UNIT-2

Object Oriented Programming:

As the name suggests, 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.

OOPs Concepts:

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.

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.

Features of Java

Object Oriented: In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

Platform Independent: Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform-independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

Simple: Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

Architecture-neutral: Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

Portable: Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. The compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

Robust: Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.

Multithreaded: With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

Interpreted: Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

High Performance: With the use of Just-In-Time compilers, Java enables high performance.

Distributed: Java is designed for the distributed environment of the internet.

Dynamic: Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.

Primitive Data types:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores all whole numbers
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

variables:

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.Variable is a name of memory location. There are three types of variables in java: local, instance and static.

Example: int data=50;

Operators:

Unary: expr++, expr, ++expr, –expr

Arithmetic: * / % + -

Relational: < > <= >=

Equality: == !=

Bitwise: & ^ |

Logical: &&, ||

Ternary : ? :

Assignment: = += -= *= /= %= &= ^= |= <<= >>= >>>=

Control Statements:

while Loop:

to execute a statement or code block repeatedly as long as an expression is true

Syntax: while (expression) {

  Statement(s) to be executed if expression is true

}

do...while Loop

Syntax:do {

  Statement(s) to be executed;

} while (expression);

for Loop:

 for (initialization; test condition; iteration statement) {

  Statement(s) to be executed if test condition is true

}

Arrays

Array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

One Dimensional Array in Java

Syntax to Declare an Array in Java

dataType[] arr; (or)

dataType []arr; (or)

dataType arr[];

Instantiation of an Array in Java

arrayRefVar = new datatype[size];

Example: int a[]=new int[5];

int a[]={33,3,4,5};

Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

dataType[][] arrayRefVar; (or)

dataType [][]arrayRefVar; (or)

dataType arrayRefVar[][]; (or)

dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in Java

int[][] arr=new int[3][3];

Classes

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.

A class in Java can contain:

Fields, Methods, Constructors, Blocks and Nested class and interface

Example:

```
class  Student{
int   id;
String  name;
public static void main(String args[]){
  Student  s1=new  Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Methods

A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the easy modification and readability of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

Example:

```
public class EvenOdd
{  public static void main (String args[])  {
Scanner scan=new Scanner(System.in);
int num=scan.nextInt();
findEvenOdd(num);
}
public static void findEvenOdd(int num)  {
if (num%2==0)  System.out.println(num+" is even");
else   System.out.println(num+" is odd");  }
}
```

Inheritance:

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.

It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.

When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Advantage of inheritance:

For Method Overriding (so runtime polymorphism can be achieved).

For Code Reusability.

Types of Inheritance:

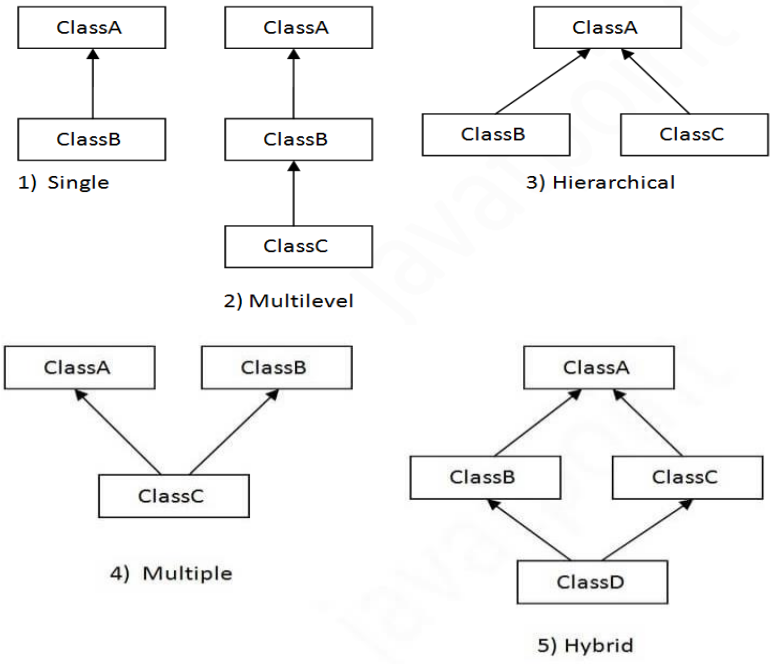
1. Single: When a class inherits another class, it is known as a single inheritance.

2. Multi Level: When there is a chain of inheritance, it is known as multilevel inheritance.

3. Multiple: When one class inherits multiple classes, it is known as multiple inheritance. Multiple and hybrid inheritance is supported through interface only.

4. Hierarchical: One base class is used to create many sub classes.

5. Hybrid: uses more than on type of inheritance.



.

Single Inheritance Example:

```
class Animal{

void eat(){System.out.println("eating...");}
```

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

MultiLevel Inheritance Example:

```
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 TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Packages:

A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code.

Creating Packages and importing:

The directory structure on your computer is related to the package name.

Suppose we have these two classes:

C:\temp\packagea\ClassA.java

```
package packagea;

public class ClassA {

}
```

C:\temp\packageb\ClassB.java

```
package packageb;

import packagea.ClassA;

public class ClassB {
 public static void main(String[] args) {
 ClassA a;
 System.out.println("Got it");
 }
}
```

Create the two files:

- C:\temp\packagea\ClassA.java
- C:\temp\packageb\ClassB.java

To Compile

```
javac packagea/ClassA.java packageb/ClassB.java
```

To Run

```
java packageb.ClassB
```

output:

```
Got it
```

Interfaces

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also represents the IS-A relationship.

It cannot be instantiated just like the abstract class.

Since Java 8, we can have default and static methods in an interface.

Since Java 9, we can have private methods in an interface.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

1. It is used to achieve abstraction.

2. By interface, we can support the functionality of multiple inheritance.

3. It can be used to achieve loose coupling.

Example:

```
interface Drawable{
void draw();
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class TestInterface1{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
}}
```

Multiple Inheritance using interfaces:

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

```
}
interface Showable{
void show();
}
class  A7  implements  Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}
public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

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.

Exception:

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.

Advantage of Exception handling:

To maintain the normal flow of the application.

To display user friendly error messages.

Exception types:

1. **Checked** Exceptions: classes that are direct sub classes of Exception class.

2. **UnChecked** Exceptions: sub classes of RuntimeException class.

Hierarchy of Java Exception classes

Throwable

Error

Exception

IOException

ClassNotFoundException

InterruptedException

SQLException

RuntimeExceptions

ArithmeticException

NullpointerException

NumberFormatException

IndexOutOfBoundsException

Java Exception Keywords

try: The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.

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.

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

throw: The "throw" keyword is used to throw an exception.

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

Exception handling Example:

```
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...");
  }
}
```

Multithreaded Programming:

Multithreading in Java is a process of executing multiple threads simultaneously.

Thread: 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.

Life Cycle of a Thread:

1. New: A new thread is created

2. Runnable: the thread is ready to run waiting for processor.

3. Running: the thread under execution

4. Blocked: block on I/O, or sleep(), wait() & suspend()

5. Terminated: completed the task and is terminated

Thread Example by extending Thread class

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

```
 }
}
```

Thread Example by implementing Runnable interface

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

Input/Output

Java I/O (Input and Output) is used to process the input and produce the output.

Java uses the concept of a stream to make I/O operation fast.

The java.io package contains all the classes required for input and output operations.

Stream:

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) System.out: standard output stream

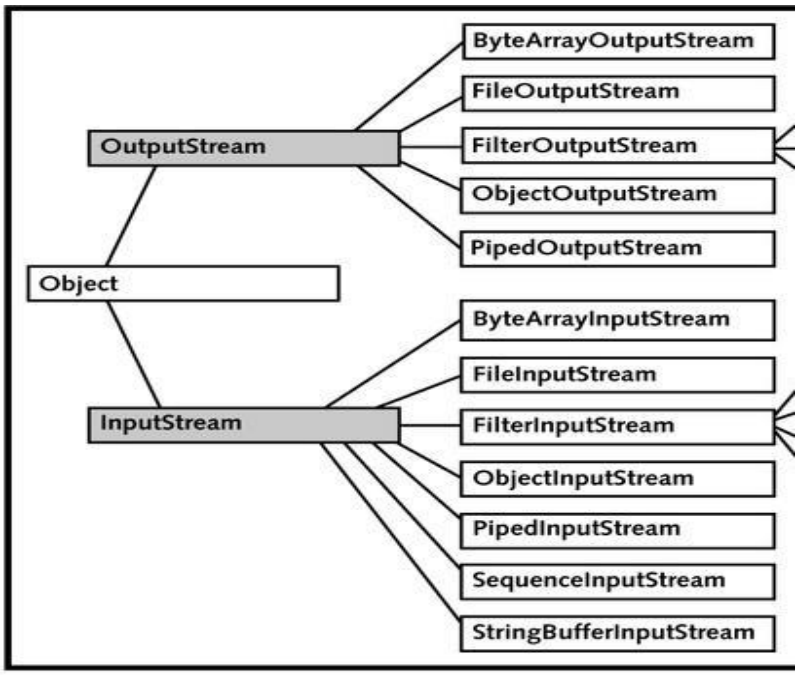
2) System.in: standard input stream

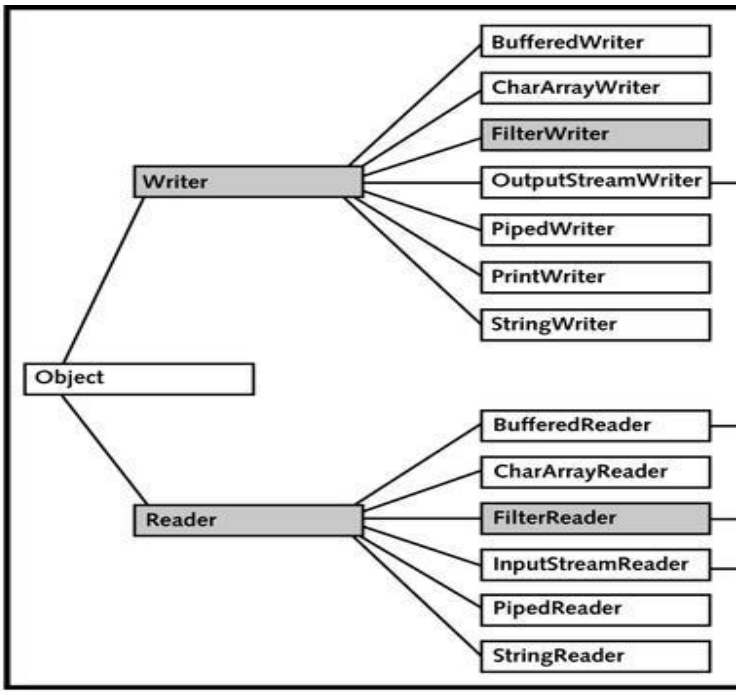
3) System.err: standard error stream

JAVA I/0 Classes:

1. Byte Streams: Read/Write the data as bytes. These classes are used to read/write primitive data types, objects. InputStream & OutputStream are the base classes for these classes.

2. Character Streams: Read/Write data as characters. These classes are used to read/write text data. Reader/Writer are the base classes for these classes.





Byte Streams - InputStream and OutputStream

```
import java.io.*;
public class fileStreamTest {
   public static void main(String args[]) {
      try {
```

```
 byte bWrite [] = {11,21,3,40,5};
 OutputStream os = new FileOutputStream("test.txt");
        for(int x = 0; x < bWrite.length ; x++) {
           os.write( bWrite[x] );   // writes the bytes
        }
 os.close();
InputStream is = new FileInputStream("test.txt");
int size = is.available();
for(int i = 0; i < size; i++) {
           System.out.print((char)is.read() + "  ");
        }
is.close();
} catch (IOException e) {
System.out.print("Exception");
}}}
```

Character Streams - FileWriter Example:

```
import java.io.*;
public class WriterExample {
    public static void main(String[] args) {
        try {
            Writer w = new FileWriter("output.txt");
            String content = "I love my country";
            w.write(content);
            w.close();
            System.out.println("Done");
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

Utility Classes:

utilities in java.util provide the more complex programming structures that you'll want to use in any professionally written classes, but aren't in the basic language. The majority of utility classes are concerned with collections, but the package also includes classes for date and time work, resource bundle handling, parsing a string (StringTokenizer) and the Timer API.

1. ArrayList: class provides resizable-array and implements the List interface.

2. Calendar: class is an abstract class that provides methods for converting between a specific instant in time and a set of calendar fields.

3. Collections class consists exclusively of static methods that operate on or return collections.

4. Formatter class provides support for layout justification and alignment, common formats for numeric, string, and date/time data, and locale-specific output.

5. HashMap class is the Hash table based implementation of the Map interface.

6. HashSet class implements the Set interface, backed by a hash table.

7. PriorityQueue class is an unbounded priority queue based on a priority heap.

8. Random class instance is used to generate a stream of pseudorandom numbers.

9. Scanner class is a simple text scanner which can parse primitive types and strings using regular expressions.

10. StringTokenizer class allows an application to break a string into tokens.

11. Timer class provides facility for threads to schedule tasks for future execution in a background thread

12. TreeMap class is the Red-Black tree based implementation of the Map interface.

13. TreeSet class implements the Set interface. The TreeSet class guarantees that the Map will be in ascending key order and backed by a TreeMap.

String

string basically represents sequence of char values.

In Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created.

Example:

1.String s1=”Shiva”; // string literal created on the string pool

2. String s2=new String(“Ravi”) // an exclusive string object created on the pool

Java String class methods

1. charAt(int index): returns char value for the particular index

2. length(): returns string length

3. substring(int beginIndex): returns substring for given begin index.

4. contains(CharSequence s): returns true or false after matching the sequence of char value.

5. join(): returns a joined string.

6. equals(): checks the equality of string with the given object.

7. concat(): concatenates the specified string.

8. split(String regex): returns a split string matching regex.

9. indexOf(int ch): returns the specified char value index.

10. toLowerCase(): returns a string in lowercase.

11. toUpperCase(): returns a string in uppercase.

12. trim(): removes beginning and ending spaces of this string.