class. Sub Class can include the inheritable fields and behaviors of its Super class as well as generated within its own.

❖ **Super Class:** The super class is also known as the parent class. It represents the category of those classes whose fields or methods are inheritable by a subclass.

❖ **Reusability:** As the name is defined, it is a technique of reusing members of the existing class in the new created class as it is. It can reuse the fields or methods of existing class. In short, we can say that it permits reusing the code.

These all terminologies are the basics of inheritance in Java. Following is the syntax and example of Inheritance in java :

#### Syntax

```
class derived_class extends base_class
{
//fields
//methods
}
```

Note : extends keyword is used to inherit the members of a super class in java.

**Following program shows the implementation of inheritance in JAVA**

```
class SupClass
{
void showStart()
{
System.out.println("Welcome to Computer Application Subject");
}
}
class SubClass extends SupClass
{
void showEnd()
{
System.out.println("Bye! Hope to see you again..");
}
}
class CAProg13
{
    public static void main(String arg[])
    {
        SubClass obj=new SubClass();
        obj.showStart();
        obj.showEnd();
    }
}
```

## Compilation, Execution and Output of Program 9.1 (CAProg13.java)

```
D:\JavaProg>javac CAProg13.java
D:\JavaProg>java CAProg13
Welcome to Computer Application Subject
Bye! Hope to see you again..
D:\JavaProg>_
```

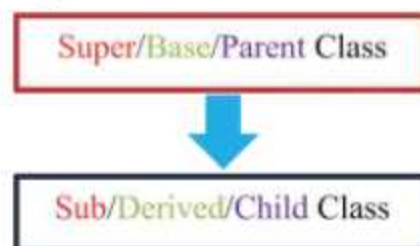
### 9.1.1 TYPES OF INHERITANCE

Java supports different types of inheritance. Some most widely used types of inheritance in java are as follows:

- ❖ Single Inheritance
- ❖ Multi-level Inheritance
- ❖ Hierarchical Inheritance
- ❖ Hybrid Inheritance

#### ❖ Single Inheritance:

This is a simplest type of inheritance in java. In single inheritance, a sub-class is derived from only one super class. It inherits the properties and behavior of a single-parent class. Sometimes it is also known as simple inheritance. We can have following graphical representation of Single Inheritance in Java.



Single Inheritance in Java

As we can see in the above diagram, parent class is inherited in child class. Here, Child class will include all the properties of the base class also. But, no property or behavior of Child class will be inherited by Parent Class. Following example shows the mechanism of single inheritance in a Java :

Following program shows the implementation of Single inheritance in JAVA

```
class Person
{
int age;
String sname;
}
```

```

class Student extends Person
{
int sclass,rollno;
void setData(int age_Input,String sname_Input,int sclass_Input,int
rollno_Input)
{
age=age_Input;
sname=sname_Input;
sclass=sclass_Input;
rollno=rollno_Input;
}
void showRecord()
{
System.out.println("Age:"+age);
System.out.println("Student Name: "+sname);
System.out.println("Student Class:"+sclass);
System.out.println("Roll No:"+rollno);
}
}
class CProgl4
{
    public static void main(String arg[])
    {
        Student obj=new Student();
        obj.setData(16,"Navleen",7,1001);
        obj.showRecord();
    }
}

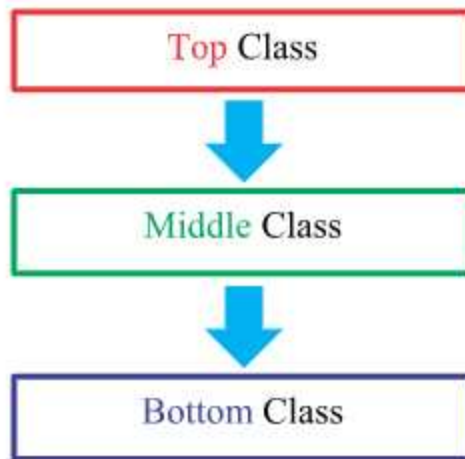
```

#### Compilation, Execution and Output of Program 9.2 (CProgl4.java)

##### ❖ Multi-level Inheritance

This type of inheritance can be viewed as a chain of single inheritance. In multi-level inheritance, a class is derived from a class which is further derived from another class. In simple words, we can say that a class that has more than one parent class, but at different levels, is called multi-level inheritance. A common sub class can not have multiple super classes at a same time in this type of inheritance. Following graphical representation shows the mechanism of multilevel Inheritance in Java.





#### Multilevel Inheritance in Java

As we can see in the above diagram, Top class is inherited by Middle class which is further inherited by Bottom Class. Here, Bottom class will include all the properties of Top and Middle class. But, No property or behavior of Middle class will be inherited by Top Class or Bottom class by Top/Middle class. Following example shows the mechanism of Multilevel inheritance in a Java.

**Following program shows the implementation of Multilevel inheritance in JAVA**

```
class Top
{
    void show()
    {
        System.out.println("This is a method of TOP Class");
    }
}

class Middle extends Top
{
    void display()
    {
        System.out.println("This is a method of Middle Class");
    }
}

class Bottom extends Middle
{
    void print()
    {
        System.out.println("This is a method of Bottom Class");
    }
}
```

```

class CAProg16
{
    public static void main(String arg[])
    {
        Bottom obj=new Bottom();
        obj.show();
        obj.display();
        obj.print();
    }
}

```

### Compilation, Execution and Output of Program 9.3 (CAProg16.java)

```

D:\JavaProg>javac CAProg16.java

D:\JavaProg>java CAProg16
This is a method of TOP Class
This is a method of Middle Class
This is a method of Bottom Class

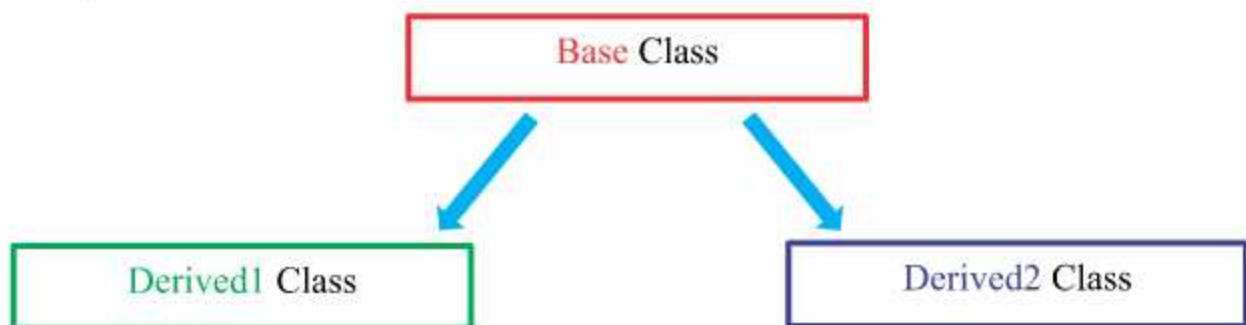
D:\JavaProg>

```

Here, we can see that the method of Top class [show()] and Middle class [display()] are inherited in Bottom class. It allows us to call all the methods [show(), display() and print()] with the object of Bottom class.

### ❖ Hierarchical Inheritance (Tree Inheritance):

When two or more classes inherits a single class, it is known as hierarchical inheritance. This type of inheritance is useful in the case when we want to share the methods or properties of one common base class into multiple derived classes. As we can see in the diagram given below, Derived1 and Derived2 classes inherits the Base class. This kind of layout forms hierarchical inheritance.



### Hierarchical Inheritance in Java

Following example shows the mechanism of Hierarchical inheritance in a Java.

Following program shows the implementation of Hierarchical inheritance in JAVA

```
class Base
{
void show()
{
System.out.println("Base Class");
}
}
class Derived1 extends Base
{
void display()
{
System.out.println("Derived 1 Class");
}
}
class Derived2 extends Base
{
void print()
{
System.out.println("Derived 2 Class");
}
}
class CAProg17
{
    public static void main(String arg[])
    {
        Derived1 obj1=new Derived1();
        obj1.show();
        obj1.display();
        Derived2 obj2=new Derived2();
        obj2.show();
        obj2.print();
    }
}
```



### Compilation, Execution and Output of Program 9.4 (CAProg17.java)

```
D:\JavaProg>javac CAProg17.java

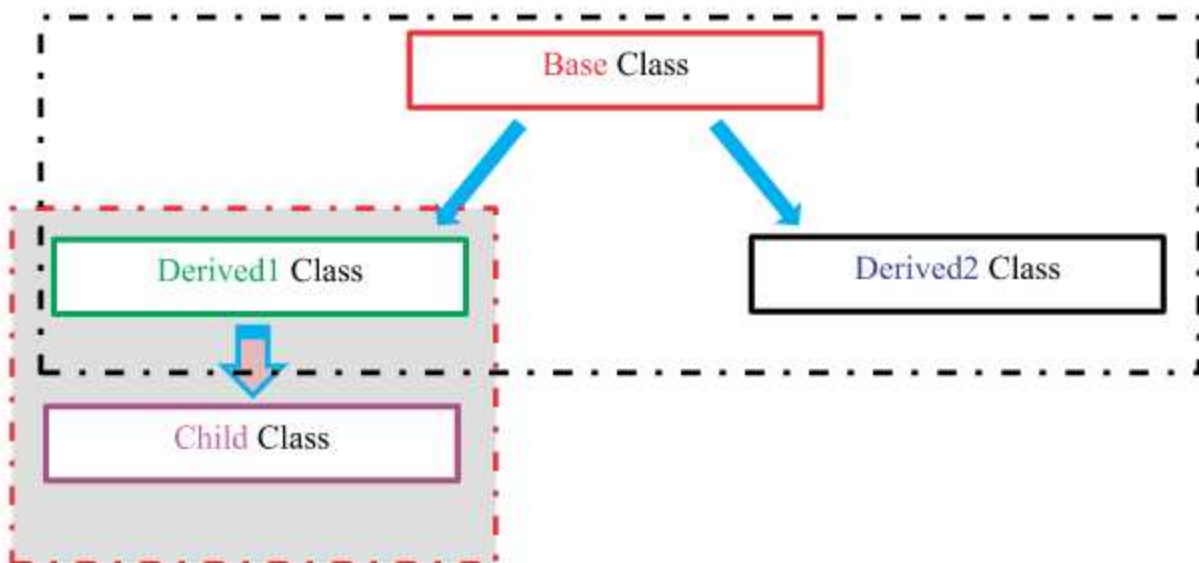
D:\JavaProg>java CAProg17
Base Class
Derived 1 Class
Base Class
Derived 2 Class

D:\JavaProg>
```

Here, we can see that the method of Base class [show()] is inherited in Both Derived1 and Derived2 class. It allows us to call the method of Base class with the object of either Derived1 class or Derived2 class.

#### ❖ Hybrid Inheritance:

Hybrid means consisting of more than one form of inheritance within one inheritance type. Hybrid inheritance is the combination of any two or more types of inheritance in Java. We can have an example of simple hybrid inheritance in the form of a diagram as shown below :



In this diagram, we use Hierarchical and single Inheritance to form Hybrid Inheritance. Following example shows the implementation of Hybrid inheritance a Java:

Following program shows the implementation of Hybrid inheritance in JAVA

```
class Base
{
void show()
{
System.out.println("Base Class");
}
}
class Derived1 extends Base
{
void display()
{
System.out.println("Derived 1 Class");
}
}
class Derived2 extends Base
{
void print()
{
System.out.println("Derived 2 Class");
}
}
class Child extends Derived1
{
void send()
{
System.out.println("This is Child Class here");
}
}
class CProgl8
{
    public static void main(String arg[])
    {
        Child obj1=new Child();
        obj1.show();
        obj1.display();
        obj1.send();
        Derived2 obj2=new Derived2();
        obj2.show();
        obj2.print();
    }
}
```

## Compilation, Execution and Output of Program 9.5 (CAProg18.java)

```
D:\JavaProg>javac CAProg18.java
D:\JavaProg>java CAProg18
Base Class
Derived 1 Class
This is Child Class here
Base Class
Derived 2 Class
D:\JavaProg>_
```

Some other types of inheritance are also available in OOPs. Java does not allow multiple inheritance among classes. We shall discuss multiple inheritance after understanding the concept of Interfaces.

### 9.2 INTERFACES IN JAVA

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body. A class implements an interface and inherit the abstract methods of the interface. Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. In the new versions of Java (Java 8 onwards), Method bodies can also be existed within an interface, but only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements. When a class implements the interface, all the methods of the interface need to be defined in the class. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Note : Abstract methods do not have body, they only have declaration but no definition.

#### Similarities between class and interface:

- ❖ Both class and Interface can contain any number of methods.
- ❖ Both class and Interface written in a file with .java extension, with the name of the interface matching the name of the file.
- ❖ The byte code of both class and Interface appears in a .class file.
- ❖ Both class and Interface appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

#### Difference between class and interface

- ❖ We cannot instantiate an interface.
- ❖ An interface does not contain any constructors.
- ❖ All of the methods in an interface are abstract (only default or static methods can have body).
- ❖ An interface can contain only static and final fields.



- ❖ An interface is not extended by a class; it is implemented by a class.
- ❖ An interface can extend multiple interfaces.

### 9.2.1 DECLARING INTERFACES

The interface keyword is used to declare an interface. Following is the syntax and example of interface:

#### Syntax:

```
interface InterfaceName
{
    // Any number of final, static fields
    // Any number of abstract method declarations
}
```

We can declare the interface using above mentioned syntax as per requirement of our program to use abstract methods. Lets have an example to understand the use of interface in detail.

**Following program shows the implementation of interface in JAVA**

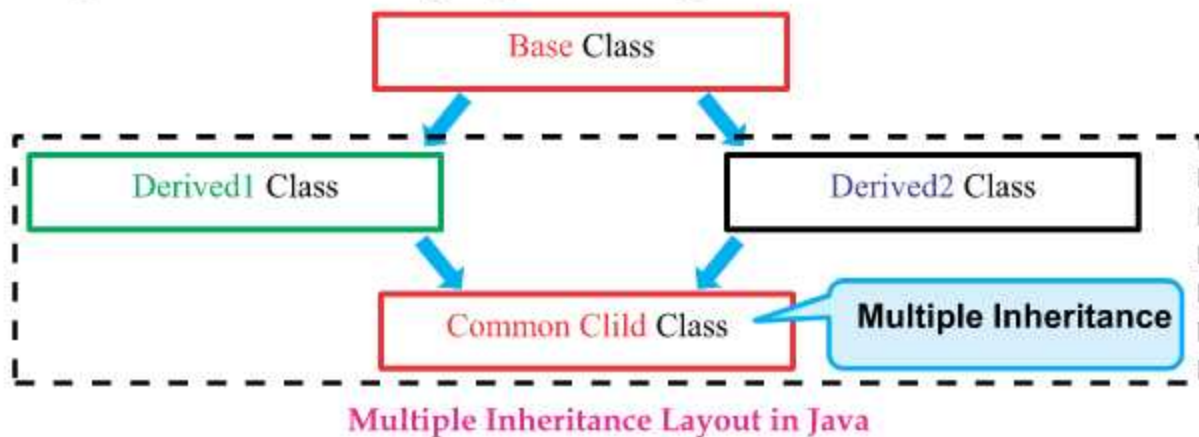
```
interface Student
{
    final int MinAge=18;
    public void show();
}
class CAProg19 implements Student
{
    public void show()
    {
        System.out.println("Welcome to Student Mangement");
        System.out.println("Minimum age of Student is:"+MinAge);
    }
    public static void main(String arg[])
    {
        CAProg19 obj=new CAProg19( );
        obj.show();
    }
}
```

## Compilation, Execution and Output of Program 9.6 (CAProg19.java)

```
D:\JavaProg>javac CAProg19.java
D:\JavaProg>java CAProg19
Welcome to Student Mangement
Minimum age of Student is:18
D:\JavaProg>
```

### ❖ Multiple Inheritance in Java

Java does not support multiple inheritances among classes due to ambiguity. For example, consider the following diagram of multiple inheritance.



The above Diagram shows the mechanism of multiple inheritance in OOPs application. If this layout is used in java classes, it gives error because the compiler cannot decide which method of base class is to be invoked. It is so because methods of base class are derived through two classes named Derived1 and Derived2. Due to this reason, Java does not support multiple inheritances at the class level but can be achieved through an interface.

Multiple Inheritance among interfaces can be shown as follows :

**Following program shows the implementation of multiple inheritance in JAVA**

```
interface Base
{
    public void show();
}
interface Derived1 extends Base {
    public void display();
}
interface Derived2 extends Base {
```

```

public void print();
}
class CommonChild implements Derived1,Derived2 // Multiple Inheritance
{
    public void show()
    {
        System.out.println("Base Class");
    }
    public void display()
    {
        System.out.println("Derived 1 Class");
    }
    public void print()
    {
        System.out.println("Derived 2 Class");
    }
}
class CAProg20
{
    public static void main(String arg[])
    {
        CommonChild obj=new CommonChild();
        obj.show();
        obj.display();
        obj.print();
    }
}

```

#### Compilation, Execution and Output of Program 9.7 (CAProg20.java)



```

D:\JavaProg>javac CAProg20.java

D:\JavaProg>java CAProg20
Base Class
Derived 1 Class
Derived 2 Class

D:\JavaProg>

```

Here we can see, we have extended one common base interface into two different Derived Interfaces which are further inherited in one common class. Multiple Inheritance is possible in java using interface only.



### 9.3 SUPER KEYWORD IN JAVA

Super keyword allows us to access the members of superclass. Common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name. We can use the concept of super in two different forms.

1. **super keyword:** This scenario occurs when a derived class and base class has same members. In that case there is a possibility of ambiguity for the JVM. super keyword, in such a case, is used to define which member is exactly needed to be invoked. Some highlights about super keywords are as under:

- ❖ To call methods of the superclass that is overridden in the subclass.
- ❖ To access attributes (fields) of the superclass if both superclass and subclass have attributes with the same name.

We can understand the concept of super keyword using following example

Following program shows the use of super keyword in JAVA

```
class Base
{
void show()
{
System.out.println("Base class show Method");
}
}
class CAProg21 extends Base
{
void show()
{
super.show(); //Accessing Base class method using super
System.out.println("Derived class show Method");
}
public static void main(String arg[])
{
CAProg21 obj=new CAProg21();
obj.show();
}
}
```

Compilation, Execution and Output of Program 9.8 (CAProg21.java)

```
D:\JavaProg>javac CAProg21.java
D:\JavaProg>java CAProg21
Base class show Method
Derived class show Method
D:\JavaProg>
```

**2. Super method :** super keyword can also be used as a method to access the parent class constructor. We can call parameterized as well as non-parameterized constructors using super method. This method plays a very significant role as we have already studied that if a class defines a constructor with parameters then its default constructor is not automatically generated. So, when we use a constructor in inheritance and our base class is having a parameterized constructor then it becomes compulsory to call the base class constructor in the derived class and pass the values to the parameters. A solution to this issue is that, we can use the super method to explicitly invoke the constructor of the base class in the derived class. There are a few important points about the super method:

- ❖ super() method must be the first statement in the derived class constructor; otherwise, a compilation error would be displayed.
- ❖ When we explicitly place super in the derived class constructor, the Java compiler doesn't call the default constructor of the parent class (base class).

We have understood the concept of inheritance so far. Let's have a closer look at the super method in the form of an example:

**Following program shows the use of super method in JAVA**

```
class Base
{
    Base(int id) //base class constructor with parameters
    {
        System.out.println("Student id is: "+id);
    }
}
class Derived extends Base
{
    Derived(int sid) //derived class constructor with parameters
    {
        super(sid); //Passing parameters to base class constructor
    }
    void show(String sname)
    {
        System.out.println("Student name is: "+sname);
    }
}

class CAProg22
{
    public static void main(String arg[])
    {
        Derived obj=new Derived(1001);
        obj.show("Shivpreet");
    }
}
```

## Compilation, Execution and Output of Program 9.9 (CAProg22.java)

```

D:\JavaProg>javac CAProg22.java

D:\JavaProg>java CAProg22
Student id is: 1001
Student name is: Shivpreet

D:\JavaProg>_

```

We can use super keyword or super method similarly in any type of inheritance for any of the element like fields, methods etc.

### 9.4 ABSTRACTION IN JAVA

Abstraction is a process of hiding the implementation details and showing only functionality to the user. In other words we can say, it shows only essential things to the user and hides the internal complexities. For example, deriving a car is just about using the controls like steering, accelerator, brakes, gears or other electronic control. Internal functioning of engine remain hidden from the user.

Abstraction can be achieved using either abstract classes or interfaces in Java. The abstract keyword is a non-access modifier, used for classes and methods: We can classify abstraction in following types:

- ❖ **Abstract class:** It is a restricted class that cannot be used to create objects. To access the members of this class, it must be inherited from another class.
- ❖ **Abstract methods:** These methods can only be used in an abstract class, and does not have a body. The body is provided by the subclass. (We have already introduced this concept in interface section of this chapter)

Lets understand the concept of abstract class as an example

**Following program shows the use of abstract class and abstract method in JAVA**

```

abstract class Base
{
    abstract void show(); //abstract method
    void display()
    {
        System.out.println("This is not an abstract function");
    }
}

class CAProg23
{
    public static void main(String arg[])
    {
        Base obj=new Base();
        obj.display();
    }
}

```



### Compilation, Execution and Output of Program 9.10 (CAProg23.java)

```
Command Prompt
D:\JavaProg>javac CAProg23.java
CAProg23.java:13: Base is abstract; cannot be instantiated
    Base obj=new Base();
                ^
1 error
D:\JavaProg>
```

As we can see, this program can not be executed because abstract class can not be instantiated. We can use the same program in correct way as given below:

**Following program shows the correct use of abstract class and abstract method in JAVA**

```
Class Base
{
    abstract void show(); //abstract method
    void display()
    {
        System.out.println("This is not an abstract function");
    }
}
class CAProg23 extends Base
{
    void show() //defination of abstract method
    {
        System.out.println("This is an abstract function");
    }
    public static void main(String arg[])
    {
        CAProg23 obj=new CAProg23();
        obj.show();
        obj.display();
    }
}
```

## Compilation, Execution and Output of Program 9.11 (CAProg23.java)

```
D:\JavaProg>javac CAProg23.java
D:\JavaProg>java CAProg23
This is an abstract function
This is not an abstract function
D:\JavaProg>_
```

In the above given example, we can see the correct way to declare and access the abstract class and abstract method in a class.

### 9.5 METHOD OVERRIDING IN JAVA

We have already studied about method overloading where multiple methods are declared with same name but having different number of arguments or different data type of arguments within a same class. But, If subclass has the same method as declared in the parent class, it is known as method overriding in Java. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding. Some basic rules for method overriding are as follows :

#### Rules for Java Method Overriding:

- ❖ The method must have the same name as in the parent class
- ❖ The method must have the same parameter as in the parent class.
- ❖ There must be an IS-A relationship (inheritance).

Lets understand method overriding in JAVA in the form of an example in detail :

**Following program shows the implementation of method overriding in JAVA**

```
class Base
{
void display()
{
System.out.println("Base class is here");
}
}
class Derived extends Base
{
void display() //Method overriding
{
System.out.println("Derived class is here");
}
}
```

```

class CAProg24
{
    public static void main(String arg[])
    {
        Derived obj=new Derived();
        obj.display();
    }
}

```

### Compilation, Execution and Output of Program 9.12 (CAProg24.java)

```

D:\JavaProg>javac CAProg24.java
D:\JavaProg>java CAProg24
Derived class is here
D:\JavaProg>

```

This example explain the use of method overriding in JAVA. We can not override static methods. The main method can also not be overridden.

### Difference between method overloading and method overriding:

There are many differences between method overloading and method overriding in java as given below:

No.	Method Overloading	Method Overriding
1)	Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2)	Method overloading is performed within same class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3)	In case of method overloading, parameters must be different.	In case of method overriding, parameters must be same.



4)	Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
5)	In java, method overloading can't be performed by changing return type of the method only.	Return type must be same in method overriding.

## 9.6 FINAL CLASS, METHOD AND VARIABLES IN JAVA

final keyword is one of the most important keywords in Java that can be used with entities in Java to restrict their use. We can use it with class, methods, variables. Java final keyword is a non-access specifiers that is used to impose some restriction on class, variable, and method. If we initialize a variable with the final keyword, then we cannot modify its value. If we declare a method as final, then it cannot be overridden by any subclasses. And, if we declare a class as final, we restrict the class from being inherited. We can graphically represent the use of final keyword as follows

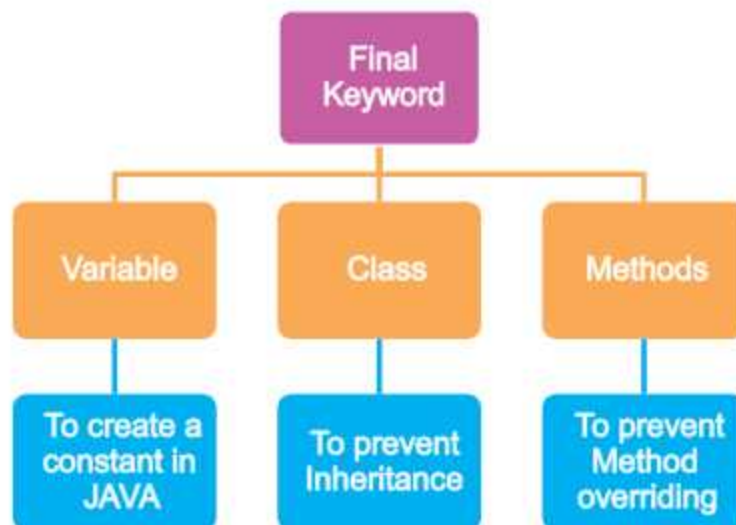


Fig : Different uses of final Keyword

Lets discuss the use of this final keyword with different elements in JAVA.

### ❖ final variable :

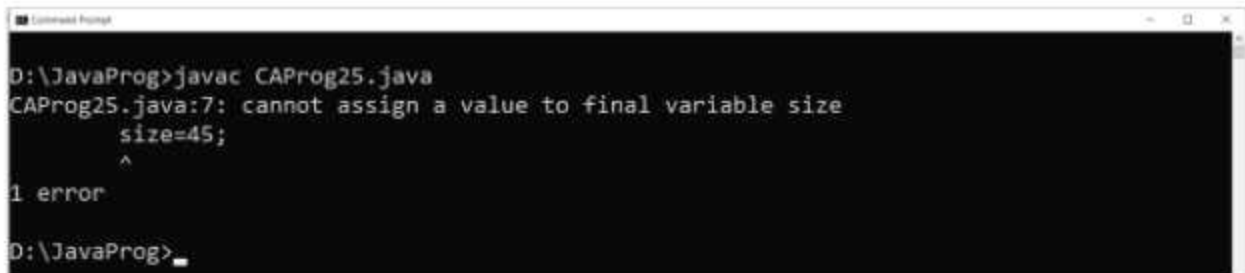
If you declare any variable as final, its value can't be modified. This variable would become a constant. This also means that we must initialize a final variable while declaration because after that no changes are allowed in final variable. If the final variable is a reference, this means that the variable cannot be re-bound to reference another object, but the internal state of the object pointed by that reference variable can

be changed i.e. we can add or remove elements from the final array or final collection. It is a good practice to represent final variables in all uppercase, using underscore to separate words. Lets understand the use of final variable in the form of an example:

**Following program shows the use of final variable in JAVA**

```
class CAProg25
{
    public static void main(String arg[])
    {
        final int size=40; //final variable
        System.out.println("Size of class is: "+size);
        size=45;
    }
}
```

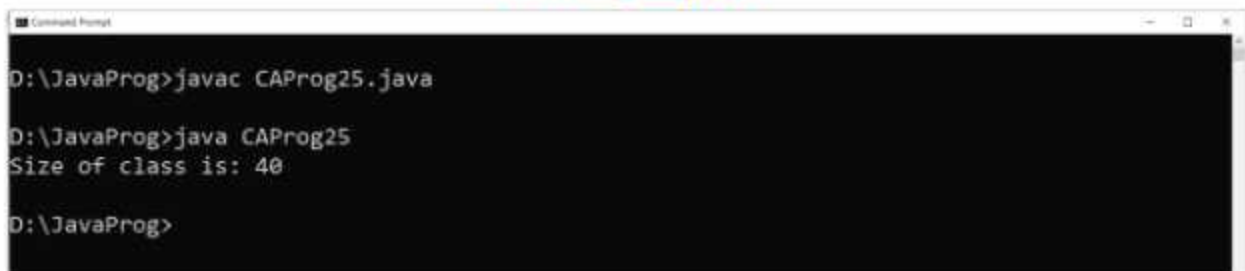
**Compilation, Execution and Output of Program 9.13 (CAProg25.java)**



```
D:\JavaProg>javac CAProg25.java
CAProg25.java:7: cannot assign a value to final variable size
    size=45;
    ^
1 error
D:\JavaProg>
```

Here we can clearly see that no change is allowed in final variable after declaration. If we delete the statement `size=45;` then the program will be executed correctly and the output would be as given bellow:

**After Correction, Compilation, Execution and Output of Program 9.13 (CAProg25.java)**



```
D:\JavaProg>javac CAProg25.java
D:\JavaProg>java CAProg25
Size of class is: 40
D:\JavaProg>
```

Here we can see the output of the program after removing the said statement.

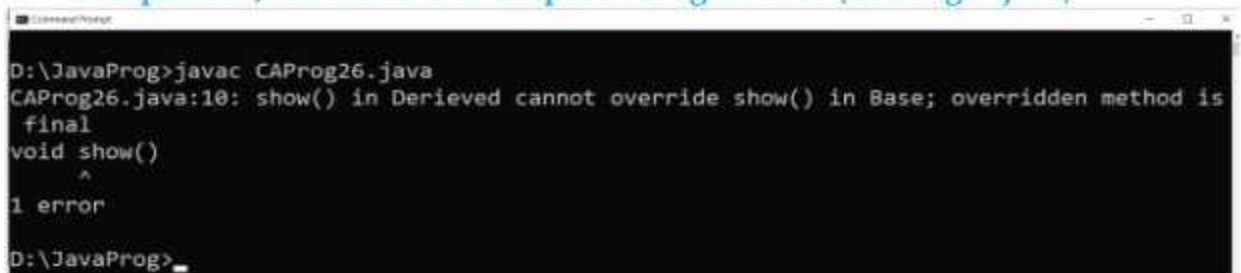
### ❖ Final Methods:

When a method is declared final keyword, it is called a final method. A final method cannot be overridden. We must declare methods with the final keyword for which we are required to follow the same implementation throughout all the derived classes. It is a requirement of several applications where any of the change is essentially required to be restricted. We can have an example of final method as follows:

**Following program shows the use of final method in JAVA**

```
class Base
{
    final void show() //final method
    {
        System.out.println("Base Function");
    }
}
class Derived extends Base
{
    void show() //overriding final method which shows error
    {
        System.out.println("Derived Function");
    }
}
class CAProg26
{
    public static void main(String arg[])
    {
        Derived obj=new Derived();
        obj.show();
    }
}
```

### Compilation, Execution and Output of Program 9.14 (CAProg26.java)



```
D:\JavaProg>javac CAProg26.java
CAProg26.java:10: show() in Derived cannot override show() in Base; overridden method is
    final
void show()
    ^
1 error
D:\JavaProg>
```



As we can see in the program output, no final method can be overridden in any of the derived class.

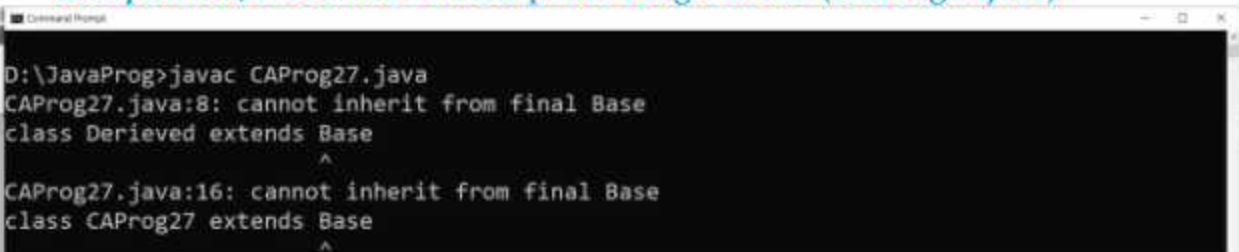
#### ❖ Final classes:

When a class is declared with final using final keyword, this class is restricted to be extended in any of the sub class. These type of classes are useful to prevent the overriding of class methods in derived classes. We can have an example of final class as follows:

**Following program shows the use of final class in JAVA**

```
final class Base //final class
{
void show1()
{
System.out.println("Base Function");
}
}
class Derieved extends Base //shows error as final class cannot be inherited
{
void show2()
{
System.out.println("Derived Function");
}
}
class CAProg27
{
    public static void main(String arg[])
    {
        Derived obj=new Derived();
        obj.show2();
    }
}
```

#### Compilation, Execution and Output of Program 9.15 (CAProg27.java)



```
D:\JavaProg>javac CAProg27.java
CAProg27.java:8: cannot inherit from final Base
class Derieved extends Base
    ^
CAProg27.java:16: cannot inherit from final Base
class CAProg27 extends Base
    ^
```

Here we can see that final class is not allowed to be inherited. If we try to do so, an error message stating the same is being displayed.



## **Points to Remember**

1. Inheritance is a feature or a process in which new classes are created from the existing classes.
2. Class is defined as a collection of objects sharing common properties. It is a kind of blueprint of objects created in a Java program.
3. Sub Class inherits the options from another class including inheritable fields and behaviors of its Super class.
4. Sub Class is also known as derived class.
5. Super Class represents fields or methods which are inheritable by a subclass.
6. Reusability is a technique of reusing methods of the existing class in the new created class.
7. Single Inheritance, Multi-level Inheritance, Hierarchical Inheritance, Hybrid Inheritance are some of the types of Inheritance.
8. Hybrid means consisting of more than one form of inheritance within one inheritance type.
9. Java does not allow multiple inheritance among classes.
10. The interface in Java is a mechanism to achieve abstraction.
11. Method bodies can also be existed within an interface, but only for default methods and static methods.
12. We cannot instantiate an interface and it does not contain any constructors.
13. Multiple inheritance can be achieved interfaces in JAVA.
14. Super keyword allows us to access the members of superclass.
15. Super method can be used to access the constructor of parent class.
16. Abstraction is a process of hiding the implementation details and showing only functionality to the user.
17. If we declare a method as final, then it cannot be overridden by any subclasses.





- iii. Explain super keyword.
- iv. What is method overriding?
- v. Write a short note on final variable?

**Que:4 Long Answer type Questions:**

- i. What is inheritance? Explain any three types of inheritance
- ii. What do you mean by abstraction? Explain abstract elements in Java.
- iii. What do mean by final keyword? Explain Final class and Method with example.
- iv. Explain multiple inheritance with suitable example?

