**Java OOPS Concepts for Selenium**  
OOPS - Object Oriented Programming System  
  
Four Fundamentals of OOPS:  
i) Inheritance  
  
ii) Polymorphism  
  
iii) Abstraction  
  
iv) Encapsulation  
------------------------  
**i) Inheritance:**  
> It is a process of inheriting (reusing) the class members variables and methods) from one class to another class is called Inheritance.  
  
> Non static (object level) class members only can be inherited.  
  
> The class from where the class members are getting inherited is called as Super class/ Parent class/base class  
  
> The class to which the class members are getting inherited is called  
Sub class/ Child class / Derived class.  
  
> The Inheritance between Super class and Sub class is achieved using "extends" keyword.  
  
Syntax:  
  
Class SubClass extends SuperClass {  
//body  
}  
They are three types of Inheritance:  
  
i) Single Inheritance  
  
Ex:  
  
Class B extends Class A  
---------------------------  
ii) Multi level inheritance:  
  
Ex:  
  
Class B extends Class A  
  
Class C extends Class B  
  
iii) Multiple Inheritance (\*Java doesn't support)  
  
Ex:  
  
Class B extends Class A  
  
Class C extends Class B  
  
In Class C  
add method  
-----------------------------  
Class C extends Class D  
-------------------------------  
  
Inheritance example:

package test;

import java.io.Console;

import java.lang.\*;

class parent

{

int a = 10;

int b = 20;

public void addition()

{

System.out.println("Addition of child a, b is: " + (a+b));

}}

class child extends parent{

int a =100;

int b =200;

public void addition(){

System.out.println("Addition of a, b is: " +(a+ b));

}

}

public class test {

public static void main (String [] args){

child myObject = new child();

parent parentObject = new parent();

myObject.addition();

}

}

----------------------  
class B {  
    int a = 10;  
    int b = 20;  
    public void addition(){  
        System.out.println("Addition of a, b is: " + (a+b));  
    }  
  public static void main (String [] args){  
      B myObject = new B();  
      myObject.addition();  
  }  
}  
---------------------------------  
package javaiooperations;  
  
class C extends B {  
/\*int a =100;  
int b =200;  
public void addition(){  
    System.out.println("Addition of a, b is: " +(a+ b));  
}\*/  
public static void main (String [] args){  
    C abc = new C();  
    abc.addition(); //   
}  
}  
-------------------------------  
package javaiooperations;  
  
public class D extends C{  
    /\*int a=1;  
    int b=2;  
    public void addition(){  
        System.out.println("Addition of a, b is: "+(a + b));  
    }\*/  
    public static void main (String [] args){  
   D obj = new D();  
   obj.addition();  
}  
}  
--------------------------------------  
**ii) Polymorphism:**  
Polymorphism means, existence of Object behavior in many forms.  
  
They are two types of polymorphism:  
  
i) Compile Time/ Static binding/ Method overloading  
  
ii) Run Time polymorphism / Dynamic binding / Method overriding  
-----------------------  
i) Compile Time  
-------------------  
If two or more methods having same method name in the same class but they differ in the following ways:  
  
i) No of Arguments  
  
Ex:  
  
add (int a, int b){  
}  
  
add (int a, int b, int c) {  
}  
--------------------------  
ii) Order of Arguments  
  
iii) Type of Arguments  
  
Ex:  
  
add (int a, int b){  
}  
  
add (double a, double b){  
}  
------------------------------------  
Example:  
package javaiooperations;  
  
public class MethodOverLoading {  
    public void add (int a, int b){  
        System.out.println(a+b);  
    }  
    public void add (int a, int b, int c){  
        System.out.println(a+b+c);  
    }  
      
        public void add (double a, double b){  
            System.out.println(a+b);      
    }  
    public static void main (String [] args){  
        MethodOverLoading obj = new MethodOverLoading();  
        obj.add(2, 5);  
        obj.add(2, 5, 7);  
        obj.add(1.234, 4.567);  
    }  
}  
---------------------------------  
ii) Run time/Method Overriding  
  
If two methods are having same name and same arguments available in the Super class and sub class, then we call those two methods are overridden.  
  
Ex:  
  
public class Y {  
int a = 10,  b=20;  
public void  add () {  
    System.out.println(a+b);  
}  
}  
-----------------  
public class Z extends Y{  
    int a = 1,  b=2;  
    public void  add () {  
        System.out.println(a+b);  
    }  
    public static void main (String [] args){  
        Z obj = new Z();  
        obj.add(); // 3  
          
        Y obj1 = new Y();  
        obj1.add(); //30  
          
        Y obj2 = new Z();  
        obj2.add();  
    }  
    }  
--------------------------------  
**iii) Abstraction:**  
> It is a process of hiding implementation details and showing only functionality to the user.  
  
> In Java, we have two types of methods  
  
i) Concrete Methods (The methods which are having body)  
  
Ex:  
  
public void addition()   
{  
// Method body  
}  
--------------  
ii) Abstract Methods (The methods which are not having body)  
  
Ex:  
  
public void addition ();  
--------------------------------  
If we know the method name, but don't know what the method performs, then we go for abstract methods.  
  
Inheritance:  
-------------  
A sub class extends Super class is known as Inheritance.  
  
Class members in java:  
  
Variables and Methods.  
  
Static Class Members (Class Level)  
  
> using class name we can access Static class members.  
  
Non Static Class Members (Object Level)  
  
> Using Object/Instance   
-----------------------  
Note 1: Static Class members are not inherited to the Sub class.  
  
Note 2: Non Static class members are inherited to the Sub class.  
--------------------------------------------  
Example for Accessing Static and Non static class members:  
-----------  
package JavaOOPS;  
public class AbstractionExample {  
    static int a = 10, b = 20; //Static variables  
    int c = 30, d = 40; // Non static variables   
      
    public static void add1() { // Static method  
        System.out.println(a+b);  
    }  
    public void add2() // Non static method  
    {  
        System.out.println(c+d);  
    }  
    public static void main (String [] args) {  
        // Access Static Class Members using Class Name  
        System.out.println(AbstractionExample.a); // 10  
        System.out.println(AbstractionExample.b); // 20  
        AbstractionExample.add1();// 30  
          
        System.out.println("");  
        // Access Non static class members using Object /Instance  
        AbstractionExample obj = new AbstractionExample();  
        System.out.println(obj.c); // 30  
        System.out.println(obj.d); // 40  
        obj.add2(); //70  
            }  
      
}  
---------------------------------------  
**Abstract Class**  
> java Class contains 100% concrete methods  
  
> Abstract (incomplete) class contains one or more abstract methods.  
  
> Abstract class may have abstract and concrete methods.  
  
Ex:  
  
Class1 (having 10 methods)  
  
10 methods are concrete methods  
----------  
Class2 (having 10 methods)  
  
5 methods concrete  
5 methods Abstract  
  
\* Abstract class  
----------------------  
Class3 (having 10 methods)  
  
10 methods Abstract  
\* Abstract class  
------------------------------  
Example for Abstract Class:  
--------------------------  
Syntax:  
  
public abstract class ClassName{  
  
public void methodName(){  
//Body  
}  
  
public abstract void methodname();   
  
}  
-----------------------------  
Super Class  
package JavaOOPS;   
  
public abstract class Bike {  
public void engine(){  
    System.out.println("Bikes have Engine");  
}  
public abstract void handle();  
  
public abstract void seat();  
}  
//--------------------------------  
//Sub Class:  
  
  
public class HeroHonda extends Bike{  
  
    @Override  
    public void handle() {  
        System.out.println("Bikes have Handle");  
        }  
  
    @Override  
    public void seat() {  
        System.out.println("Bikes have Seats");  
        }  
    public static void main (String [] args){  
        HeroHonda obj = new HeroHonda();  
        obj.engine();  
        obj.handle();  
        obj.seat();  
    }  
}  
------------------------------------------------  
**iv) Encapsulation:**  
It is a process of wrapping code and data together into a single unit.  
  
Encapsulation is the technique making the fields in a class private and providing access via public methods.  
  
Advantages:  
  
> It provides us the control over the Data.  
  
> By providing only setter and getter methods, we can make a class read only or write only.  
  
If we don't define setter method then read only  
  
If we don't define getter method then write only.  
---------------------------------------------