## ****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).

These objects always correspond to things found in the real world, i.e., real entities. So, they are also called a run time entity of the world. These are self–contained which consists of methods and properties which makes data useful. Objects can be both physical and logical data. It contains addresses and takes up some space in memory. Some of the 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 1:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 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.

We can also create multiple objects in the same class and we can create in one class and access it in another class. This method is used for better organization of classes and always remember that name of the java file and the class name remains the same.

**Example 2:**

The below example shows how multiple objects are created in the same class and how they are accessed from another class.

* Mybook.java

|  |  |
| --- | --- |
| 1  2  3  4 | Public class Mybook {  int x=10;  int y=8;  } |

* Count.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Class Count {  Public static void main (String [] args)  {  Mybook myobj1 = new myobj1();            Mybook myobj2 = new myobj2();             System.out.println (myobj1.x);  System.out.println (myobj2.y);  }  } |

When this program is compiled, it gives the result as 10, 8 respectively.

## ****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:

1. **Modifiers**:Can be public or default access.
2. **Class name:** Initial letter.
3. **Superclass:** A class can only extend (subclass) one parent.
4. **Interfaces:** A class can implement more than one interface.
5. **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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | class classname {  type instance variable 1;  type instance variable 2;  .  .  .  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  }   } |

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.

There are also abstract class 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 the methods, i.e., abstract methods and regular methods.

Now let us see an example related to abstraction.

Suppose we want to create a student application and ask to collect the information about the student.

We collect the following information.

* Name
* Class
* Address
* Dob
* Fathers name
* Mothers name and so on.

We may not require every information that we have collected to fill the application. So, we select the data that is required to fill the application. Hence, we have fetched, removed, and selected the data, the student information from large data. This process is known as abstraction in oops concept.

**Abstract class example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | //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 ();  }  } |

**Output:**  
Roar

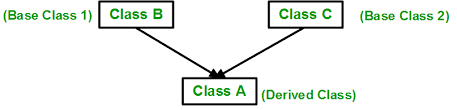
## ****What is Inheritance?****

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.

For example, a whale is a part of the classification of marine animals, which is part of class mammal, which is under that class of animal. We use hierarchical classification, i.e., top-down classification. If we want to describe a more specific class of animals such as mammals, they would have more specific attributes such as teeth; cold-blooded, warm-blooded, etc. This comes under the subclass of animals where animals come under superclass. The subclass is a class which inherits properties of the superclass. This is also called a derived class. A superclass is a base class or parental class from which subclass inherits properties.

We use inheritance mainly for method overriding and R:

To inherit a class, we use the extend keyword.



There are five types of inheritance single, multilevel, multiple, hybrid and hierarchical.

* **Single level**

In this one class i.e., derived class inherits properties from its parental class.  This enables code reusability and also adds new features to the code. Example: class b inherits properties from class a.

Class A is base or parental class and class b is derived class.

**Syntax:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Class a {  …  }  Class b extends class a {  …  } |

* **Multilevel**

This one class is derived from another class which is also derived from another class i.e., this class has more than one parental class, hence it is called multilevel inheritance.

**Syntax:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Class a {  ….  }  Class b extends class a {  ….  }  Class c extends class b {  …  } |

* **Hierarchical level**

In this one parental class has two or more derived classes or we can say that two or more child classes has one parental class.

**Syntax:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Class a {  …  }  Class b extends class a {  ..  }  Class c extends class a {  ..  } |

* **Hybrid inheritance**

This is the combination of multiple and multilevel inheritance and in java multiple inheritance is not supported as it leads to ambiguity and this type of inheritance can only be achieved through interfaces.

Consider that class a is the parental or base class of class b and class c and in turn class b and class c are parental or base class of class d. Class b and class c are derived classes from class a and class d is derived class from class b and class c.

Following program creates a super class called add and a subclass called sub, uses extend keyword to create a subclass add.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | // 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);  }  } |

It gives output as – **total = 22**

## ****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:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | 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 ” ) ;  }  } |

In the above example, we can see common action sound () but there are different ways to do the same action. This is one of the examples which shows polymorphism.

Polymorphism in java can be classified into two types:

1. Static / Compile-Time Polymorphism
2. 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:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | 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

In this example, we declared a private field called age that cannot be accessed outside of the class.

To access age, we used public methods. These methods are called getter and setter methods. Making age private allows us to restrict unauthorized access from outside the class. Hence this is called data hiding.

## ****Coupling in Java****

Couplingrefers to the relationship between two classes. It indicates the knowledge one object or class has of another. That means that if one class changes its properties or behavior, it will affect the dependent changes in the other class. Therefore, these changes will depend upon the level of interdependence the two classes have between them. There are two types of coupling, namely tight coupling, and loose coupling.

* **Tight coupling:**If one class is strongly interrelated to another class, it is said to have tight coupling with that class.

public class College{

public void status() {

System.out.println("College is open today");

}

}

public class Student{

College obj = new College();

public void goToCollege() {

obj.status();

}

}

In the above code example, the student class is dependent on the college class. That is, any change in the college class requires student class to change. Here, therefore, student class and college class are tightly coupled with each other.

* **Loose coupling:** If one class is weakly interrelated to another class, it is said to have loose coupling with that class. Loose coupling is preferred over tight coupling. A class can achieve this with the help of interfaces, as shown below.

public interface College{

void status();

}

class CollegeStatus1 implements College{

public void status() {

System.out.println("College is open monday to friday");

}

}

class CollegeStatus2 implements College{

public void status() {

System.out.println("College is open on saturday");

}

}

public class Student{

College obj = new CollegeStatus1();

public void goToCollege() {

obj.status();

}

}

In the above code example, CollegeStatus1 and CollegeStatus2 are loosely coupled. Here, student class is not directly or tightly coupled with a CollegeStatus1 or CollegeStatus2 class. By applying a dependency injection mechanism, the loose coupling implementation is achieved to allow a student to go to college with any class which has implemented a college interface. In addition, it means we can use CollegeStatus2 whenever the college is open on Saturday.