INTRODUCTION TO JAVA , BASICS OF OOPs

Java programming language was originally developed by Sun Microsystems

- By James Gosling
- Released in 1995 as core component of Sun Microsystems

Object Oriented

- In java everything is an object
- Java has been one of the most popular programming languages for many years. Java is **Object Oriented**.

Platform Independent

- Java platform independent means ' write once and run anywhere' or WORA. It implies that it doesn't matter on what operating system (let's say, Windows OS) the code was written, it could be run on the other operating system (let's say Linux) conveniently and without any issue.

Multi-Threaded

- Java is a **multi-threaded programming language** which means we can develop multi-threaded program using Java.
- A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

Interpreted

- Java and the JVM were designed with portability in mind. Therefore, most popular platforms today can run Java code.
- This might sound like a hint that **Java is a purely interpreted language**. However, before execution, Java source code needs to be compiled into bytecode. Bytecode is a special machine language native to the JVM.

JAVA PROGRAM LIFECYCLE

It has four phases

- **EDIT**
- **COMPILE**
- **LOAD**
- **EXECUTE**
- Life cycle of a java program tells us what happens right from the point when we type source code in a text editor to the point that source code is converted into machine code (0's and 1's). There are four main stages in the **life cycle** of a **java program.** They are: Editing the program; Compiling the source Code; Class loader stores byte code in memory; Executing the byte code.

DATA TYPES IN JAVA

- Java language has a rich implementation of data types.
- Data types specify the different sizes and values that can be stored in values.
- In java data types are classified int two types.
    1. Primitive Data type
    2. Non-Primitive Data type

INTEGER

- This group includes byte,short,int,long.
    - **Byte:**
        - It is 1 byte(8 bits) integer data type.
        - Value range from –128 to 127.
        - Default value zero.
    - **Short:**
        - It is a 2 bytes(16 bits)intger data types.
        - Value range from –32768 to 32767.
        - Default value zero.
    - **Int:**
        - It is 4 bytes(32 bits) integer data types.

    - Value range from INTEGER.MIN_VALUE to INTEGER.MAX_VALUE.
    - Default value zero.
- **Long:**
    - It is 8 bytes(64 bits) integer data types.
    - Value range from LONG.MIN_VALUE to LONG.MAX_VALUE.

Floating-Point Number

- Floating type is useful to store decimal poinr values.
- This group includes float,double.
    - **FLOAT:**
        - It is a 4 bytes(32 bits).
        - Default value 0.0f.
    - **DOUBLE**
        - It is a 8 bytes(64 bits).
        - Default value 0.0d

CHARACTERS

- This group represent char, which represent symbols in a character set,like letters and numbers.
    - **Char:**
        - It is 2bytes(16 bits)
        - Minimum value is \u0000(or 0)
        - Maximum value is \uffff(or 65,535 inclusive)
        - Char data type is used to store any character.

BOOLEAN

- This group represent boolean.
- They are defined constant of the language.
    - **boolean**
        - The boolean data type has two possible values, either true or false.
        - Default value : false.

OBJECT ORIENTED PROGRAMMING

- **Object-oriented Programming** or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc. In programming, the main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

CONCEPTS OF OOPs

- **Encapsulation**
- **Abstraction**
- **Polymorphism**
- **Inheritance**

METHOD OVERLOADING IN JAVA

- It is a concept that allows to declare multiple methods with same name but different parameters in same class.
- It always occur in same class(unlike overriding)
- Method overloading is one of the ways through which java supports polymorphisim.
- Method overloading can be done by changing numbers of arguments or by changing the data type cannot be overloaded
- There are two types
    - Different datatype of arguments
    - Different number of arguments

Java objects:

- Object is an instance of a class while class is the blueprint of an object.
- An object represents the class and consists of properties and behaviour
- Properties refer to the field declared with in class and behaviour represents to the methods available in the class.
    - **Creating objects:**
        - Object in java is essentially a block of memory that contains space to store all the instance variables.
        - Creating an object is also known as instantiating an object.
        - New--> creates an object of the specified class and returns a reference to that object

Java access modifiers:

- In java access modifiers are used to set the accessbility(visisbility) of classes, interfaces,variables,methods,constructors,data members and the setter methods.
- For example,
    - Class Animal{

    public void method1(){…}

    private void method2(){…}

    }

        - method1 is public – this means accessed by the othr classes.
        - method2 is private – this means it cannot be accessed by other classes.

Object Intialization in java:

- Intializing an object means storing data into the object,
- There are 3 ways to intialize object in java
    - By reference variable
    - By method
    - By constructor

Java Constructor:

- A constructor is called a special method that is used to initialize an object.
- Every class has a constructor either implicitly or explicitly.
- A constructor same name has the class name which it is declared.
- Constructor in java cannot be abstract,static,final or synchronized.These modifiers are not followed for constructor.
- **SYNTAX**
  - ClassName(Parameter-list){

  code-statements;

  }
- There are two types of constructor
  - Default constructor
  - Parameterized constructor
- Parameter-list, is optional because constructors can be parameterize and non-parameterize as well.

ENCAPSULATION

- It is one of the more important OOP concepts.
- **Encapsulation in java** is a mechanism to wrap up variables(data) and methods (code)together as a single unit.
- To prevent direct modifications of data fields, we should declare the data field private using private modifier.
- A private data cannot be accessed by an object from outside the class that defines the private field.
- To enable a private data field to update provide a set method to set a new value.
- get method is method as getter(accessor)
- set method is referred to as setter(mutator)

Get method ()

- public returnType getProperty()

If return type is boolean:

- public returnType isPropertyName()

Set Method ()

- Public void setPropertyName(data type property value)

Example program

```
Public class circle {
Private int circle;
    circle () {
radius =1;
    }
    circle (int r) {
radius = r;
    }

Private in getRadius(){
     Return radius;
}
Public void setRadius(){
   Radius= r;
}

public class RadiusTest(){
   Public static void main(String args[]){
Circle c1 = new circle(5);
System.out.println(“radius = “ +c1.getRadius());
```

```
Circle c2 = new Circle(10);
System.out.println("radius = "+c2.getRadius());
    }
}
```

OUTPUT

radius = 5

radius = 10

CONCLUSION :

- We use the getRadius accessor to return the value from the private field
- If we tried to invoke it by(c1.radius) java would throw an error.

INHERITANCE

- It is one of the key features of Object Oriented Programming.
- It provide a mechanism that allow a class to inherit property of another class.
- The idea behind the inheritance in java is that we can create new classes that are built upon existing classes.
- When a new class extends another class it inherits all non-private members including fields and methods.
- Inheritance in java can be understood in terms of parent and child relationship, also know as super class(parent) and sub class(child) in java language.
- Inheritance defines a **is-a** relationship between a super class and its sub-class.
- When we inherit from an existing class, we can reuse methods and fields of the parent class. Moreover, we can add new methods and fields in our current class also.

SYNTAX

Class Subclass-name extends Superclass-name{

//methods and fields

}

- The extends keyword indicates that we are making new class that derives from an derived class.
- **Sub class/Child Class –** Subclass is a class which inherits the other class.It is also called as derived class,extended class, or child class.
- **Super class/Parent Class –** Superclass is the class from a where a subclass inherits the feature. It is also called a base class or a parent class.

Example

```
class Base{
int I;
void method1(){
System.out.println(“method 1”);
}
}
class Sub extends Base{
float f;
void method2(){
```

```
System.out.println("method 2");
                }
}
class InheritTest{
    public static void main(String args[]){
                    Sub obj = new Sub();
                    System.out.println("obj.i"=+obj.i);
                    System.out.println("obj.f"=+obj.f);
                    Obj.method1();
                    Obj.method2();
                }
            }
```

OUTPUT

Obj.i = 0

Obj.f =0.0

method 1

method 2

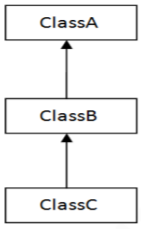
TYPES OF INHERITANCES IN JAVA

- Single
- Multilevel
- Hierarchical

- Multiple(does not support directly however implemented using secondary inheritance path in the form of interfaces)

```
class A
{
   int a = 10;
   void showA()
   {
      System.out.println("a = "+a);
   }
}
class B extends A
{
   int b = 10;
   void showB()
   {
      System.out.println("b = "+b);
   }
}
public class C extends B
{
   public static void main(String[] args)
   {
   C c = new C();
   c.showA();
   c.showB();
   }
}
```

Multilevel Inheritance Example



2) Multilevel

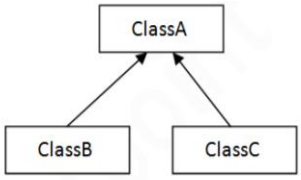
```
Command Prompt
G:\JAVA_PGMS>javac C.java

G:\JAVA_PGMS>java C
a = 10
b = 10
```



```
class A
{
   int a = 10;
   void showA()
   {
      System.out.println("a = "+a);
   }
}
class B extends A
{
   int b = 20;
   void showB()
   {
      System.out.println("b = "+b);
   }
}
class C extends A
{
   int c = 30;
   void showC()
   {
      System.out.println("c= "+c);
      showA();
   }
}
```

Hierarchical Inheritance Example



3) Hierarchical

```
public class HierarchicalTest
{
   public static void main(String[] args)
   {
      B b = new B();
      b.showA();
      b.showB();
       C c = new C();
      c.showC();
      c.showA();
   }
}
```

```
Command Prompt
G:\JAVA_PGMS>javac HierarchicalTest.java

G:\JAVA_PGMS>java HierarchicalTest
a = 10
b = 20
c= 30
a = 10
```


CONSTRUCTORS

- In java, constructor of base class with no arguments gets automatically called in derived class constructor.
- If we want to call parameterized constructor of base class, then we can call it using super()

- The point to note is base class constructor call mus be the first line in derived class constructor.

EXAMPLE

```
Class Base{
Base(){
System.out.println("base class constructor is called");
        }
    }
    Class Derived extends Base{
        Derived(){
            System.out.println("Derived class is called");
        }
    }
    public class Tester{
        Public static void main(String args[]){
            Derived d = new Derived();
        }
    }
```

OUTPUT :

```
base class constructor is called
Derived class is called
```

METHOD OVERRIDING

- When we declare the same method in child class (sub class) which is already present in the parent class(super class), then the method of the subclass class overides the method of method the superclass . This is know as **method overriding**
- In this case when we call the method from child class object,the child class version of the method is called.
- However we can call the parent class method using super keyword
- Method overriding performs only if two classes have is-a relationship.it mean class must have inheritance.
- Method overriding is also referred to as runtimre polymorphism because caling method is decided by JVM during run time.
- In overriding,methods of both class must have same name and equal number of parameters.
- We cannot override static method.Because static is bound to class whereas method overriding is associated with object.

POLYMORPHISM

RUNTIME POLYMORPHISM

- It is a process in which a call to an ovrridden method is resolved at runtime rather than compile time.
- In this process , an overridden method is called through the reference variable of superclass
- The determination of the method to be called is based on the object being referred to by the reference variable.
- When parent class reference variable refers to child class object it is know as upcasting.
- In those cases we can create a parent class refrence and assign child classes objects to it.

EXAMPLE

```
class Animal{
void eat(){
System.out.println(“Animal eats food”);
        }
    }
    Class Dog extends Animal{
    Void eat(){
    System.out.println(“dogs drink milk”);
        }
    }
    Class BabyDog extends Dog{
    Void move(){
    System.out.println(“BabyDogs can move”);
        }
    }
    Class RunTime Polymorphism{
    Public static void(String args[]){
     Animal a = new Animal();
    A.eat();
        }
    }
```

OUTPUT

Dogs drink milk

ABSTRACTION

- Abstraction is basically the art of hiding implementation details from user and provide the user what they want,
- Most of us are quite fond of owning a car. When we go to place order for the car we are not really interested into understand very fine details.
- We leave those technical details and implementation for manufacturing engineers and mechanics to understand.

EXAMPLE

```
Class Test{
void add(int a ,int b ){
int z = a+b;
System.out.println(z);
        }
    }
    Public class Tester{
    Public static void main(String args[]){
    Test sum = new Test();
            System.out.println("enter two numbers");
    Scanner s = new Scanner(System.in);
    Scanner s1 = new Scanner(System.in);
    int a = s.nextInt();
            int b = s1 . NextInt();
    sum.add(a,b);
        }
    }
```

OUTPUT

```
enter two numbers
```

9

6

15

CONCLUSION

- The user only consent about the add operation so we are abstracting the method.