OOPS CONCEPTS AND FILE HANDLING IN JAVA

OBJECT ORIENTED PROGRAMMING

- Oops in java is to improve code readability and reusability by defining a Java program efficiently. The main principles of object-oriented programming are abstraction, encapsulation, inheritance, and polymorphism. These concepts aim to implement real-world entities in programs.

LIST OF OOP’s CONCEPTS IN JAVA

- Object
- Class
- Abstraction
- Inheritance
- Polymorphism
- Encapsulation

OBJECTS

Objects are always called instances of a class which are created from a class in java or any other language. They have states and behaviour.

These objects always correspond to things found in the real world, i.e., real entities. So, they are also called run-time entities of the world. These are self–contained which consists of methods and properties which make data useful. Objects can be both physical and logical data. It contains addresses and takes up some space in memory. Some examples of objects are a dog, chair, tree etc.

When we treat animals as objects, it has states like colour, name, breed etc., and behaviours such as eating, wagging the tail etc.

Suppose, we have created a class called My book, we specify the class name followed by the object name, and we use the keyword new.

EXAMPLE PROGRAM FOR OBJECTS

```
Public class Mybook {
int x=10;
Public static void main (String args []) {
Mybook Myobj= new Mybook ();
System.out.println(MyObj.x);
}
}
```

In the above example, a new object is created, and it returns the value of x which may be the number of books.

Mybook Myobj= new Mybook ();

This is the statement used for creating objects.

System.out.println(Myobj.x);

This statement is used to return the value of x of an object.

CLASSES

Classes are like object constructors for creating objects. The collection of objects is said to be a class. Classes are said to be logical quantities. Classes don't consume any space in the memory. Class is also called a template of an object. Classes have members which can be fields, methods and constructors. A class has both static and instance initializers.

A class declaration consists of:

Modifiers: These can be public or default access.

Class name: Initial letter.

Superclass: A class can only extend (subclass) one parent.

Interfaces: A class can implement more than one interface.

Body: Body surrounded by braces, { }.

EXAMPLE PROGRAM FOR CLASSES

```
CLASSS classname {
type instance variable 1;
type instance variable 2;  l
Type instance variable n;
type methodname 1 (parameter list) {
// body od method
}
type methodname 2 (parameter list) {
// body od method
}
type methodnamen (parameter list) {
// body od method
}
 }
```

ABSTRACTION

Abstraction is a process which displays only the information needed and hides the unnecessary information. We can say that the main purpose of abstraction is data hiding. Abstraction means selecting data from a large number of data to show the information needed, which helps in reducing programming complexity and efforts.

There are also abstract classes and abstract methods. An abstract class is a type of class that declares one or more abstract methods. An abstract method is a method that has a method definition but not implementation. Once we have modelled our object using data abstraction, the same sets of data can also be used in different applications—abstract classes, generic types of behaviours and object-oriented programming hierarchy. Abstract methods are used when two or more subclasses do the same task in different ways and through different implementations. An abstract class can have both methods, i.e., abstract methods and regular methods.

EXAMPLE PROGRAM FOR ABSTRACTION

```
//abstract parent class
        Abstract class animal {
         //abstract method
       public abstract void sound ( ) ;
         }
      Public class lion extends animal {
       Public void sound ( ) {
System.out.println (“ roar “ );
}
public Static void main ( String args [ ] ) {
 animal obj = new lion ( );
obj. Sound ();
}
}
```

INHERITENCE

- Inheritance is a method in which one object acquires/inherits another object's properties, and inheritance also supports hierarchical classification. The idea behind this is that we can create new classes built on existing classes, i.e., when you inherit from an existing class, we can reuse methods and fields of the parent class. Inheritance represents the parent-child relationship

EXAMPLE PROGRAM FOR INHERITANCE

```
// a simple example of inheritance
//create a superclass
Class  Add {
int my;
int by;
void setmyby (int xy, int hy) {
my=xy;
by=hy;
}
}
/create a sub class
class b extends add {
int total;
void sum () {
public Static void main (String args [ ] ) {
b subOb= new b ( );
subOb. Setmyby (10, 12);
subOb. Sum ( ) ;
System.out.println(“total =” + subOb. Total);
}
}
```

POLYMORPHISM

- Polymorphism refers to many forms, or it is a process that performs a single action in different ways. It occurs when we have many classes related to each other by inheritance. Polymorphism is of two different types, i.e., compile-time polymorphism and runtime polymorphism. One of the examples of Compile time polymorphism is that when we overload a static method in java. Run time polymorphism also called a dynamic method dispatch is a method in which a call to an overridden method is resolved at run time rather than compile time. In this method, the overridden method is always called through the reference variable. By using method overloading and method overriding, we can perform polymorphism. Generally, the concept of polymorphism is often expressed as one interface, and multiple methods. This reduces complexity by allowing the same interface to be used as a general class of action.

EXAMPLE PROGRAM FOR POLYMORPHISM

```
Public class Bird {
Public void sound ( ) {
System.out.println ( " birds sounds " );
}
}
public class pigeon extends Bird {
@override
public void sound ( ) {
System.out.println( " cooing " ) ;
}
}
public class sparrow extends Bird ( ) {
@override
Public void sound ( ){
System.out.println( " chip " ) ;
}
}
```

ENCAPSULATION

Encapsulation is one of the concepts in OOPs concepts; it is the process that binds together the data and code into a single unit and keeps both from being safe from outside interference and misuse. In this process, the data is hidden from other classes and can be accessed only through the current class's methods. Hence, it is also known as data hiding. Encapsulation acts as a protective wrapper that prevents the code and data from being accessed by outsiders. These are controlled through a well-defined interface.

Encapsulation is achieved by declaring the variables as private and providing public setter and getter methods to modify and view the variable values. In encapsulation, the fields of a class are made read-only or write-only. This method also improves reusability. Encapsulated code is also easy to test for unit testing.

EXAMPLE PROGRAM FOR ENCAPSULATION

```
Class animal {
// private field
private int age;
//getter method
Public int getage ( ) {
return age;
}
//setter method
public void setAge ( int age ) {
this. Age = age;}
}
class Main {
public static void main (String args []);
Animal a1= new Animal ();
A1. setAge (12);
// access age using getter
System.out.println(“ animal age is ” + a1. getage ( ) );
}
}
```

FILE HANDLING DEFINITION

- File handling holds immense significance in programming languages as it empowers us to store program outputs in files and facilitates various operations on them. Essentially, file handling entails the act of reading and writing data to and from a file.
- You use streams to conduct input/output (I/O) operations on files. It is critical to understand how file operations are carried out in Java by becoming acquainted with this idea.Java has two types of streams: input streams for reading data and output streams for writing data.

INPUT STREAMS

The Java InputStream class serves as the superclass for all input streams. Input streams are responsible for reading data from various input devices, such as keyboards or network connections. While InputStream is an abstract class and not directly useful on its own, its subclasses are utilized for reading data.

Some notable subclasses of the InputStream class include:

AudioInputStream

ByteArrayInputStream

FileInputStream

FilterInputStream

StringBufferInputStream

ObjectInputStream

OUTPUT STREAMS

On the other hand, output streams are responsible for writing data to various output devices, such as monitors or files. The OutputStream class acts as an abstract superclass that represents an output stream. Similar to InputStream, OutputStream is abstract and requires the use of its subclasses for writing data.

Some notable subclasses of the OutputStream class include:

ByteArrayOutputStream

FileOutputStream

StringBufferOutputStream

ObjectOutputStream

DataOutputStream

PrintStream

FILE HANDLING EXAMPLE IN JAVA

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

- These are used for reading and deleting the files this is the detailed explanation of oops and file handling concepts in java

THE END