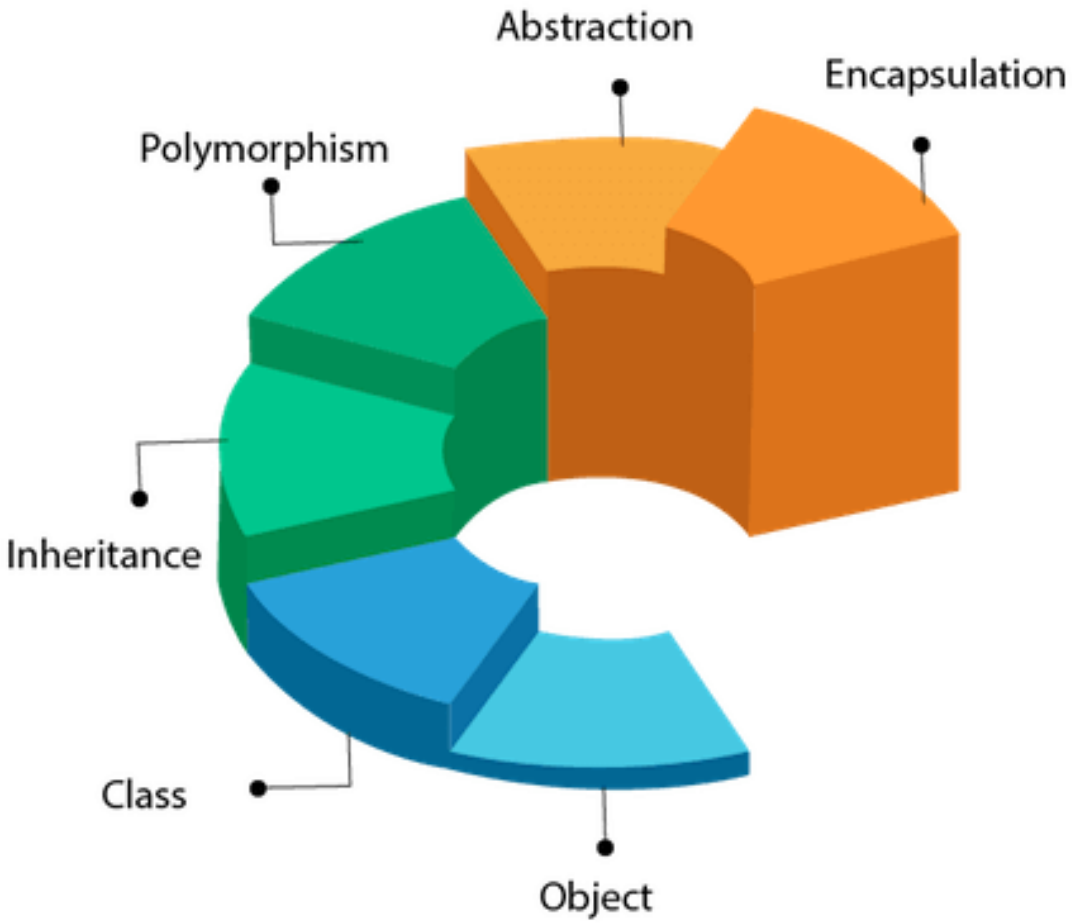
Java

Advance Programming

OOPs (Object-Oriented Programming System)



OOPs (Object-Oriented Programming System)

- **Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:
  - Object
  - Class
  - Inheritance
  - Polymorphism
  - Abstraction
  - Encapsulation
- Apart from these concepts, there are some other terms which are used in Object-Oriented design:
  - Coupling
  - Cohesion
  - Association
  - Aggregation
  - Composition

OOPs (Object-Oriented Programming System)

- Object :- Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
- Class :- *Collection of objects* is called class. It is a logical entity. A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.
- Inheritance :- *When one object acquires all the properties and behaviors of a parent object*, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.
- Polymorphism :- If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.
- In Java, we use method overloading and method overriding to achieve polymorphism.
- Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Cont....

- Abstraction :- *Hiding internal details and showing functionality* is known as abstraction. For example phone call, we don't know the internal processing. In Java, we use abstract class and interface to achieve abstraction.
- Encapsulation :- *Binding (or wrapping) code and data together into a single unit are known as encapsulation*. For example, a capsule, it is wrapped with different medicines. A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.
- Coupling :- It refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

Cont....

- Cohesion :- It refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.
- Association :-It represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:
  - One to One
  - One to Many
  - Many to One, and
  - Many to Many
  - Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).
  - Association can be undirectional or bidirectional.
- Aggregation :- It is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a *has-a* relationship in Java. Like, inheritance represents the *is-a* relationship. It is another way to reuse objects.
- Composition :- It is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

Brief Overview of

JAVA

What is java?

- **Java** is a computer programming language that is concurrent, class-based, object oriented.
- It is intended to let application developers “write once run any where" (WORA).
- Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture.
- Java is, as of 2014, one of the most popular programming languages in use, particularly for client-server web applications.
- Java is the Internet programming language.

Java's History

- Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation).
- Originally named OAK in 1991 First non commercial version in 1994
- Renamed and modified to Java in 1995 and released as a core component of Sun Microsystems' Java platform.
- First Commercial version in late 1995
  - Hot Java
  - The first Java-enabled Web browser

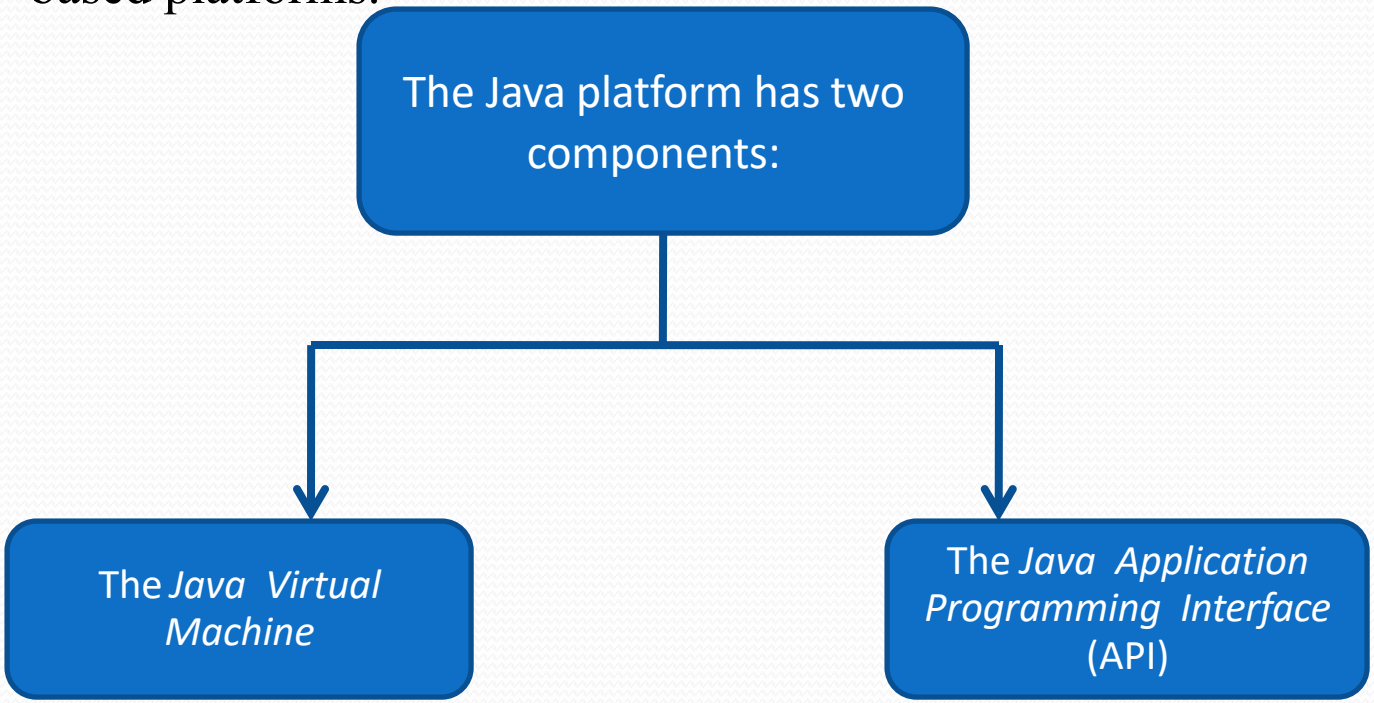
Java Technology

What is Java?

- Java technology is both a programming language and a platform.
- It is a part of Java programming language. It is an advanced technology or advance version of Java specially designed to develop web-based, network-centric or enterprise applications. It includes the concepts like Servlet , JSP, JDBC, RMI, Socket programming, etc.
- It is a specialization in specific domain.

The Java

- A platform is the hardware or software environment in which a program runs. Some of the most popular platforms are Microsoft Windows, Linux, Solaris OS, and Mac OS.
- Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware- based platforms.



Simple Java Program

```
Class demo
{
    public static void
    main(String s[])
    {

            System.out.println
    (“hellow java”);
    }
}
```

O/P

hellow java

```
Class add
{
    public static void main(String
    s[])
    {
            int a=5, b=8.5;
            float sum;
            sum= a+b;

            System.out.println(“Addi
    tion is :”+sum);
    }
}
```

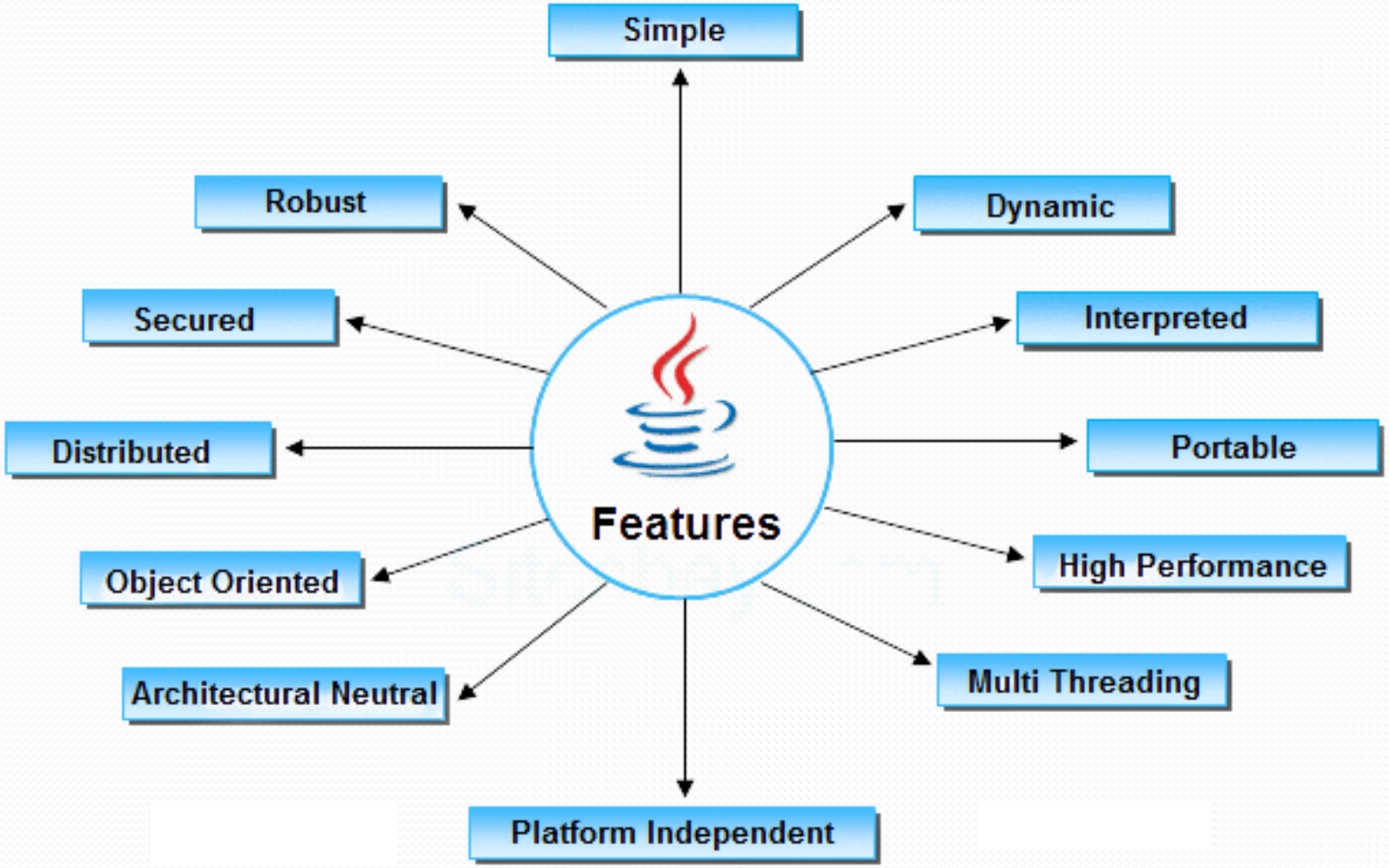
O/P

Addition is: 13

Java and Internet

- **Internet** users can use **Java** to create applet programs and run them locally using a "**Java**-enabled browser" such as HotJava. ... They can also use a **Java**-enabled browser to download an applet located on a computer anywhere in the **Internet** and run it on his local computer.

Features of java



- Simple: Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because: Java syntax is based on C++ (so easier for programmers to learn it after C++). Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.

- Object-oriented: Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

- Platform Independent: Java is platform independent because it is different from other languages like C, C++, etc.
  - Runtime Environment
  - API(Application Programming Interface)

  Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

- Secured: Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:
  - No explicit pointer
  - Java Programs run inside a virtual machine sandbox

- Architecture-neutral: Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed. In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

- Portable: Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

- High-performance: Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

- Robust: Robust simply means strong. Java is robust because: It uses strong memory management. There is a lack of pointers that avoids security problems. There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore. There are exception handling and the type checking mechanism in Java. All these points make Java robust.

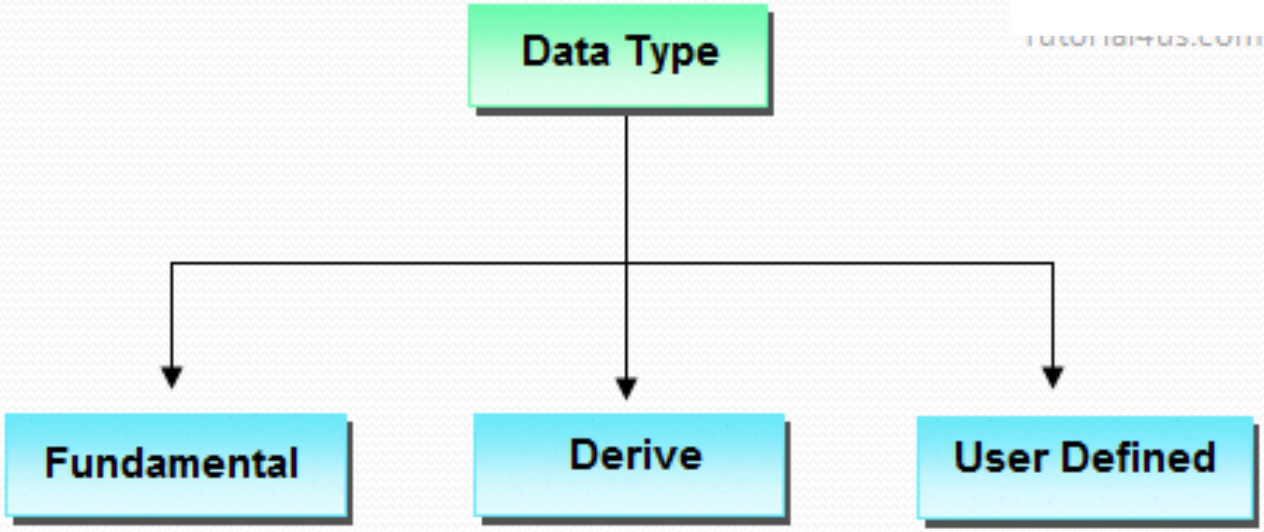
- Distributed: Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

- Multi-threaded: A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

- Dynamic: Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.Java supports dynamic compilation and automatic memory management (garbage collection)

Data types

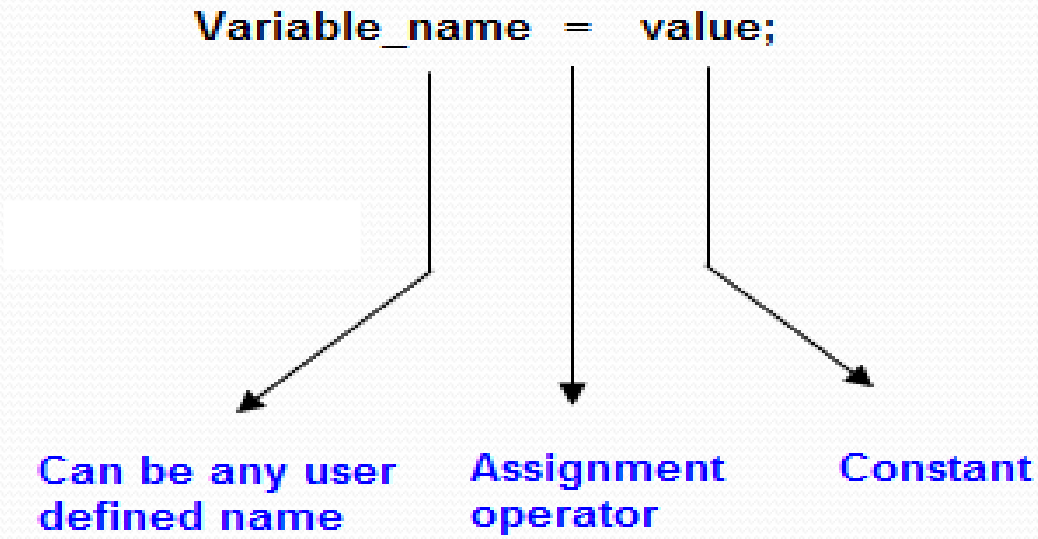
- Datatype is a special keyword used to allocate sufficient memory space for the data, in other words Data type is used for representing the data in main memory (RAM) of the computer.
- In general every programming language is containing three categories of data types. They are
  - Fundamental or primitive data types
  - Derived data types
  - User defined data types.



- Primitive data types
  - Primitive data types are those whose variables allows us to store only one value but they never allows us to store multiple values of same type. This is a data type whose variable can hold maximum one value at a time.
    - Example int a; // valid
    - a=10; // valid
    - a=10, 20, 30; // invalid
- Derived data types
  - Derived data types are those whose variables allow us to store multiple values of same type. But they never allows to store multiple values of different types. These are the data type whose variable can hold more than one value of similar type. In general derived data type can be achieve using array.
    - Example int a[] = {10,20,30}; // valid
    - int b[] = {100, 'A', "ABC"}; // invalid
- User defined data types
  - User defined data types are those which are developed by programmers by making use of appropriate features of the language.
    - Example Student s = new Student();

Variables and Arrays

- Variable is an identifier which holds data or another one variable is an identifier whose value can be changed at the execution time of program. Variable is an identifier which can be used to identify input data in a program.
  - **Syntax**
    - Variable_name = value;



- **Array** is a collection of similar type of data. It is fixed in size means that you can't increase the size of array at run time. It is a collection of homogeneous data elements. It stores the value on the basis of the index value.
  - There are two types of array in Java.
    - Single Dimensional Array
    - Multidimensional Array

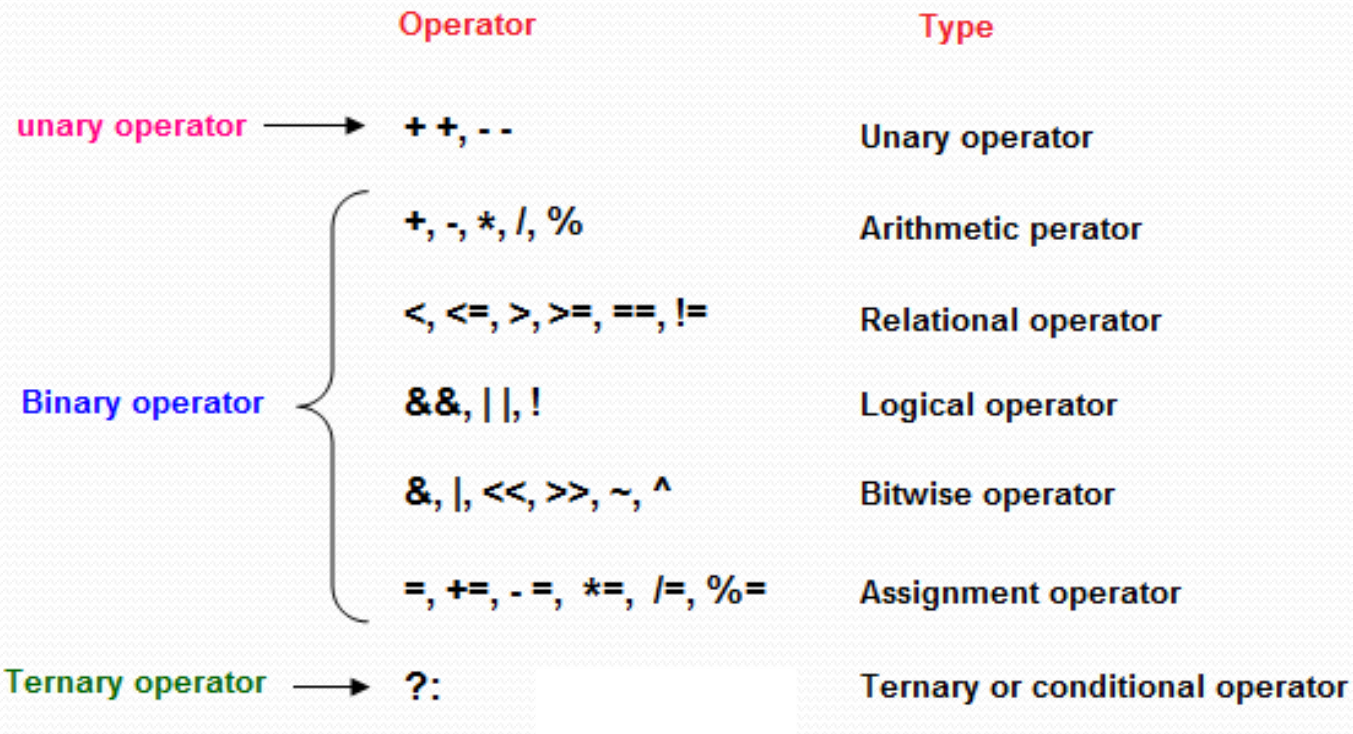
```
public class ArrayExample
{
    public static void main(String []args)
    {
            int arr[] = {10,20,30};
           for (int i=0; i < arr.length; i++)
            {
                    System.out.println(arr[i]);
           }
    }
}
```

- **Output**

```
10
 20
 30
```

Operators

- **Operator** is a special symbol that tells the compiler to perform specific mathematical or logical Operation. Java supports following lists of operators.
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Bitwise Operators
  - Assignment Operators
  - Ternary or Conditional Operators



Classes in Java

- A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

```
class Student{
 int id;
 String name;
}
class TestStudent3{
 public static void main(String args[]){
  //Creating objects
  Student s1=new Student();
  Student s2=new Student();
  //Initializing objects
  s1.id=101;
  s1.name="ABC";
  s2.id=102;
  s2.name="Amit";
  //Printing data
  System.out.println(s1.id+" "+s1.name);
  System.out.println(s2.id+" "+s2.name);
 }
}
```

Output:
101 Sonoo
102 Amit

Constructor in Java

- In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.

- **Rules for creating Java constructor**

  There are two rules defined for the constructor.
  - Constructor name must be the same as its class name
  - A Constructor must have no explicit return type
  - A Java constructor cannot be abstract, static, final, and synchronized

- **Types of Java constructors**

  There are two types of constructors in Java:
  - Default constructor (no-arg constructor)
  - Parameterized constructor

Examples

Default Constructor

```
class Bike1{
Bike1()
{
System.out.println("Bike is created");
}
public static void main(String args[])
{
Bike1 b=new Bike1();
}
}
```

Output:

Bike is created

Parameterized Constructor

```
class Student4{
  int id;
   String name;
   Student4(int i,String n){
   id = i;
   name = n;
  }
 void display(){
System.out.println(id+" "+name);
}
 public static void main(String args[]){
    Student4 s1 = new Student4(111,"Karan");
    //Student4 s2 = new Student4(222,"Aryan");
    s1.display();
   // s2.display();
   }
}
```

Output:

111 Karan

222 Aryan

Difference between Method and Constructor

	Method	Constructor
1	Method can be any user defined name	Constructor must be class name
2	Method should have return type	It should not have any return type (even void)
3	Method should be called explicitly either with object reference or class reference	It will be called automatically whenever object is created
4	Method is not provided by compiler in any case.	The java compiler provides a default constructor if we do not have any constructor.

Constructor Overloading

- In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

```
class Stud5
{
   int id;
   String name;
   int age;
   Stud5(int i, String n)
   {
      id = i;
      name = n;
    }
   Stud5(int i, String n, int a)
   {
   id = i;
   name = n;
   age=a;
   }
   void display()
   {
   System.out.println(id+" "+name+" "+age);
   }
```

```
 public static void main(String args[])
{
   Stud5 s1 = new Stud5(111,"Karan");
   Stud5 s2 = new Stud5(222,"Aryan",25);
   s1.display();
   s2.display();
  }
}
```

Output:

0111 Karan
222 Aryan 25

Garbage Collection

- In java, garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
- To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.
- Advantage of Garbage Collection
  - It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
  - It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

- How can an object be unreferenced?
  There are many ways:

1) By nulling a reference:

```
Employee e=new Employee();
e=null;
```

2) By assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by e1 is available for garbage collection
```

3) By anonymous object:

```
new Employee();
```

- finalize() method
  - The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing.
  - This method is defined in Object class as:

    protected void finalize(){}

    gc() method
  - The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

    public static void gc(){}

Example of garbage collection

```
public class TestGarbage
  {
     public void finalize()
      {
        System.out.println("object is garbage collected");
       }
    public static void main(String args[])
    {
       TestGarbage s1=new TestGarbage();
       TestGarbage s2=new TestGarbage();
        s1=null;
        s2=null;
        System.gc();
    }
}
```

Output:

```
object is garbage collected
object is garbage collected
```

Inheritance

- The process of obtaining the data members and methods from one class to another class is known as **inheritance**. It is one of the fundamental features of object-oriented programming.

- **Important points**
  - In the inheritance the class which is give data members and methods is known as base or super or parent class.
  - The class which is taking the data members and methods is known as sub or derived or child class.
  - The data members and methods of a class are known as features.
  - The concept of inheritance is also known as re-usability or extendable classes or sub classing or derivation.


Son i am
Base class
Inheritance
Dad i am
Derive class
Sitesbay.com

- Terms used in Inheritance
  - **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
  - **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
  - **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
  - **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

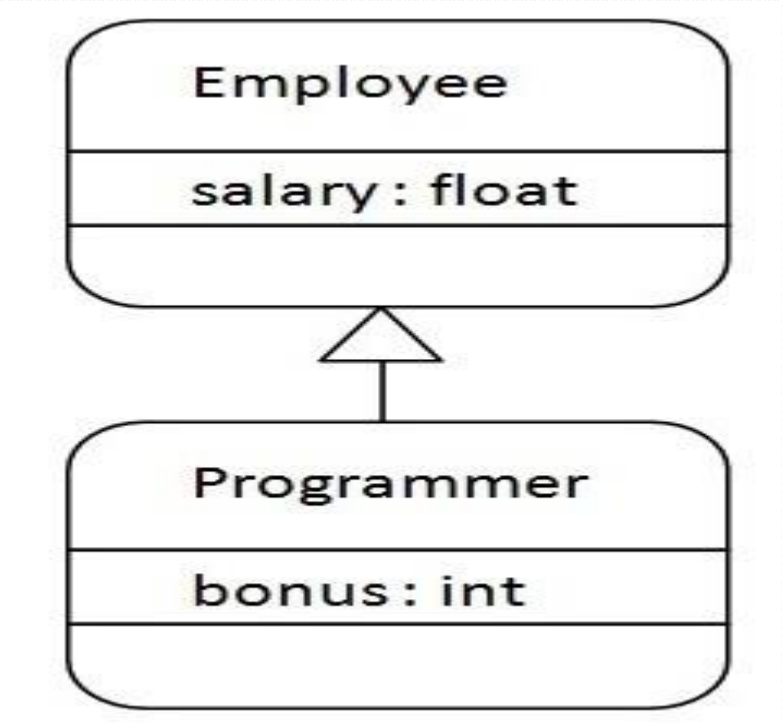
  The syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

  - The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.
  - In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

- **Why use Inheritance ?**
  - For Method Overriding (used for Runtime Polymorphism).
  - It's main uses are to enable polymorphism and to be able to reuse code for different classes by putting it in a common super class
  - For code Re-usability

Java Inheritance Example



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. It means that Programmer is a type of Employee.

Example

```
class Employee
 {
   float salary=40000;
 }

class Programmer extends Employee
{
      float bonus=10000;
      public static void main(String args[])
     {
        Programmer p=new Programmer();
        System.out.println("Programmer salary is:"+p.salary);
        System.out.println("Bonus of Programmer is:"+p.bonus);
      }
}
```

Output:

Programmer salary is:40000.0

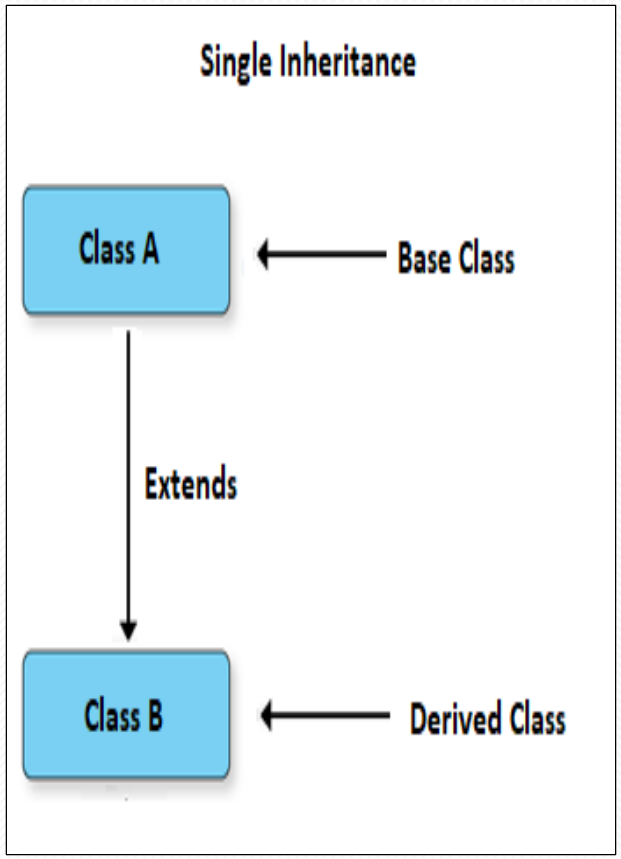
Bonus of programmer is:10000

- **Advantage of inheritance**
  - If we develop any application using concept of Inheritance than that application have following advantages,
  - Application development time is less.
  - Application take less memory.
  - Application execution time is less.
  - Application performance is enhance (improved).
  - Redundancy (repetition) of the code is reduced or minimized so that we get consistence results and less storage cost.

- **Types of Inheritance**

  Based on number of ways inheriting the feature of base class into derived class we have five types of inheritance; they are:
  - Single inheritance
  - Multiple inheritance
  - Hierarchical inheritance
  - Multilevel inheritance
  - Hybrid inheritance

- **Single inheritance**
  - In single inheritance there exists single base class and single derived class.



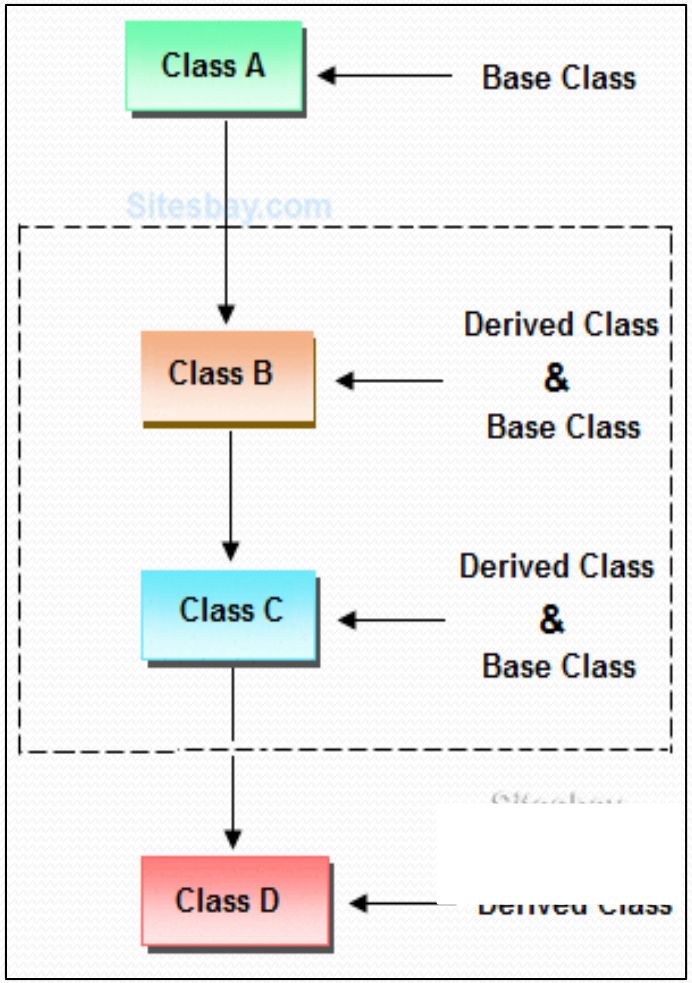

```
class Faculty
{
   float salary=30000;
}
class Science extends Faculty
  {
     float bonous=2000;
      public static void main(String args[])
      {
           Science obj=new Science();
           System.out.println("Salary is:"+obj.salary);
           System.out.println("Bonous is:"+obj.bonous);
       }
  }
```

Output:

Salary is: 30000.0

Bonous is: 2000.0

- **Multilevel inheritances in Java**
  - In Multilevel inheritances there exists single base class, single derived class and multiple intermediate base classes.
  - **Single base class + single derived class + multiple intermediate base classes.**



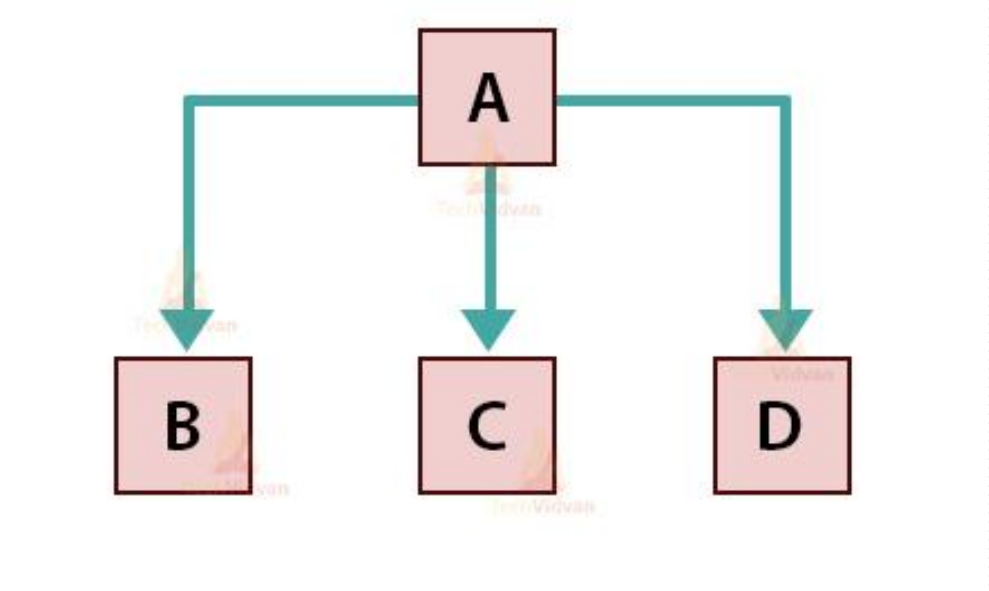
```
class Animal{
void eat(){
    System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){
    System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){
    System.out.println("weeping...");}
}
class TestInheritance{
public static void main(String args[])
{
    BabyDog d = new BabyDog();
    d.weep();
    d.bark();
    d.eat();
    }
}
```

Output:

weeping...
barking...
eating...

Hierarchical Inheritance

- When more than one classes inherit a same class then this is called hierarchical inheritance. For example class B, C and D extends a same class A. when a class has more than one child classes (sub classes) or in other words more than one child classes have the same parent class then this **type of inheritance** is known as **hierarchical inheritance**.



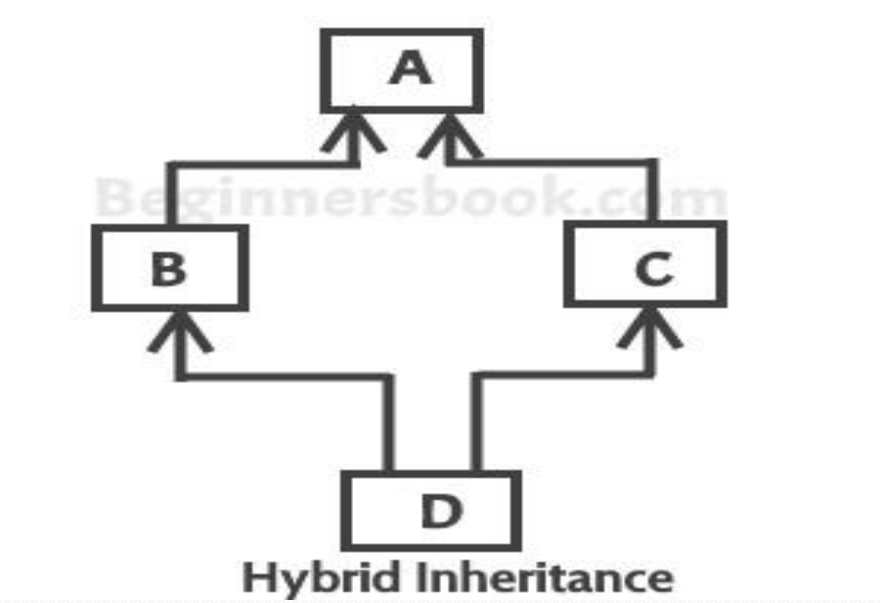
```
class Animal{
    void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
    void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
    void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
Dog d = new Dog();
d.bark();
d.eat();
}
}
```

Output:

```
meowing...
eating...
```

Hybrid Inheritance

- A hybrid inheritance is a combination of more than one **types of inheritance**. For example when class A and B extends class C & another class D extends class A then this is a hybrid inheritance, because it is a combination of single and hierarchical inheritance.



- Why 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.
  - Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

Interfaces

- **Interface** is similar to class which is collection of public static final variables (constants) and abstract methods.
- The interface is a mechanism to achieve fully abstraction in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.
- **Why we use Interface ?**
  - It is used to achieve fully abstraction.
  - By using Interface, you can achieve multiple inheritance in java.
  - It can be used to achieve loose coupling.

- **properties of Interface**
  - It is implicitly abstract. So we no need to use the abstract keyword when declaring an interface.
  - Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
  - Methods in an interface are implicitly public.
  - All the data members of interface are implicitly public static final.
- **How interface is similar to class ?**
  - Whenever we compile any Interface program it generate .class file. That means the bytecode of an interface appears in a .class file.

- **How interface is different from class ?**
  - You can not instantiate an interface.
  - It does not contain any constructors.
  - All methods in an interface are abstract.
  - Interface can not contain instance fields. Interface only contains public static final variables.
  - Interface is can not extended by a class; it is implemented by a class.
  - Interface can extend multiple interfaces. It means interface support multiple inheritance

```
interface Person
{
void run();
}
class Employee implements Person
 {
public void run()
 {
System.out.println("Run fast");
}
}
```

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Output:
Hello

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 }
}
```

Output:
Hello
Welcome

```
interface Printable{
void print();
}
interface Showable{
void print();
}

class TestInterface3 implements Printable, Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestInterface3 obj = new TestInterface3();
obj.print();
 }
}
```

Output:
Hello

```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank{
public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
public float rateOfInterest(){return 9.7f;}
}
class TestInterface2{
public static void main(String[] args){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());
}}
```

Output:
ROI: 9.15

Sr. No	Abstract class	Interface
1	It is collection of abstract method and concrete methods.	It is collection of abstract method.
2	There properties can be reused commonly in a specific application.	There properties commonly usable in any application of java environment.
3	It does not support multiple inheritance.	It support multiple inheritance.
4	Abstract class is preceded by abstract keyword.	It is preceded by Interface keyword.
5	Which may contain either variable or constants.	Which should contains only constants.
6	The default access specifier of abstract class methods are default.	There default access specifier of interface method are public.
7	These class properties can be reused in other class using extend keyword.	These properties can be reused in any other class using implements keyword.
8	Inside abstract class we can take constructor.	Inside interface we can not take any constructor.
9	For the abstract class there is no restriction like initialization of variable at the time of variable declaration.	For the interface it should be compulsory to initialization of variable at the time of variable declaration.
10	There are no any restriction for abstract class variable.	For the interface variable can not declare variable as private, protected, transient, volatile.

Exception handling

- The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.
- What is Exception in Java
  - **Dictionary Meaning:** Exception is an abnormal condition.
  - In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.
- What is Exception Handling
  - Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Types of Java Exceptions

- There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle

1) Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

```
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
      //code that may raise exception
      int data=100/0;
         }
     catch(ArithmeticException e){
       System.out.println(e);}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

Output:

java.lang.ArithmeticException:/ by zero

rest of the code...

Finally block

- **Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc.
- Java finally block is always executed whether exception is handled or not.
- Java finally block follows try or catch block.
- Why use java finally
- Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){
     System.out.println(e);
}
  finally{
     System.out.println("finally block is always executed");
     }
  System.out.println("rest of the code...");
  }
}
```

OUTPUT

```
5
finally block is always executed
```

Multithreading

- **Multithreading in Java** is a process of executing multiple threads simultaneously.
- A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.
- However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.
- Advantages of Java Multithreading
  1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.
  2) You **can perform many operations together, so it saves time**.
  3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Thread class

- Threads allows a program to operate more efficiently by doing multiple things at the same time.
- Threads can be used to perform complicated tasks in the background without interrupting the main program.
- Creating a Thread
  - There are two ways to create a thread.
    - By extending Thread class
    - By implementing Runnable interface.

- Extend Syntax

```
public class Main extends Thread
{
   public void run() {
   System.out.println("This code is running in a thread");
   }
}
```

- Constructors of Thread class:

1. **Thread():** This allocates a new Thread object.
2. **Thread(Runnable target):** This allocates a new Thread object.
3. **Thread(Runnable target, String name):** This allocates a new Thread object.
4. **Thread(String name):** This constructs allocates a new Thread object.
5. **Thread(ThreadGroup group, Runnable target):** This allocates a new Thread object.
6. **Thread(ThreadGroup group, Runnable target, String name):** This allocates a new Thread object so that it has target as its run object, has the specified name as its name, and belongs to the thread group referred to by group.
7. **Thread(ThreadGroup group, Runnable target, String name, long stackSize):** This allocates a new Thread object so that it has target as its run object, has the specified name as its name, belongs to the thread group referred to by group, and has the specified stack size.
8. **Thread(ThreadGroup group, String name):** This allocates a new Thread object.

Methods of Thread class

- **run()**
  - Which contains the main business logic that can be executed by multiple threads simultaneously in every user defined thread class run method should be overridden.

```
public Class_Name extends Thread {
public void run() {
..... .....
}
}
```

- **start()**
  - Used to convert ready state thread to running state.

```
Thread t=new Thread();
t.start();
```

- **stop()**
  - This method is used to convert running state thread to dead state.

```
Thread t=new Thread();
t.stop();
```

```
class Demo extends Thread{
public void run(){
System.out.println("thread is running...");
-----
------
}
public static void main(String args[]){
Demo  t1=new  Demo();
t1.start();
t1.stop();
}
}
```

Output:

thread is running...

Runnable Interface

- Runnable is one of the predefined interface in java.lang package, which is containing only one method and whose prototype is " Public abstract void run "
- The run() method of thread class defined with null body and run() method of Runnable interface belongs to abstract. Industry is highly recommended to override abstract run() method of Runnable interface but not recommended to override null body run() method of thread class.

```
Class Demo implements Runnable{
public void run()
{
System.out.println("thread is running...");
}

public static void main(String args[]){
Demo  m1=new  Demo();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

Output

thread is running...

User Define Exception

- If any exception is design by the user known as user defined or Custom Exception. Custom Exception is created by user.
- **Rules to design user defined Exception**
  - Create a package with valid user defined name.
  - Create any user defined class.
  - Make that user defined class as derived class of Exception or RuntimeException class.
  - Declare parametrized constructor with string variable.
  - call super class constructor by passing string variable within the derived class constructor.
  - Save the program with public class name.java

```
class UserDefine extends Exception
{
      UserDefine(String  s)
      {
                  super(s);
      }
}
Class Input
{
void method() throws UserDefine
      {
      throw new UserDefine ("Entered input is incorrect"):
      }
}
public class User
{
      public static void main(String[] args)
      {
                  try{
                  new Input().method();
                  }
                  catch(UserDefine  w)
                  {
                  System.out.println(w.getMessage());
                  }
      }
}
```

OUTPUT:
Entered input is incorrect

Inter-thread communication

- **Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.
- Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of **Object class**:
- wait()
  - Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object

- notify()
  - Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

    public final void notify()
- notifyAll()
  - Wakes up all threads that are waiting on this object's monitor.

  public final void notifyAll()

```
class Pen{}
class Paper{}

public class Write {

 public static void main(String[] args)
 {
   final Pen pn =new Pen();
   final Paper pr =new Paper();

   Thread t1 = new Thread() {
     public void run()
     {
        synchronized(pn)
        {
           System.out.println("Thread1 is holding Pen");
           try{
              Thread.sleep(1000);
           }
           catch(InterruptedException e){
             // do something
           }
           synchronized(pr)
           {
             System.out.println("Requesting for Paper");
           }
        }
     }
    };
```

```
Thread t2 = new Thread() {
        public void run()
        {
           synchronized(pr)
           {
              System.out.println("Thread2 is holding Paper");
              try {
                 Thread.sleep(1000);
              }
              catch(InterruptedException e){
                 // do something
              }
              synchronized(pn)
              {
                 System.out.println("requesting for Pen");
              }
           }
        }
     };

     t1.start();
     t2.start();
  }
}
```

OUTPUT:

Thread1 is holding Pen
Thread2 is holding Paper

Thank You