UNIT 1 JAVA

FUNDAMENTALS

✩ History of Java

Java is general purpose, object oriented programming language.

Developed by Sun Microsystems of USA in 1991 AD by James Gosling.

In 1990 - Sun Microsystems of decided to develop special software for consumer electronic devices.

In 1991 - Using C++ the team announced a new language named oak

In 1992 - The Green project team by Sun, show new language to control home applionces using hand-held devices with tiny touch-sensative screen.

In 1993 - The WWW appeared on the internet and transformed the text based internet into graphics rich environment using web Applets.

In 1994 - The team developed a web browser called HOT Java for web Applets.

In 1995 - "Oak" was renamed "Java". Netscape and microsoft support Java.

In 1996 - Java established itself as a leader for internet and general purpose programming.

☆ Features of Java

1. Platform independent

→ Java programs use the java virtual machine as abstraction and do not access the operating system directly. This makes java programs highly portable. A Java programs can run unmodified on all supported platform.
eg:- Windows or Linux.

2. Object oriented programming language

→ Except the primitive data types, all elements in Java are object.

3. Strongly - Typed programming language

→ Java is strongly-typed language. Eg, the types of the used variables must be pre-defined and conversion to other object is relatively strict.

4. Interpreted and compiled language

→ Usually a computer language is either compiled

or interpreted. Java combines both these approaches thus making a two-stage system. First Java compiler translates source code into the byte code format which does not dependend on the target platform. These byte code (Intermediate code) will be interpreted by the Java virtual Machine (JVM) and generates machine code that can be directly executed by computer.

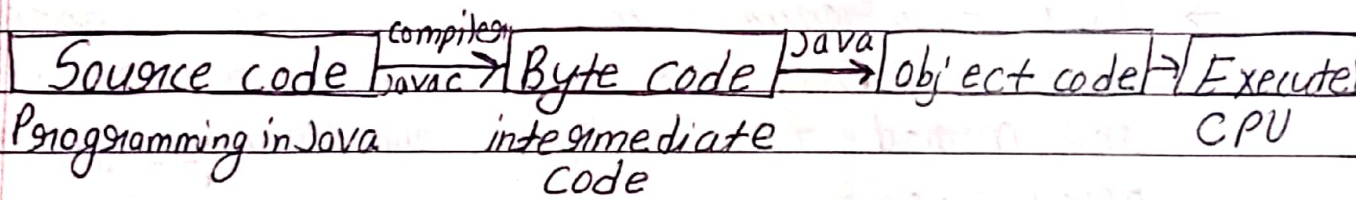

fig: compilation & Interpretation process

5. Multithreading and interactive

→ Multithreading means handling multiple task simultaneously, Java support Multithreading programs. This means that we didn't wait other applications to finish one task before beginning. This features greatly improves the interative performance of graphical application.

a Thread :- Thread is flow of execution of the process code.

FIFO - First In First Out

6. Scalability and performance

→ Java ensures a significant increase in Scalability and performance by improving the start-up time and reduced the amount of memory used in runtime environment.

Java Environment

→ Java Environment includes a large number of development tools and hundred of classes and method for developing and running java programs.

→ Java development tools are part of the system known as Java Development kit (JDK) and the classes and methods are part of the Java Standard Library (JSL), also known as Application programming Interface (API).

- Java Development kit (JDK)

The Java development kit (JDK) comes with collection of tools that are used for developing and running Java programs.
They includes:-

① Appletviewer: It enables to run java applets without using Java compatible browser.

② Javac : The java compiler, which translates java source code into bytecode file the interpreter can understand.

③ Java : Java interpreter, which runs applets and applications by reading and interpreting bytecode file.

④ Javadoc : Create HTML-format documentation from Java source code file.

⑤ Javah : Produces header files for use with native methods.

⑥ Javap : Java disassembler, which enables to convert bytecode file into program description.

⑦ Jdb : Java debugger, which helps to find error in the program.

☆ Java standard Library (JSL) or Application programming Interface (API)

→ API includes hundreds of class and method grouped into several functional packages.

→ Package is group of related classes and methods.

→ Java has following common packages:-

1. Language Support Package (Java.lang):- Includes classes and method for implementing basic features of Java. This package is automatically imported.

2. Utility Package (Java.util):- Includes classes to provides utility functions such as data and time.

3. Input/output Package (Java.io):- Includes classes for I/O Manipulation.

4. Networking Package (Java.net):- Includes classes for supporting networking operations.

5. Abstract Window Tool (AWT) Package (Java.awt):- Includes classes for implementing the components of graphical user interface (like button, menu etc).

6) Applet Package (Java.applet):- Includes classes for creating applets programs.

Note: If we use any API classes and method in program we must import package related to the classes or methods.

→ Import keyword is used to import package in the program. Eg:- Import Java.io.*;
Import Java.applet.*;

Basic Program Structure of Java :-

→ A typical structure of a Java programs consists of following parts.

1) Documentation section
2) Package declaration
3) Import statement
4) Interface section
5) class definition
6) Class variables
7) Main method class
8) Methods and behaviors

1) Documentation section

→ This section includes basic information about Java programs. The information includes author's name, data of creation, version, program name, company name etc. of the program. It includes the readability of the program but optional in Java programs. To write the statement in the documentation section, we use comments.

The comments be single line, Multiline line and documentation comments.

a) single line comment :- starts with a pair of forwarding slash (//).

Eg :- // This is first Java program

b) Multiline comment :- It start with a /* and ends with */.

eg
```
/* This the example of Multiline
   Comments */
```

c) Documentation comment :- It start with the delimiter (/**) and ends with */.

eg
```
/** This the example of documentation
   comment */
```

2) Package Declaration

→ It is optional placed just after documentation Section.

→ In this section we declare name in which is placed.

→ Note that there can be only one package Statement in java programs.

→ It is necessary because a java class can be placed in different package and directory based on the module they are used.

Package keyword is used to declare package name.

syntax:

```
Package Package-name;
```

eg:-

```
Package student;
Package school.student;
```

3) Import statement

→ Package contains many pre-defined classes and interface.

→ If you want to use any class of a particular package, we need to import that class.

→ Import keyword is used to import the class in program.

→ In java, import statement is used in two ways :- Either import a specific class or import all classes of a particular package

For example : import java.util.Scanner;
// import scanner class only

import java.util.*;
// import all the classes of java.util package.

4) Interface Section

→ It is an optional

→ An interface is created in this section.

→ An interface is slightly different from class that contains only constant and method decleration.

→ Interface keyword is used to create an interface.

For example;

```
interface car
{
  Void start();
  Void stop();
}
```

5) Class definition

→ The classes is a blueprint of a java program

→ It ~~converts~~ contains user-defined methods variables and constant.

→ A java program contains one or more classes.

→ Class keyword is used to define class.

Example:

```
class student    // class definition
{
    body of class
}
```

6) Class variables and constant

→ After class definition, variables and constant are defined.

→ Variable and constant store values of parameters.

→ Also can define the scope of variables by access modifiers.

Example:

```
class student // class definition
{
    int age;  // variable definition
    String name;
    double percentage;
}
```

7) Main method class

→ In this section, the main() method is defined

→ Must defined inside one class

→ Execution of all java programs starts from the main() method.

→ Inside main() method object of class all created and method are called.

→ Following statement is used to define main() method

```
public static void main (String args[])
    {

    }
```

Here,

- public: Public is an access specifier. It must be public keyword before main() method so that JVM can identify the execution point of programs.
- Static: Static is keyword that makes the main ~~(+) method~~ () method to call without creating object. static methods are invoked without creating the objects, so we do not need any object to call main() method.
- Void: Void is return type acknowledges the compiler that the main() method does not return any value.
- main(): main is a default signature which is predefined in the Java (JVM) Virtual Machine
- String args[]: It is the String type arguments accepted by main() method. It accepts a group of String, which is called String array.

Sample Java Program

```
// program to display own name.
import java.lang.*;
class Test
{
    public static void main(string args[])
    {
        System.out.println("My name is Bipana.");
    }
}
```

Compiling and Running Java program

Using command promt:

1) First install JDK in your computer
2) Open text editor such as notepad and type code
3) Save it with .java extension (eg. Myfile.Java)
4) Open command promt
5) Use following command to compile java program. It generates a class file in same folder.
   (Java Myfile.Java)
6) The following command to run the java program.
   (Java Myfile)
7) Output will appears like this: "My name is Bipana

Note: Java program can be compile & run using Integrated Development Environment (IDE) Such as NeatBeam.

Illustration of class

→ A class is a blueprint for the object.
→ It encapsulates field (Data) and methods (functions) in a single unit.
→ It is designed before creating an object.
→ Created using class keyword.

Syntax

```
class class_name
{
    // Fields
    // methods
```

Here,

Fields (variables) and methods represents the state and behaviour of object respectively.

→ Fields are used to store data.
→ methods are used to perform some operation.

Examples:

```
class Rectangle
{
    int length;
    int breadth;
    int area (int l, int b)
    {
        return (l * b);
    }
}
```

Illustration of object

- An object is called an instance of a class.
- A java object is a member of a java class.
- Each object has an identity, a behaviour and state
- The state of of an object is stored in fields (variables), where methods display the object behaviour.
- objects are created using new statement.

Syntax:

```
ClassName object = new className();
```

where,

className() is constructor. Constructor is similar to method and have same name as class & no return type.

Examples

```
Rectangle obj1 = new Rectangle();
```

// program illustrating class and object

```
class Rectangle
{
    int length;
    int breadth;
    void setData (int l, int b)
    {
        length = l;
        breadth = b;
    }
```

```
int RectArea()
{
    int area;
    area = length * breadth;
    return area;
}
}

class RectangleArea
{
    public static void main(String args[])
    {
        Rectangle obj = new Rectangle(); // creating object
        obj.setData(10, 5); // methods calling
        int result = obj.RectArea();
        System.out.println("Area of Rectangle = "+result);
    }
}
```

- Data Abstraction (Data Hiding)

→ Data abstraction is the process of hiding certain details and showing only essential information to user.

→ Abstraction can be achieved with either abstract class or interface.

→ The abstract keyword a non-access modifier used for classes and methods.

→ Abstract class is restricted class that can not be used to create objects. To access it, it must be inherited from another class.

→ Abstract method can only be used in abstract class, and it does not have a body. The body is provided by the subclass.

→ An abstract class can have both abstract and regular methods.

// program illustrating data abstraction in java.

```
abstract class Animal {
    public abstract void animalSound();
    public void sleep()
    {
        System.out.println("zzz");
    }
}

class Pig extends Animal
{
    public void animalSound()
    {
        System.out.println("The pig sound");
    }
}

class AbstractionTest
{
    public static void main(String args[])
```

```
{
    pig obj: new pig();
    obj. animalsound();
    obj. sleep();
}
}
```

* Encapsulation

↳ Encapsulation is the process or mechanism of wrapping the data (variables) and code acting on the data (method) together as a single unit.

↳ In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of current class.

↳ Therefore, it is also known as data hiding.

To achieve encaplusation in java:

1. Declare the variables of a class as private.
2. Provide public getter and setter methods to modify and view variable's value.

```
// Program to illustrate encpsulation in java
    public class EncapTest
    {
        private String name;
        private int age;
```

```
public int getAge()
{
    return age;
}
public String getName()
{
    return name;
}
public void setAge(int newage)
{
    age = newage;
}
public void setname(String newname)
{
    name = newname;
}
}
public class Encaplun
{
    public static void main(String args[])
    EncapTest obj = new EncapTest();
    obj.setName("Bipana");
    obj.setAge(17);
    System.out.println("Name:"+obj.getName());
    System.out.println("Age:"+obj.getAge());
}
```

getter method

setter method

* Polymorphism

↳ The polymorphism refers to the ability of a class to provide different implementation of method depending on the type of object is passed to the method.

↳ The same entity (method or operator or subject) can perform different operations in different scenario

↳ So, polymorphism is the ability of an object to take many forms.

↳ The most use of polymorphism occurs when a parent class reference is used to refer to a child class object.

// Program to illustrate polymorphism in java

```
class Polygon
{
  public void input()
  {
    System.out.println("I am the polygon");
  }
}

class Square extends Polygon
{
  public void input()
  {
```

```
    System.out.println("I am the square");
  }
}
class Circle extends Polygon
{
  public void input()
{
  System.out.println("I am the circle");
  }
}
  class PolyResult
{
  public static void main(string args[])
{
  Square S1 = new Square();
    S1. ~~square~~ input();
  Circle C1 = new Circle();
    C1. input();
  Polygon P1 = new Polygon();
    P1. input();
  }
}
```

* Types of polymorphism
1) Method overloading
2) Method overriding

1) Method Overloading:

⇒ Method overloading is the process that can creat multiple methods of same name, in the same class, and all methods work in different ways.

↳ It occurs when their is more than one method of the same name in the class.

```
// Example illustrating method overloading.

Class ~~Circle~~ CalculateArea
    {
      public void area()
    {
      System.out.println("Calcuting area");
    }
      public void area(int r)
    {
      System.out.println("The are of
                  circle ="~~3+3~~+3.14*r*r);
    }
      public void area(int L, int b)
    {
      System.out.println("The area of rectangle
                  ="+L*b);
    }
}
  class Test
{
  public Static void main (String args[])
{
      CalculateArea obj = new CalculateArea();
      Obj.area();
      obj.area(6);
      obj.area(6, 7);
}
}
```

// Program to input two integer numbers from user find sum.

```
//using Java scanner class
    import java.util.*;
    class CalculateSum
    {
    public static void main(String[] args)
    {
        int a, b, sum;
    Scanner sc = new Scanner(System.in);
            //System.in is a standard input
            Stream
    System.out.println("Enter first number:");
            a = sc.nextInt();
        System.out.println("Enter second number:");
                b = sc.nextInt();
                sum = a+b;
    System.out.println("Total sum =" + sum);
    }
    }
```

Java Scanner Class

↳ Scanner class allows to take input from user.
↳ This class belongs to java.util package
↳ It is used to input primitive data types such as int, double, long, short, float and byte.

Syntax:

```
Scanner sc = new Scanner(System.in);
```

This creates a constructor of the scanner class having scanner.in as an argument. It means

it means it is going to read from the standard input stream of the program.

- sc is an object of class Scanner.

Methods of Scanner class

1. nextInt() : reads integer value
2. nextFloat() : reads floating value
3. nextDouble() : reads double value
4. nextByte() : reads Byte value
5. nextLine() : reads single line or string
6. nextBoolean() : boolean value either true or false
7. nextLong() ; reads long value

Method overriding:

↳ Method overriding enables to define, and use methods repeatedly in sub class without defining in super class.

Examples:

```
class Super
{
 int x;
 Super(int x)
 {
  this.x = x;
 }
  void Display()
 {
```

```
System.out.println("super x = "+x);
}
}
class sub extends Super
{
  int y;
  sub(int x, int y)
{
    super(x);
    this.y = y;
}
  void Display()
{
  System.out.println("super x = "+x);
  System.out.println("super y = "+y);
}
}
class override-Test
{
public static void main(String args[])
{
  public Sub S1 = new Sub(10,20)
        S1.Display();
    }
  }
```

↳ In short, if sub class has the same method as declared in the super class, it is known as method overriding.

```
System.out.println("super x = "+x);
}
}
class sub extends Super
{
  int y;
  sub(int x, int y)
{
    super(x);
    this.y = y;
}
  void Display()
{
  System.out.println("super x = "+x);
  System.out.println("super y = "+y);
}
}
class override-Test
{
public static void main(String args[])
{
  public Sub S1 = new Sub(10,20)
        S1.Display();
    }
  }
```

↳ In short, if sub class has the same method as declared in the super class, it is known as method overriding.