**Pillars of OOPS**

**1.Objects-**

an entity that has state and behaviour is an object.Ex. chair, pen, mouse.

**2.Classes-**

Collection of Objects is known as class.

**3.Inheritance-**

if an object acquires behaviour and properties of its parent class.In java 'extends' keyword is used to inherit class.

Syntax:

class sub\_classname extends Super\_classname

Types of Inheritance:

1.Single level

2.Multievel

3.Hierarchical

4.Multiple(is not allowed in Java because of ambiguity)

5.Hybrid

**4.Polymorphism-**

When a task is performed in different ways polymorphism can be achieved. Example, to speak something like a dog barks, a cats meaw

In java, method overloading and method overriding is used to achieve polymorphism.

**5.Abstraction-**

Hiding internal details and showing functionality. Example, a phone call as we don’t know the internal processing.

**6.Encapsultion-**

Wrapping up of data into a single unit is known as Encapsulation. Ex. A capsule.

**Data Members-**

Data members are those whose memory is created each and every time when object is created.

**Members functions-**

**Methods in Java-**

Methods are used perform some operations and are of two types:

**Non-Static**- used to perform repeated operations

**Static**- used to perform only one operation. Main method is an example of static methods.

**Different type of Variable:**

**Instance Variable:**

Created inside the class but outside the method and can accessible anywhere in the class.

**Static Variable:**

If you put Static keyword with any variable it becomes Static variable. It is used to refer a common (not unique) property of all objects.

Ex. Making college name as static variable for student class.

**Difference between JDK, JRE and JVM-**

|  |  |  |
| --- | --- | --- |
| **JVM(Java Virtual Machine)** | **JRE(Java Runtime Environment)** | **JDK(Java Development Kit)** |
| It is an Abstract Machine. It is a speci-fication that provides runtime environment in which java byte code can be executed.  The JVM performs main tasks:   * Loads code * Verifies code * Executes code * Provide Runtime Environment | It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.  JRE contains everything required to ***run*** Java application which has already been compiled. | It physically exists. It contains JRE + development tools. |



* **JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.**
* **JRE = JVM + Required Library to run Application.**
* **JDK = JRE + Required Library to develop Java Application.**

**How to install JDK?**

* Browse to [Oracle Java SE Downloads (http://www.oracle.com/technetwork/java/javase/downloads/index.html)](http://www.oracle.com/technetwork/java/javase/downloads/index.html) to download the latest JDK.
* Check the option to Accept License Agreement, and select the download file. "jdk-8u60-windows-x64.exe" in my case.
* Run the downloaded file after download completed. And follow the steps.
* After installed JDK, you have to set Path to it.
* To set path, search "Environment Variables", click "Edit the system environment variables".
* Click on "Environment Variables..."
* Select and edit Path
* Add the  path of the bin folder of your JDK, "C:\Program Files\Java\jdk1.8.0\_60\bin" in my case.
* After path set, you can run javac in command prompt.

**Different task of JVM-**

* Loads code
* Verifies code
* Execute code
* Used to provide runtime environment.

**Working of JRE-**

JRE (Java Runtime Environment) is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

**Latest version of JDK-** JDK 8 is the latest version of JDK.

**Constructors-**

A constructor in Java is a block of code similar to a method that's called when an instance of an object is created.

**Difference b/w Methods and Constructors**

* **Methods** expose the behavior of object and **constructor** is used to initialize the object.
* **Methods** must have the return type but **Constructor**s must not have any return type.
* **Constructors** name same as the class name whereas **method** may or may not the same class name.

**Encapsulation-**

**Encapsulation** is one of the four fundamental OOP concepts. Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) 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 their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java −

* Declare the variables of a class as private.
* Provide public setter and getter methods to modify and view the variables values.

**Benefits of Encapsulation-**

By providing only setter or getter method, you can make the class **read-only or write-only**.

It provides you the **control over the data**. Suppose you want to set the value of id i.e. greater than 100 only, you can write the logic inside the setter method.

**Example:**

public class EncapTest {

private String name;

private String idNum;

private int age;

public int getAge() {

return age;

}

public String getName() {

return name;

}

public String getIdNum() {

return idNum;

}

public void setAge( int newAge) {

age = newAge;

}

public void setName(String newName) {

name = newName;

}

public void setIdNum( String newId) {

idNum = newId;

}

}

The public setXXX() and getXXX() methods are the access points of the instance variables of the EncapTest class. Normally, these methods are referred as getters and setters. Therefore, any class that wants to access the variables should access them through these getters and setters.

The variables of the EncapTest class can be accessed using the following program −

/\* File name : RunEncap.java \*/

public class RunEncap {

public static void main(String args[]) {

EncapTest encap = new EncapTest();

encap.setName("James");

encap.setAge(20);

encap.setIdNum("12343ms");

System.out.print("Name : " + encap.getName() + " Age : " + encap.getAge());

}

}

**Polymorphism-**

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

Example:

public interface Vegetarian{}

public class Animal{}

public class Deer extends Animal implements Vegetarian{}

Deer d = new Deer();

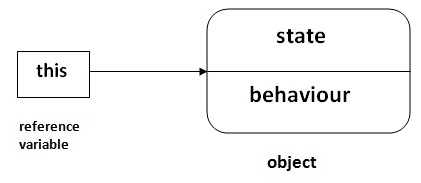
Animal a = d;

Vegetarian v = d;

Object o = d;

**this reference-**

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.

****

**JAVA Static Keyword-**

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

**Static Variable-**

It is used to refer a common property of objects (that is not unique).

e.g. Company name of employees, College name of students.

**Static Method-**

If you apply static keyword with any method, it is known as static method.

* A static method belongs to the class rather than object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* Static method can access static data member and can change the value of it.

**Static Block-**

* Is used to initialize the static data member.
* It is executed before main method at the time of classloading.

**Static Import-**

* There is no need to qualify it by the class name.\

**\*\* What is the difference between import and static import?**

The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification.

**Static Nested Classes-**

A static class i.e. created inside a class is called static nested class in java.

**Super Keyword-**

The **super** keyword in java is a reference variable that is used to refer immediate parent class object.

**Usage of super keyword-**

1. super is used to refer immediate parent class instance variable.
2. super() is used to invoke immediate parent class constructor.
3. super is used to invoke immediate parent class method.

**Final keyword-**

The final keyword in java is used to restrict the user.

## 1) Java final variable-

If you make any variable as final, you cannot change the value of final variable(It will be constant).

## 2) Java final method-

If you make any method as final, you cannot override it.

## 3) Java final class-

If you make any class as final, you cannot extend it.

**Method Overloading and Method Overriding-**

**Method Overriding-**

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

**class** Bank{

**int** getRateOfInterest(){**return** 0;}

}

**class** SBI **extends** Bank{

**int** getRateOfInterest(){**return** 8;}

}

**class** ICICI **extends** Bank{

**int** getRateOfInterest(){**return** 7;}

}

**class** AXIS **extends** Bank{

**int** getRateOfInterest(){**return** 9;}

}

**class** Test2{

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

SBI s=**new** SBI();

ICICI i=**new** ICICI();

AXIS a=**new** AXIS();

System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());

System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());

System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());

}

}

**Method Overloading-**

If a class has multiple methods by same name but different parameters, it is known as **Method Overloading**.

**\*\*How to overload a method-**

1. By changing number of arguments
2. By changing the data type

**class** Calculation{

**void** sum(**int** a,**int** b){System.out.println(a+b);}

**void** sum(**int** a,**int** b,**int** c){System.out.println(a+b+c);}

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

  Calculation obj=**new** Calculation();

  obj.sum(10,10,10);

  obj.sum(20,20);

  }

}