- **Software** is a set of instructions or programs that allow a computer system to perform specific tasks. It is an intangible component of a computer system that enables the hardware to carry out various operations and provide functionality to users. Software is created by developers using any programming languages and tools that play a crucial role in enabling computers or computing devices to carry out various functions and operations.
    - **System software** is responsible for managing and controlling the computer's hardware and resources. It includes the operating system, device drivers, and utilities.
    - **Application software** is used to perform specific tasks, such as word processing, web browsing, and playing games.

- **Introduction to OOPs:** Object-oriented programming is based on the concept of "objects," which combine data and code into a single unit. Data is in the form of fields or attributes, and code is in the form of methods or procedures.

Object-oriented programming is based on various concepts, such as

1. **Objects:** An object is a real-world entity. It is a member of the class. Each object has an identity, a behaviour, and a state.

2. **Class:** A class is a blueprint or group of objects that have common properties. It represents a logical entity in code that defines the properties and behaviours of objects.

3. **Method:** A method is a block of code that performs a specific task. When we divide a large programme into basic building blocks known as methods,
    - → Methods improve the reusability and readability of the code.
    - → Methods reduce code redundancy and make the code more modular.
    - → Methods reduce the complexity of a large programme and optimise the code.

4. **Abstraction:** An abstraction is a process of hiding the implementation details of an object from the user and showing only functionality to the user.

5. **Encapsulation:** Encapsulation is the technique of combining the data and the member functions of an object into a single unit. Encapsulation provides data protection, modularity, flexibility, and controlled access to code.

6. **Inheritance:** Inheritance is a mechanism by which a new class acquires all the properties and behaviours of an existing parent class.

7. **Polymorphism:** Polymorphism is a concept by which we can perform a single action in different ways.

   There are two types of polymorphism:

   - → Compile the time polymorphism that can be achieved by the method of overloading.
   - → Runtime polymorphism that can be achieved by the methods overriding.

- **Object-oriented programming** allows us to better represent the real world in software programmes by modelling objects and their interactions. It improves software development productivity and maintainability because object-oriented programming supports scalability, making it suitable for both small and large applications. and it also facilitates parallel development, enabling simultaneous teamwork, which promotes organised and efficient software development processes.

- ▪ **Introduction to java:** Java is a high-level, object-oriented, robust, and secure programming language. It was developed by Sun Microsystems in 1995 and is now one of the most popular programming languages in the world. Java is used to develop a wide variety of applications, including web applications, mobile apps, desktop applications, and embedded systems. It is also used in the cloud and in big data processing.

- James Gosling is known as the father of Java.
- Flavours of Java: Core Java, Advanced Java, Android Java

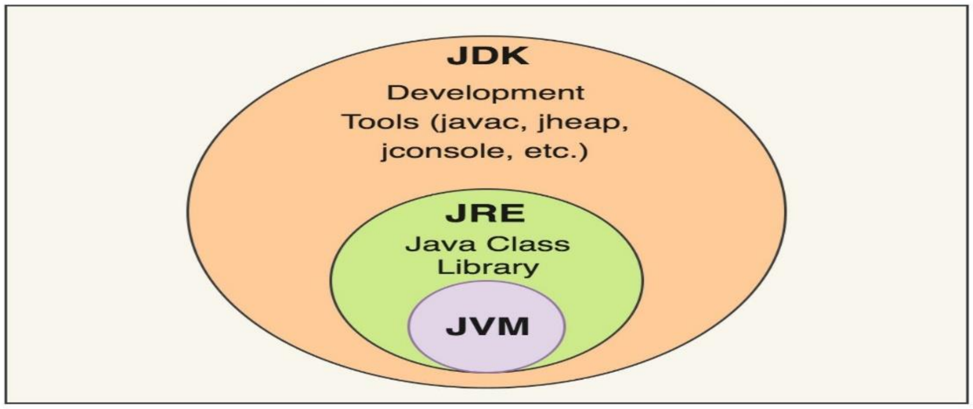
- ▪ **Features of java programming language:**

- **Open source:** Java is an open-source language, which means that it is free to use and distribute. There is a large community of Java developers who contribute to the language and its ecosystem.
- **Object-Oriented:** Java is a fully object-oriented programming language, promoting the use of classes and objects, which make it easier to organize and manage code.
- **Platform Independence:** Java is a platform-independent language, which means that Java programmes can run on any operating system that has a JVM. This makes Java a good choice for developing cross-platform applications.
- **Embedded:** Java can be used to develop embedded systems, such as those found in cars, TVs, medical devices, and various other types of embedded hardware.
- **Large library & framework**: Java has a large and comprehensive library of classes and frameworks. This makes it easy to develop a wide variety of applications.
- **Compiler and interpreter:** Java is both a compiled and interpreted language. Java programs are first compiled into bytecode, which is then interpreted by the JVM.
- **Simple and Clear Syntax:** Java has a relatively simple and readable syntax, making it accessible to developers and reducing the likelihood of errors.

- **Robust:** Java incorporates features like strong type checking, automatic memory management, garbage collection, and exception handling, which contribute to its robustness and help to prevent common programming errors and vulnerabilities.
- **Security:** Java has built-in security features such as classloaders, bytecode verification, a security manager, and secure sockets to ensure secure execution of code.
  - **Bytecode verification:** Java bytecode is verified before it is executed, which helps to prevent malicious code from being executed.
  - **Security manager:** The security manager allows the system administrator to control what resources an application can access.
  - **Secure sockets:** Java provides secure sockets, which provide a secure way for applications to communicate over a network.

- **Strong type checking**: Java has strong type checking, which means that the compiler ensures that the types of variables and expressions are compatible. This helps to prevent errors at compile time and can help to improve the reliability of programs.

- **Garbage collection:** Java Garbage collection used to automatically manage memory allocation and deallocation. This frees developers from having to worry about memory management, which can help to improve the reliability of programs.



Structure of java programming language:

Documentation section

Package statement

Import statement

Interface statement

Class class_name

```
 {
   public static void main(String args[])
    {
          System.out.println("Welcome to java programming");

      }

}
```

To compile: javac fileName.java

Run: java className

Class: class is used to declare class.

Public: It is an access modifier which represent visibility.

Static: It is used to declare static methods and static variables.

Void: It is the return type of the methods.

Main: It represents the starting point of the program.

String args[]: It is used for command line argument.

```
//Find biggest from three numbers;

import java.util.Scanner;

class Biggest
  {
      int biggestNumber(int a,int b,int c)
        {
            int G=a;
            if(b>G)
              {
                G=b;
              }
            if(c>G)
              {
                G=c;
              }
            return(G);
        }
  public static void main(String args[])
       {
        Scanner scan=new Scanner(System.in);
        System.out.println("Enter three numbers=");
        int x=scan.nextInt();
        int y=scan.nextInt();
        int z=scan.nextInt();
        Biggest obj=new Biggest();
        int Big=obj.biggestNumber(x,y,z);
        System.out.print("Biggest Number= "+Big);
       }
  }
```

Components of java class:

1. variables
2. Methods
3. Constructors
4. Blocks (Instance block and static block)
5. Nested class and interface

Note: object and package are not components of a Java class.

Difference between Java, C++ and C languages.

Java	C++	C
The java is based on both C and C++.	The C++ is based on C.	The C is based on BCPL **(Basic Combined Prog. Languages)**
Object oriented language	Object oriented language	Procedural language
The code is executed by JVM (java virtual machine).	The code is executed by directly.	The code is executed by directly.
Java is platform independent because of byte code.	Platform dependent.	Platform dependent.
It uses both compiler and interpreter.	Compiler only	Compiler only
Java generates .class file.	C++ generates .exe files.	C generates .exe and .bak file.
The java source code file has .java extension.	The C++ source code file has .cpp extension.	The C source code file has .c extension.
Java does not support pointer.	It supports pointer.	It supports pointer.
It supports constructor only.	It support both constructor and destructor.	It does not support constructor and destructor.
Java is uses for web application, mobile application windows application.	C++ is widely used for develop system programming.	C is widely used for develop drivers and operating system.

- **A keyword** is a reserved word that has a predefined meaning in the Java programming language. Keywords cannot be used as variable names, method names, class names, or any other identifier. There are 67 keywords in Java.

- **An identifier** refers to the name given to a variable, method, class, interface, or other program elements.

- **A constant** is a fixed value that cannot be change during the equation of the program. To declare a constant in Java, you use the final keyword.

Control Statements

Control statements are a fundamental part of programming languages that allow programmers or developers to control the flow of a program. They determine how and when certain blocks of code are executed based on specified conditions or criteria.

1. Decision making statements (**if, if_else, else_if_ladder, nested_if and switch**)
2. Loop statements (**while, do_while, for, and for_each**)
3. Transfer statements (break, continue, return)

For_each loop: it is used for traverse the array or collection until the last element. This loop eliminates the possibility of programming errors.

Syntax:

```
for(data_type variable : array/collection)
    {       Body of loop    }
```

Operators: An is a symbol which is used to perform operations on operands.

1. Unary: Incremental and Decremental operators
2. Binary: Arithmetic operators (+, - , *, / , %)
   Assignment operators (=, +=, -=, /=, *=)
   Relational or Comparison operators (==, !=, >, <, >=, <=)
   Logical operators (&&, ||, !)
   Bitwise operators(&, |, ^, ~, <<, >>)
3. Ternary: Conditional operators(val=(condition)? True_statement : False_statement;)

- A variable is the name of the memory location. It is the container that holds the value while the program is executed. It is assigned with data types."

- **Local variables** are declared inside the body of a method and can only be used within that method. They are created when the method is called and destroyed when the method returns.

- **Instance variables** are declared inside a class but outside of any method. They are associated with individual instances (Object) of the class and are created when an object is created and destroyed when the object is garbage collected.

- **Static variables** are declared with the static keyword inside a class but outside of any method. They are associated with the class itself, not with any particular object of the class and are created when the class is loaded and destroyed when the class is unloaded.

This means that each object of a class has its own copy of an instance variable, while there is only one copy of a static variable for the entire class.

- **Final:** The final keyword in Java is used to restrict certain actions in programming. It is applied to variables, methods, or classes to make them unmodifiable or unextendible.
  - If we declare a variable as final, then we cannot change the value of that variable after it has been initialised.
  - If we declare a method as final, then we cannot override or change the implementation of that method in any subclass. However, subclasses can still inherit and use the method as it is defined in the parent class.
  - If we declare a class as final, then that class cannot be extended or inherited by subclasses.

- Data types in programming determine the type and size of data associated with variables**.**
  - **Primitive data types** are predefined data types in the Java programming language. There are eight primitive data types in Java: boolean, byte, int, float, char, double, long, and short.
  - **Non-primitive data types** are data types that are created by the programmer, such as arrays, classes, strings, and interfaces.

- A constructor is a special type of method whose name is the same as the class name, and it is automatically called at the time of object creation.
  - Every class in Java has at least one constructor.
  - A constructor must have no explicit return type.
  - Constructors cannot be abstract, static, or synchronized.

  1. **A default constructor** is provided by Java if a class doesn't define any constructors explicitly. It takes no arguments and performs minimal or default initialization.
  2. **A parameterized constructor** is defined with parameters and allows custom object initialization during creation.
  3. **A private constructor** is used to restrict object creation. It does not allow a class to create an object outside the class. If a class has a private constructor, we cannot extend that class.

Example:

```
class PrivCons
 {
   private PrivCons()
     {
       System.out.print("Welcome to private costructor");
     }
   public static void main(String args[])
     {
       PrivCons obj=new PrivCons();
     }
 }
```

- **Method:** A method is a block of codes that perform specific task. When we divides a large program into the basic building blocks known as methods.
  - Methods improves reusability and readability of the code.
  - Method reduce code redundancy and make code modular.
  - Method reduce complexity of big program and make optimize the code.

  Syntax:

```
AccessModifer returnType methodName( arguments… )

  {

     //body of methods….

  }
```

- **The super keyword** in Java refers to the superclass of the current class. It can be used to call superclass methods and access the superclass constructor. One of the main uses of the super keyword is to eliminate confusion between superclass and subclass methods with the same name. Note: Use of super() to access superclass constructor

```
class A
 {
   A()
   {
     System.out.println("Welcome to superclass");
   }
 }

 class B extends A
 {
   B()
   {  super();
     System.out.println("Welcome to subclass");
   }
   public static void main(String args[])
   {
     B obj=new B();
   }
 }
```

Output:

```
Welcome to superclass
Welcome to subclass
```

- **Types of Classes in java**
  1. **Final Class:** The final class is a class that is declared with the final keyword. It cannot be extended or inherited by subclasses.

2. **Abstract Class:** An abstract class cannot be instantiated directly, but it can be used as a superclass for other classes. It may contain abstract methods without implementation that must be implemented by its subclasses.

3. **Concrete Class:** A concrete class is a regular class that can be instantiated to create objects and provides implementations for all methods.

4. **Static Class:** A static class is a nested class defined within another class and can be accessed using the outer class name. It cannot access non-static members of the outer class.

5. **Local class:** A local class is a class that is defined within a method. It can only be accessed from within the method in which it is defined.
6. **Inner class:** An inner class is defined within the body of another class. It can access the members (including private) of the outer class.

- **Instance block:**

  - Instance blocks are executed every time an instance of a class is created.
  - Instance blocks are used to initialize the instance data members of a class.
  - Instance blocks do not require any keyword.
  - Instance blocks are executed after the constructor of the class is called.
  - Instance blocks can access both static and non-static variables.

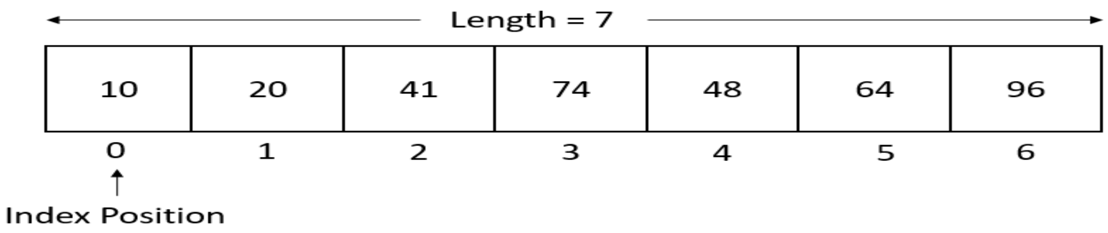
- **Static block:**

  - Static blocks are executed only once, when the class is loaded into the JVM.
  - Static blocks are used to initialize static variables or call static methods.
  - The static keyword is required to define static blocks.
  - Static blocks can only access static variables.

Example:

```
class Blocks
  {
    {
      System.out.println("Welcome to instance block");
      /*Instance block will be call at the time of object creation*/
    }
    static{
      System.out.println("Welcome to static block");
    }
    public static void main(String src[])
      {
       System.out.println("Welcome to main block");
       Blocks obj=new Blocks();
      }                                    Output:       Welcome to static block
  }                                                       Welcome to main block
                                                         Welcome to instance block
```

- **Arrays:** In Java, an array is an object that contains a collection of similar data elements stored at contiguous memory locations. Arrays in Java are mutable, which means we can change the values of elements in an array after the array is created.



Syntax to declare an array:

1. DataType   arrayName[ ]={data elements separated with comma};
2. DataType   [ ]arrayName={data elements separated with comma};
3. Datatype[ ] arrayName={data elements separated with comma};

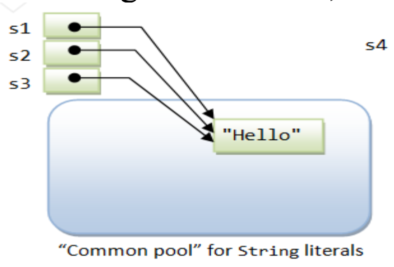
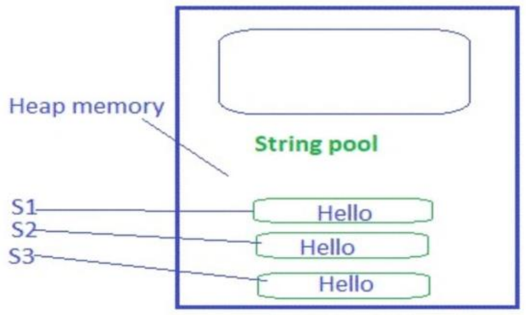
Using new keyword:

- Single dimensional: DataType arrayName[]=new dataType[size];
- Two dimensional: DataType arrayName[][]=new dataType[size][size];

- **String**: In Java, a string is an object that represents a sequence of characters enclosed within double quotes.

Strings are immutable in java which means once we created a string, we cannot change the value.

There are two ways to create a string in java:

String literal	Using new keyword
Ex: String str=”welcome”; When we create a string literal the JVM check the “String constant pool” first. If the string is already exists in the pool, then a reference to the pooled instance is returned. If string is does not exist in the pool, a new string instance is created and placed in the pool. Ex: String s1=”Hello”; String s2=”Hello”; String s2=”Hello”;  s1 s2 s3 s4 "Hello" “Common pool” for String literals	Ex: String str=new String(“welcome”); String object can be created using new keyword. JVM will create a new string object in normal heap memory. Ex: String s1=new String(“Hello”); String s2=new String(“Hello”); String s3=new String(“Hello”);  Heap memory String pool S1 S2 S3 Hello Hello Hello Array of String: String Str[ ]=new String[Size];

StringBuffer: In java StringBuffer is used to create a mutable (Modifiable) string objects. The StringBuffer is same as string except it is mutable. A string which is created by the StringBuffer it can be modified or changed.

Ex: StringBuffer str=new StringBuffer(“Welcome”);

append()= str.append("java"); replace()= str.replace(1,3,”Java”);

insert()=str.insert(1,”java”); reverse()=str.reverse();

Method overloading	Method overriding
Whenever a class contains more then one methods with same name and different types of parameters is known as method overloading.	Whenever we writes a method in Base class and derived class in such a way that method name and parameter must be same known as method overriding.
It is example of compile time polymorphism.	**It is example of run time polymorphism.**
Method overloading occurs in same the class.	Method overriding occurs between Base class and Derived class.
Parameters must be different.	Parameters must be same.

- **Abstraction:** An abstraction is a process of hiding the implementation details of an object from the user and showing only functionality to the user. It can be achieved with either **abstract classes** or **interfaces.**

There are two ways to achieve abstraction in java

1. **Abstract class (0 to 100%)**
2. **Interface (100%)**

Abstract class: A class which is declared as abstract keyword is known as an **abstract class**. It can have abstract and non-abstract methods. We cannot create the object (instantiated) of abstract class, it must be inherited from another class.

- Abstract classes can also have static methods, which are associated with the class itself, not with instances of the class.
- Abstract classes can have final methods. When a method in an abstract class is marked as final, it means that subclasses cannot override or change the implementation of that method. It enforces a fixed behavior for that method in all subclasses.

Ex: **abstract class** ClassName{ …….. }

Abstract Method: A method which is declared as abstract keyword known as abstract method. it does not have a body. The body is provided by the subclass (inherited from).

Ex:

```
abstract class Bank
 {
  abstract void rateOfInterest();
 }
class POSB extends Bank
 {
  void rateOfInterest()
   {
    System.out.println("PostOfficeSavingBank\nInterest="+7.6);
   }
  public static void main(String src[])
   {
    Bank obj=new POSB();//Upcasting
    obj.rateOfInterest();
   }
 }
```

Output:

```
PostOfficeSavingBank
Interest=7.6
```

- **Interface:** An **interface in** is a blueprint of a class. It has static constants and abstract methods. It is declared by using the interface keyword. There can be only abstract methods in the interface, not method body.

There are mainly three reasons to use interface.

- It is used to achieve perfectly (completely) abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

➞ By default interface variables are public, static and final.

➞ By default interface methods are public and abstract.

Ex:

```
interface A
   {
    public abstract void input();
   }
 interface B
      {
        public abstract void output();
      }
 class Rectangle implements A,B
    {
     int l,b;
     public void input()
       {
         l=97;   b=98;
       }
     public void output()
       {
         System.out.println("Rectangle area="+(l*b));
       }
     public static void main(String args[])
       {
         Rectangle obj=new Rectangle();
         obj.input();   obj.output();
       }
 }
```

Abstract	Interface
Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
An abstract class can be extended using keyword "extends".	An **interface can** be implemented using keyword "implements".
An abstract class can have class members like private, protected, public.	Members of a Java interface are public by default.
Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.

The abstract keyword is used to declare abstract class.	**The interface keyword is used to declare interface.**
Variables are not final by default. It may contain non-final variables.	**Variables are final by default in an interface.**

This keyword: The **this** keyword refers to the current object in a method or constructor. The most common use of the **this** keyword is to eliminate the confusion between class attributes and parameters with the same name.

- To invoke current class constructor or method.
- To return the current class object.
- To pass an argument in the method call or constructor call.

Ex:

```
Class T
 {
  int var=98;
  void print()
   {
    int var=97;
    System.out.print("Var="+var+" Var="+this.var);
   }
   public static void main(String src[])
    {
     T obj=new T();
     obj.print();
    }
 }
```

Output:

```
Var=97  Var=98
```

- **Inheritance:** Inheritance is a mechanism in which a new class acquires all properties and behaviors of an existing parent class.

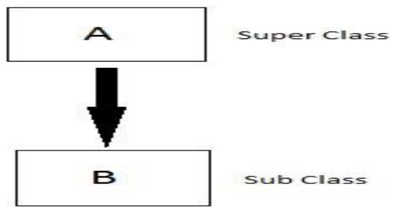
Advantages of Inheritance

- Inheritence improves reusability and readability of the code.
- Minimizing duplicate code.
- Method Overriding can be achieved.
- Inheritence reduce complexity of big program and make optimize the code.

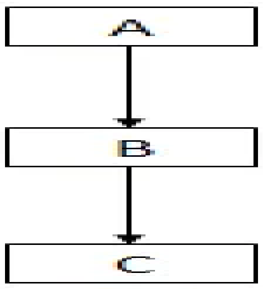
The syntax of Java Inheritance

```
class Subclass_name extends Superclass_name
        {
           //methods and fields
        }
```

- Single Inheritance: **When a class inherits another class, it is known as a single inheritance.**

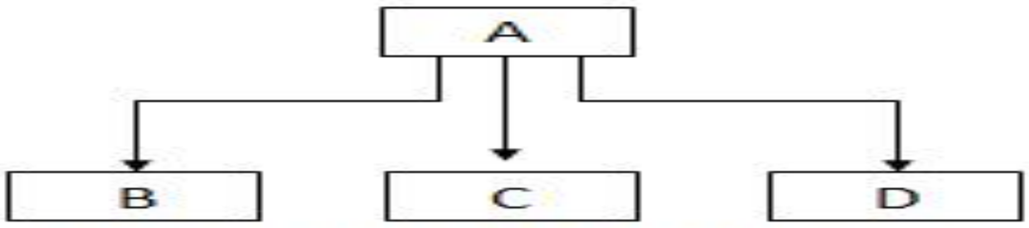


- Multilevel Inheritance: **When there is a chain of inheritance, it is known as multilevel inheritance.**



Multilevel Inheritance

- **Hierarchical Inheritance: When two or more classes inherits a single class, it is known as hierarchical inheritance.**



Hierarchical Inheritance

- Multiple inheritance is not supported in java:

To reduce the complexity and simplify the language, multiple inheritance is not supported in java. Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class. Therefore to prevent this ambiguity java does not support multiple inheritance.

But Interface supports multiple inheritance.

- **Access modifiers** in Java are used to define the accessibility or visibility of classes, methods, variables, and constructors.
  1. **Public:** Public members are accessible from anywhere, both within and outside the class or package.
  2. **Private:** Private members are only accessible within the same class.
  3. **Protected:** Protected members are accessible within the same class, subclasses, and also by inherited form.
  4. **default:** Default members can be accessed from within the class itself and from other classes in the same package.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

- **Java Package:** A java package is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

Built-in packages: Java has already defined some packages and included that in java software, these packages are known as built-in packages or predefined packages. These packages contains a large number of classes and interfaces.

Ex: java.awt, java.io, java.lang, java.net, java.sql, java.util.

User-defined package: User-defined packages are those packages that are designed or created by the developer to categorize classes, interfaces and packages.

Advantage of Java Package

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Improves code reusability.
- Information Hiding.
- Java package provides access protection.
- Java package removes naming collision.

➢ Creating User-defined Package:

```
package packg;                      //Package declaration
public class BigNum
 {
   public int getBigNum(int Ar[])           //package method
     {
      int G=Ar[0];
      for(int i:Ar)
        {
          if(G<i)
           {
            G=i;
           }
        }
      return(G);
     }
    public static void main(String src[])
      {
        System.out.println("Created package has been compiled successfully...!!");
      }
 }
```

Save: className.java **Ex: BigNum.java**

Package compilation: javac –d . className.java **Ex: javac –d . BigNum.java**

Package run: java packageName.className **Ex: java packg.BigNum**

Output: Created package has been compiled successfully...!!

➢ Importing created package

import packg.*;

```
class ArrG extends BigNum

 {

   public static void main(String args[])

     {

       int ar[]={1,23,4,54,56798,56};

       ArrG obj=new ArrG();

       int G=obj.getBigNum(ar);

       System.out.println("Greatest Number= "+G);

     }       }

                ……..OR……..

import packg.BigNum;

class ArrG

 {

   public static void main(String args[])

     {

       int ar[]={1,23,4,54,56798,56};

       BigNum obj=new BigNum();

       int G=obj.getBigNum(ar);

       System.out.println("Greatest Number= "+G);

     }  }
```

Compile: javac fileName.java **Run: java className**

Output:

```
Greatest Number= 56798
```

- **Exception handling:** Exception handling is a mechanism to handle run time errors such as ClassNotFoundException, IOException, SQLException, FileNotFoundException, ArithmeticException, NullPointerException etc.

The main advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application, that’s why we need to handle exceptions.

Core java

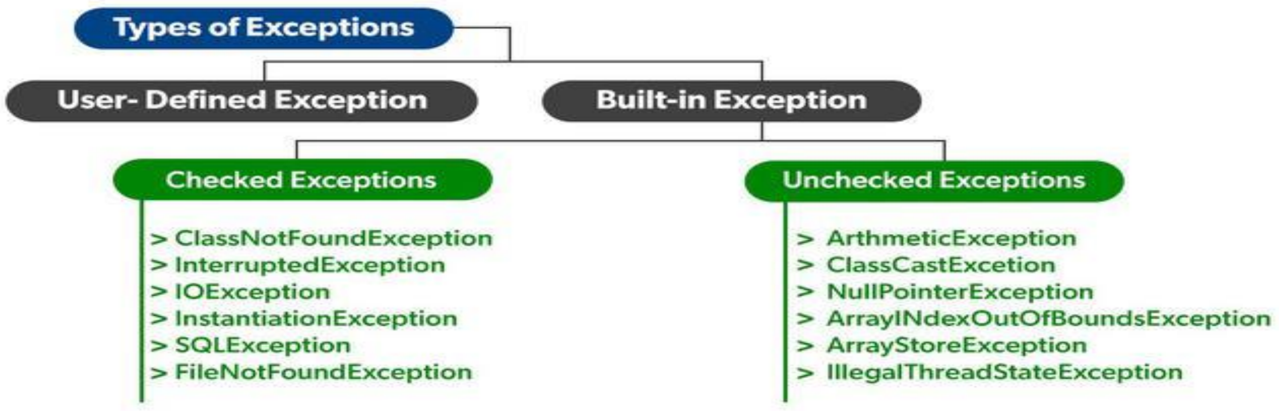
There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception.

Checked Exception **Unchecked Exception** **Error**

- **Checked exceptions** are checked at compile-time. Ex: IOException, SQLException, etc.

- **Unchecked exceptions** are not checked at compile-time, but they are checked at runtime.

- **Error** is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.



Java provides five keywords that are used to handle the exception.

1. **Try :** The "try" keyword is used to specify a block of code where exception may be occur. It means we can't use try block alone. The try block must be followed by either catch or finally.

2. **Catch:** The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

3. **Finally:** The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.

4. **Throw:** The "throw" keyword is used to throw an exception.

5. **Throws:** The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Ex:

```
import java.util.Scanner;
class ExHandling
 {
   static float division(float a,float b) throws ArithmeticException
     {
        if(b==0)
          {
             throw new ArithmeticException();
          }
          float d=a/b;
          return d;
     }
   public static void main(String src[])
    {
      Scanner scan=new Scanner(System.in);
      System.out.print("Enter two numbers= ");
      int x=scan.nextInt();
      int y=scan.nextInt();
      try{
             System.out.println("Division= "+division(x,y));
         }
         catch(ArithmeticException ex)
           {
               System.out.println("Arithmetic exception occur...");
           }
      Finally{
                 System.out.print("program is terminating...");
             }                                    Output:        Enter two numbers= 8      0
        }                                                         Arithmetic exception occur...
    }                                                             program is terminating...
```

Throw **VS** **Throws**

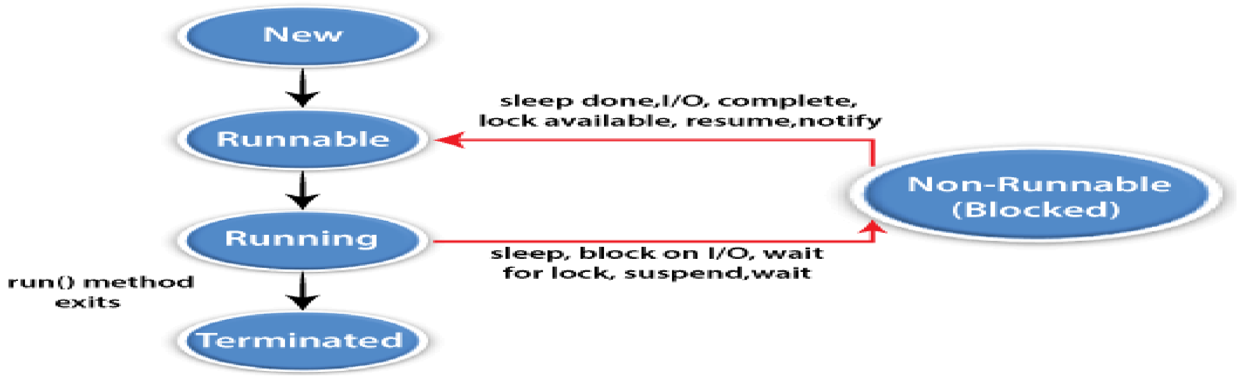
A throw is used to throw an exception explicitly.	A throws is used to declare one or more exceptions, separated with commas.
Throw keyword is followed by the instance variable.	Throws keyword is followed by the exception class.
Throw keyword is used within the method body.	Throws keyword is used with the method signature.

We cannot throw multiple exceptions.	We can declare multiple exceptions.
Only unchecked exceptions propagated using throw keyword.	Only checked exceptions propagated using throws keyword.

- **Multithreading** refers to a process of executing two or more thread simultaneously for maximum utilization of the CPU.

A **Thread** is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously.

Lifecycle of a Thread


New
Runnable
Running
Terminated
Non-Runnable (Blocked)
sleep done,I/O, complete, lock available, resume,notify
sleep, block on I/O, wait for lock, suspend,wait
run() method exits

Advantages of Multithreading

- It **doesn't block the user** because threads are independent and we can perform multiple operations at the same time.
- We **can perform many operations together, so it saves time**.
- Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Threads can be created by using two mechanisms:

1. **Extending the thread class**
2. **Implementing the runnable interface**

Extending the thread class : Thread class provides constructors and methods to create and perform operations on a thread.

We create a class that extends the **Thread** class. This class overrides the run() method available in the Thread class. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

Thread class methods

- **start():** **It is used to start the execution of the thread.**

- **run():** It is used to do an action for a thread.
- **sleep():** It sleeps a thread for the specified amount of time.
- **suspend():** It is used to suspend the thread.
- **resume():** It is used to resume the suspended thread.
- **stop():** It is used to stop the thread.
- **wait():** It sets the thread back in waiting state.
- **notify():** It gives out a notification to one thread that is in waiting state.
- **notifyAll():** It gives out a notification to all the thread in the waiting state.

Ex:

```
class Maxim extends Thread
 {
   @Override
   public void run()
     {
       for(int i=1; i<=5; i++)
         {
          System.out.println("Jai Shri Raam");
         }
     }

 }
class MainClass extends Thread
 {
        @Override
        public void run()
           {
              for(int i=1; i<=5; i++)
                {
                 System.out.println("Jai Shri Krishna");
                }
           }
   public static void main(String src[])
     {
       MainClass O=new MainClass();
       Maxim obj=new Maxim();
       O.start();
       obj.start();
     }
 }
```

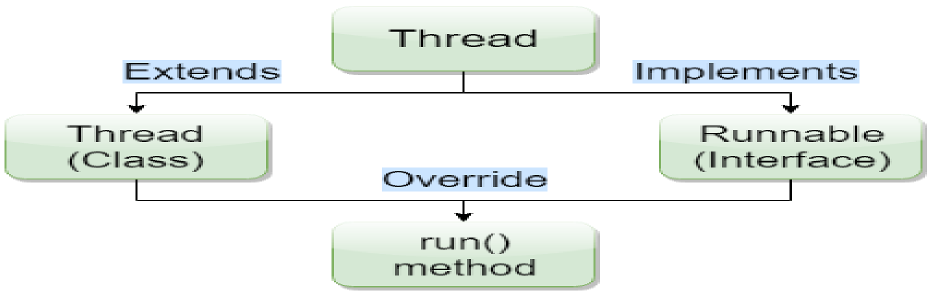
Implementing the runnable interface: We create a class that implements the **Runnable interface** and overrides the run() method available in the Runnable interface. Then we instantiate a Thread class object and pass implemented Runnable interface class object in argument, and call the start() method.

The thread will execute the code which is mentioned in the run() method of the Runnable object passed in its argument.

Ex:

```
class RunIn implements Runnable
 {
  @Override
  public void run()
   {
    for(int i=1; i<=5; i++)
     {
      System.out.println("Shri MannNarayan");
     }
   }
 }
class MainClass
 {
  public static void main(String src[])
   {
    RunIn ob=new RunIn();
    Thread obj=new Thread(ob);
    obj.start();
   }
```

Thread Class vs. Runnable Interface

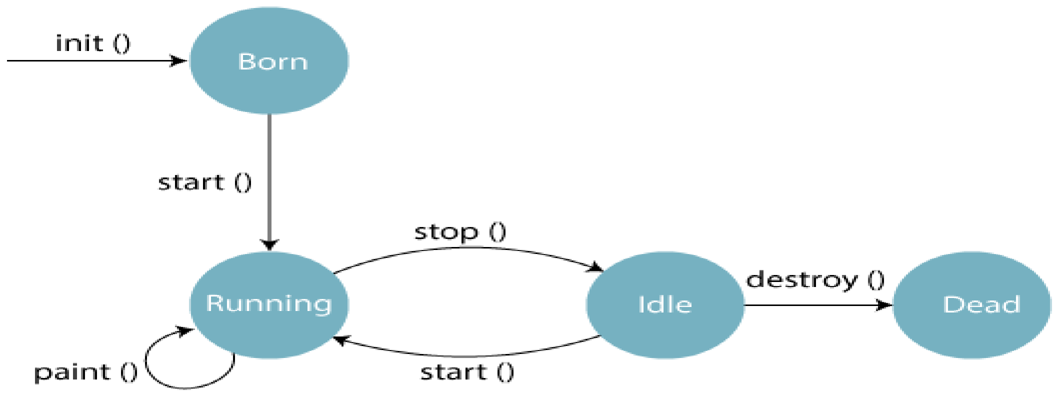


- If we extend the Thread class, our class cannot extend any other class because Java doesn't support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.
- We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield(), interrupt() etc. that are not available in Runnable interface.
- Maintenance of the code is easy if we implement the Runnable interface.

- **Applet** is a special type of program that embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.
  Advantages of applet:
  - It works at client side so less response time.

- Secured.
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac OS etc.

Applet Life Cycle:



Synchronization is the capability to control the access of multiple threads to any shared resource.

It is better option where we want to allow only one thread to access the shared resource.

The synchronization is mainly used to

- To prevent thread interference.
- To prevent consistency problem.

There are two types of synchronization

Process Synchronization **Thread Synchronization**

Synchronized keyword is used to implement synchronization.

Inter-thread communication(Co-operation) is a process in which a thread is paused running in its critical region and another thread is allowed to enter (or lock) in the same critical region to be executed.

It is implemented by following methods of **Object class**:

wait() **notify()** **notifyAll()**

- **Finalize()** is the method of Object class. This method is the garbage collector always calls just before the deletion/destroying the object which is eligible for Garbage Collection to perform clean-up activity.

Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection, or we can say resource de-allocation.

- **Binding:** Connecting a method call to the method body (method implementation) is known as binding.

There are two types of binding

1. Static Binding (also known as Early Binding).
2. Dynamic Binding (also known as Late Binding).

static binding: When type of the object is determined at compiled time, it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

Ex:

```
class Cow
 {
    private void eat()
      {
         System.out.println("Cow is eating...");
     }
  public static void main(String args[])
         {
            Cow obj=new Cow();
            obj.eat();
         }
}
```

Dynamic binding: When type of the object is determined at run-time, it is known as dynamic binding.

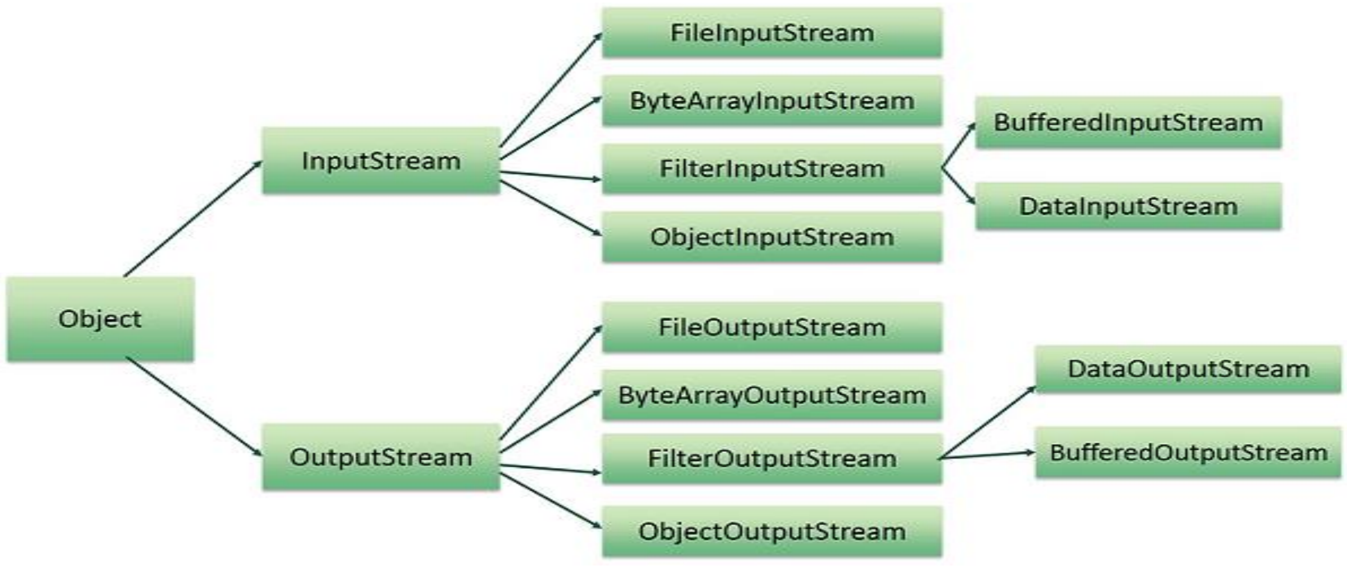
```
class Animal
   {
     void eat()
          {
            System.out.println("animal is eating...");
          }
```

```
    }

class Cow extends Animal
{
     void eat()
        {
          System.out.println("Cow is eating...");
        }

 public static void main(String args[ ])
      {
         Animal obj=new Cow();
         obj.eat();
       }
}
```

File handling: **File handling means performing various operations on a file, like reading, writing, editing etc.**



A stream can be defined as a sequence of data to perform I/O operations on a file.

- **InputStream is used to read data from a source.**
- **OutputStream is used for writing data to a destination.**

We can perform the following operation on a file:

Create a File, Get File Information, Write to a File, Read from a File, Delete a File

Java File Class Methods

Method Name	Description	Return Type
canRead()	It tests whether the file is readable or not.	Boolean
canWrite()	It tests whether the file is writable or not.	Boolean
createNewFile()	It creates an empty file.	Boolean
delete()	It deletes a file.	Boolean
exists()	It tests whether the file exists or not.	Boolean
length()	Returns the size of the file in bytes.	Long
getName()	Returns the name of the file.	String
list()	Returns an array of the files in the directory.	String[]
mkdir()	Creates a new directory.	Boolean
getAbsolutePath()	Returns the absolute pathname of the file.	String

Program for file:

```
import java.io.*;
import java.util.Scanner;
class CreateFile
  {
    public static void main(String args[])
      {
       System.out.println(".....Wecome to java file operations.....");
       while(true)
         {
           Scanner s1=new Scanner(System.in);
           System.out.println("\n 1.for create new file\n 2.for write to a File\n 3.for read from a File");
           System.out.println(" 4.for get File Information\n 5.for delete a File\n 6.for exits");
           System.out.print("Enter your choice=");
           int choice=s1.nextInt();
           if(choice==1)
             {
               System.out.print("Enter the file name= ");
```

```
                Scanner s2=new Scanner(System.in);
                String name=s2.nextLine();
                File obj=new File("E:\\"+name+".txt");
                try{
                    if(obj.createNewFile())
                      {
                        System.out.println("File "+obj.getName()+" has been created successfully...!!");
                      }
                     else
                      {
                        System.out.println("File already exists with desired "+obj.getName()+" name...!!");
                      }
                   }
               catch(Exception x)
                  {
                   System.out.println(x);
                  }
             }
           else if(choice==2)
             {
                try{
                    System.out.print("Enter the file name= ");
                    Scanner s3=new Scanner(System.in);
                    String name=s3.nextLine();
                    FileWriter obj1=new FileWriter("E:\\"+name+".txt");
                    System.out.print("Enter the data= ");
                    String data=s3.nextLine();
                    obj1.write(data);
                    obj1.close();
                    System.out.println("Contents has been written successfully to the file.");
                   }
                catch(Exception x)
                   {
                    System.out.println(x);
                   }
             }
           else if(choice==3)
             {
               try{
                   System.out.print("Enter the file name= ");
                   Scanner s4=new Scanner(System.in);
                   String name=s4.nextLine();
                   FileReader obj2=new FileReader("E:\\"+name+".txt");
                     try{  int r;
                         while((r=obj2.read())!= -1)
                    {
                     System.out.print((char)r);
                    }
                obj2.close();
               }
             catch(Exception x)
              {
               System.out.println(x);
              }
          }
      catch(FileNotFoundException x)
        {
         System.out.println("Sorry file is not avaliable with desired name ");
        }
     }
    else if(choice==4)
     {
        System.out.print("Enter the file name= ");
        Scanner s5=new Scanner(System.in);
        String name=s5.nextLine();
        File obj3=new File("E:\\"+name+".txt");
        if(obj3.exists())
          {
           System.out.println("The absolute path of the file is: " + obj3.getAbsolutePath());
           System.out.println("Is file writeable?: " + obj3.canWrite());
           System.out.println("Is file readable " + obj3.canRead());
           System.out.println("The size of the file in bytes is: " + obj3.length());
          }
        else
          {
          System.out.println("The file does not exist.");
          }
     }
    else if(choice==5)
     {

       System.out.print("Enter the file name for delete= ");
       Scanner s6=new Scanner(System.in);
       String name=s6.nextLine();
       File obj4=new File("E:\\"+name+".txt");
       if(obj4.delete())
         {
          System.out.println("File has been deleted successfully...!!");
         }
       else
```

```
            {
             System.out.println("The file does not exist.");
            }
        }
        else if(choice==6)
        {
          System.out.println("Exits...");
          break;
        }
       else
        {
          System.out.println("Wrong choice...!!");
        }
     }
   }
}
```

- **Collection:** The collection is a framework that provides an architecture to store and manipulate the group of objects.

Java collections can achieve all the operations that we perform on a data such as searching, sorting, insertion, manipulation and deletion.
The java.util package contains all the classes and interfaces for the collection framework.

Ex:

```
import java.util.ArrayList;
class JavaCollection
 {
   public static void main(String args[ ])
    {
      ArrayList<String> Arr=new ArrayList<String>();
      Arr.add("Patna");
      Arr.add("Muzaffarpur");
      System.out.println(Arr);
      Arr.add("Delhi");
      System.out.println(Arr);
      Arr.add(2,"Bettiah");
      System.out.println(Arr);
      Arr.remove(2);
      System.out.println(Arr);
      Arr.set(1,"Champaran");
      System.out.println(Arr);
      System.out.println(Arr.get(1));
    }
```

```
Output:

[Patna, Muzaffarpur]

[Patna, Muzaffarpur, Delhi]

[Patna, Muzaffarpur, Bettiah, Delhi]

[Patna, Muzaffarpur, Delhi]

[Patna, Champaran, Delhi]

Champaran
```

```
 }
```

Ex:

```
import java.util.LinkedList;
class JavaCollection
 {
   public static void main(String args[])
    {
      LinkedList<String> L=new LinkedList<String>();
      L.add("Bihar");
      L.add("UP");
      System.out.println(L);
      L.addFirst("Delhi");
      System.out.println(L);
      L.addLast("Kolkata");
      System.out.println(L);
      L.add(2,"Chandigarh");
      System.out.println(L);
      L.removeLast();
      for(String s:L)
       {
         System.out.println(s);
       }
    }
 }
```

```
Output:

[Bihar, UP]

[Delhi, Bihar, UP]

[Delhi, Bihar, UP, Kolkata]

[Delhi, Bihar, Chandigarh, UP, Kolkata]

Delhi

Bihar

Chandigarh

UP
```

Ex:

```
import java.util.Stack;
class JavaCollection
 {
   public static void main(String args[ ])
    {
      Stack<String> S=new Stack<>();
      S.push("Patna");
      S.push("Motihari");
      System.out.println(S);
      S.push("Chanpatia");
      System.out.println(S);
      S.pop();
      System.out.println(S);
    }
 }
```

```
Output:

[Patna, Motihari]

[Patna, Motihari, Chanpatia]

[Patna, Motihari]
```

Ex:

```
import java.util.ArrayDeque;
class JavaCollection
 {
   public static void main(String args[])
{
 ArrayDeque<String> Q=new ArrayDeque<>();
 Q.add("Patna");
 Q.add("Delhi");
 System.out.println(Q);
 Q.remove(); //FIFO
 System.out.println(Q);
} }
```

```
Output:

[Patna, Delhi]

[Delhi]
```

✪ **Program for find sorting of a strin**

```
import java.util.Arrays;
import java.util.Scanner;
class StringSort
 {
   String stringSorting(String s)
    {
      int len=s.length();
      char sArr[]=s.toCharArray();
      for(int i=0; i<len; i++)
       {
         for(int j=i+1; j<len; j++)
          {
            if(sArr[i]>sArr[j])
             {
              char c=sArr[i];
              sArr[i]=sArr[j];
              sArr[j]=c;
             }
          } //Arrays.sort(sArr);
       }
      s=new String(sArr);
      return(s);
    }
   public static void main(String args[])
    {
      Scanner scan=new Scanner(System.in);
      System.out.print("Enter a string value=");
      String str=scan.nextLine();
      StringSort obj=new StringSort();
      System.out.print("Sorted string: "+obj.stringSorting(str));
    }
 }
```

```
Output:

Enter a string value=LKJHGFDSA

Sorted string: ADFGHJKLS
```

- **Program replace a character from a given string.**

import java.util.Arrays;

```
import java.util.Scanner;
class StringRep
 {
   String stringReplace(String s,char c1,char c2)
    {
      int len=s.length();
      for(int i=0; i<len; i++)
       {
         char v=s.charAt(i);
         if(v==c1)
          {
           s=s.replace(c1,c2);
          }
       }
      return(s);
    }
   public static void main(String args[])
    {
      Scanner scan=new Scanner(System.in);
      System.out.print("Enter a string value=");
      String str=scan.nextLine();
      System.out.print("Which char you want to replace=");
      char c1=scan.next().charAt(0);
      System.out.print("Enter the replace value=");
      char c2=scan.next().charAt(0);
      StringRep obj=new StringRep();
      System.out.print("Replaced string: "+obj.stringReplace(str,c1,c2));
    }
 }
```

```
Output:

Enter a string value=sdfghgffhgygf

Which char you want to replace=f

Enter the replace value=v

Replaced string: sdvghgvvhgygv
```

Variables:

- Variables are named locations in memory that hold values.
- They are declared using any datatype keyword (e.g., int, String, double).
- You can access the value of a variable using its name.
- Variables themselves do not have any behavior; they are essentially placeholders for data.

Objects:

- Objects are data structures that encapsulate data (attributes or fields) and methods into a single unit.
- They are declared using the class keyword, defining a blueprint for creating instances.

- You access the attributes and methods of an object using the object's name and the dot operator.
- Objects have behavior defined by the methods they contain. Methods allow objects to perform actions and interact with data.

- The Java programming language was originally named Oak but was later renamed Java after the Indonesian island of Java. The name Java was chosen because it was short, easy to remember,unique, unrelated to any existing technology, and had a pleasant sound.

Java is a general-purpose programming language used for developing a wide range of applications, while JavaScript is primarily used as a scripting language for enhancing interactivity on web pages.

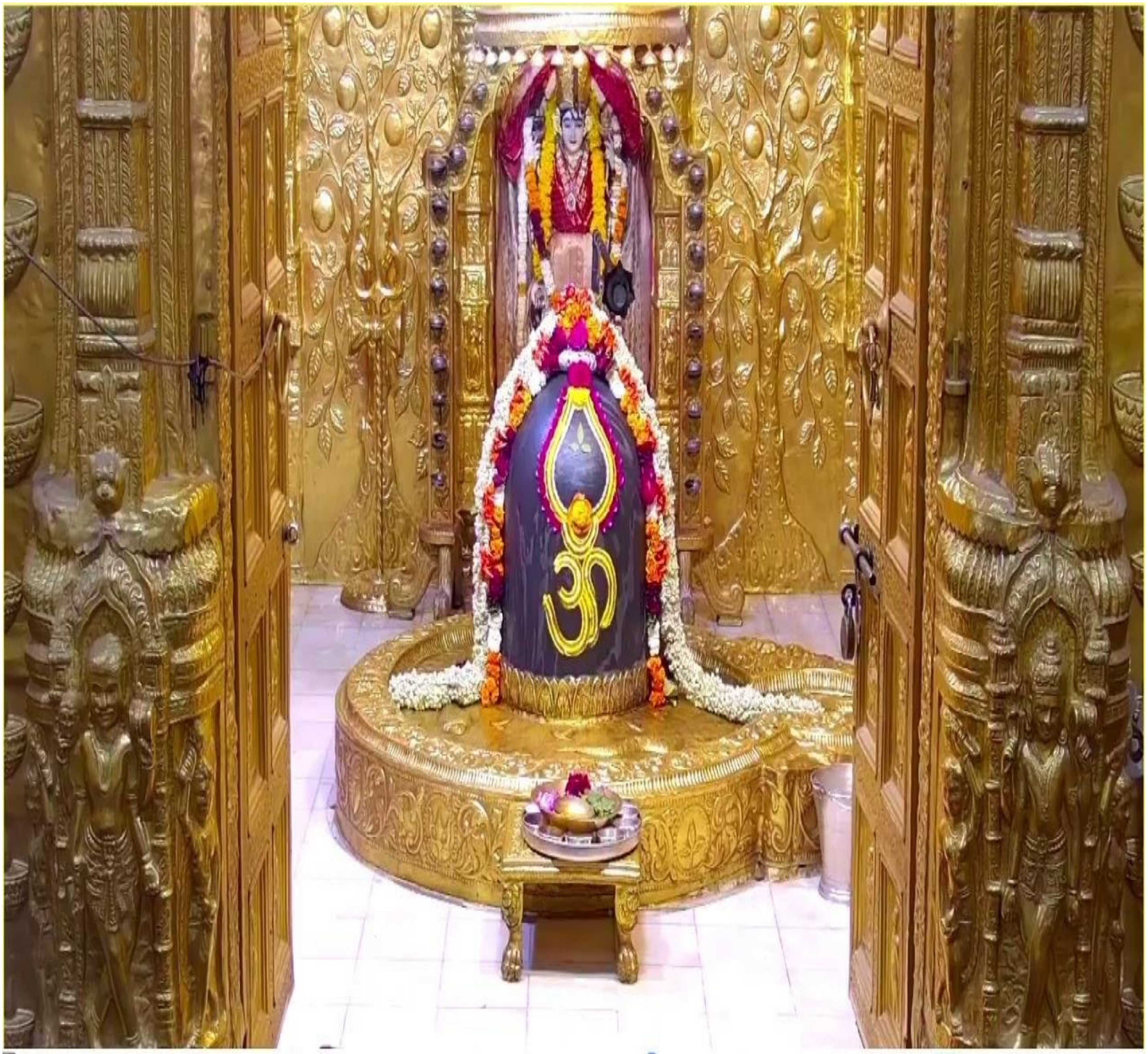
- **Note:** If we have declared a method or variable within the class but outside the main method, then outside the main method (within the class), we can use that method or variable without the help of an object. But if we want to use those methods and variables inside the main method, then we have to create the object of that class, and with the help of the object, we can use those methods and variables inside the main method.

- If we have declared variables inside the main method, then we can use those variables without the help of objects inside the main method. But if we want to use those variables (created within the main method) outside the main method (within the class inside the function), then we have to pass the variables through methods.

- If we have declared variables within the methods, then without object, we can access those variables within the method that are local variables for the methods.

- Java does not support “directly” nested methods. But Python supports nested methods.

॥ॐ नमः शिवाय॥



➨ **|| ॐ तत्पुरुषाय विद्महे, महादेवाय धीमहि, तन्नो रूद्र प्रचोदयात् ||**

➨

➨ ♨ஜ★ஜ♨