Chapter 2 : Essential Classes and OOP's Concepts

What is OOPs?

Object-oriented programming is a method used for designing a program using classes and objects. Object-oriented programming is also called the core of java. Object-oriented programming organizes a program around objects and well-defined interfaces. This can also be characterized as data controlling for accessing the code.

In this type of approach, programmers define the data type of a data structure and the operations that are applied to the data structure.

This implies software development and maintenance by using some of the concepts :

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

What is OOPs concepts in java?

OOPs, concepts 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.

What are Objects?

Objects are always called as instances of a class. Objects are created from class in java or any other languages. Objects are those that have state and behaviour. Objects are abstract data types (i.e., objects behaviour is defined by a set of values and operations).

What are 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: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, { }.

A class keyword is used to create a class. A simplified general form of the class definition is given below:

class classname {

type instance variable 1;

type instance variable 2;

.

.

.

type instance variable n;

type methodname 1 (parameter list) {

```
// body  method
}
type methodname 2 (parameter list) {
// body  method
}
type methodname n (parameter list) {
// body method
}
 }
```

The variables or data defined within a class are called as instance variables. Code is always contained in the methods. Therefore, the methods and variables defined within a class are called members of the class. All the methods have the same form as main () these methods are not specified as static or public.

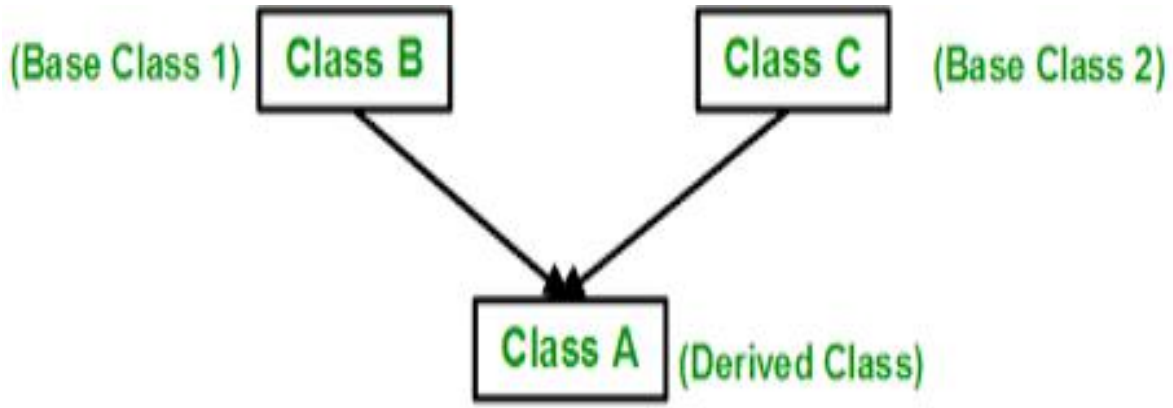
What is Abstraction?

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. 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.

What is Inheritance?

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.

To inherit a class, we use the extend keyword.



What is 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 in Compile time polymorphism is that when we overload a static method in java.

Run time polymorphism is 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, multiple methods.

This reduces complexity by allowing the same interface to be used as a general class of action.

Example :

```
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 ” ) ;
}
}
```

Polymorphism in java can be classified into two types :

Static / Compile-Time Polymorphism

Dynamic / Runtime Polymorphism

What is Compile-Time Polymorphism in Java?

Compile-Time polymorphism in java is also known as Static Polymorphism. to resolved at compile-time which is achieved through Method Overloading.

What is Runtime Polymorphism in Java?

Runtime polymorphism in java is also known as Dynamic Binding which is used to call to an overridden method that is resolved dynamically at runtime rather than at compile-time.

What is 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 the re-usability. Encapsulated code is also easy to test for unit testing.

Example :

```
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 []);
//create an object of person
Animal a1= new Animal ();
//change age using setter
A1. setAge (12);
// access age using getter
System.out.println(“ animal age is ” + a1. getage ( ) );
}
}
```

Output : Animal age is 12

Exception Handling in Java

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

What is Exception in Java?

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.

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error.

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked.

An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

- Checked Exception
- Unchecked Exception
- Error

Difference between Checked and Unchecked Exceptions

1. **Checked Exception :** The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2. **Unchecked Exception :** The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3. **Error :** Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

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