Pillars of OOPS
 a. Datahiding   -> achieved using private access modifier
 b. Abstraction   -> achieved using abstract and interfaces
 c. Encapuslation -> Datahiding + abstraction
 d. Polymorphism -> achieved in inheritance(Static, Dynamic polymorphism)
++
Polymorphism
      Poly   -> Many
      morphism -> forms

 a. static polymorphism
            eg: Method Overloading, Method Hiding
 b. dynamic polymorphism
            eg: Method Overriding

MethodOverloading
+++++++++++++++++
  Two methods is said to be overloaded,iff both the methods have same name but different argument types.
  In case of methodOverloading, Compiler will bind the call of the method to the body of the method.
  JVM should just execute the method body, so we say MethodOverloading as "CompileTimeBinding/EarlyBinding".

eg#1.
```
class Calculator
{
      public void add(int a,int b){
            System.out.println("int-int argument");
      }
      public void add(float a,float b){
            System.out.println("float-float argument");
      }
      public void add(double a,double b){
            System.out.println("double-double argument");
      }
}
class Test
{
      public static void main(String[] args)
      {
            Calculator c = new Calculator();
            c.add(10,20); //int-int
            c.add(10.5f,20.5f);//float-float
            c.add(10.0,20.0);//double-double
      }
}
```
Output
```
D:\Decode Java1.0Batch>javac Test.java

D:\Decode Java1.0Batch>java Test
int-int argument
float-float argument
double-double argument
```

Case1: Automatic type promotion in Overloading

eg#1.

```
class Calculator
{
      public void add(int a){
            System.out.println("int argument");
      }
      public void add(float a){
            System.out.println("float argument");
      }
}

class Test
{
      public static void main(String[] args)
      {
            Calculator c = new Calculator();
            c.add('a');//char----> char,int
            c.add(19L);//long----> long,float,double

            //CE: no suitable method found
            c.add(10.5);//double -->double
      }
}
```

Case2: Ambigous method call CompileTime Error

Eg#1.

```
class Calculator
{
      public void add(int a,float b){
            System.out.println("int-float argument");
      }
      public void add(float a,int b){
            System.out.println("float-int argument");
      }
}
class Test
{
      public static void main(String[] args)
      {
            Calculator c = new Calculator();
            c.add(10,25.5f);//int,float
            c.add(20.5f,10);//float,int
            c.add(10,20);//CE: ambigous method call
      }
}
```

Output

```
D:\Decode Java1.0Batch>javac Test.java

D:\Decode Java1.0Batch>java Test
int-float argument
float-int argument
```

Case3:

```
class Sample
{
      public void methodOne(String s){
            System.out.println("String version...");
      }
```

```
	public void methodOne(Object o){
		System.out.println("Object version...");
	}
}

class Test
{
	public static void main(String[] args)
	{
		Sample s =new Sample();
		s.methodOne("sachin");//String     --> String
		s.methodOne(new Object());//Object --> Object
		s.methodOne(null);// null ---> String(reference), Object(reference)
	}
}
Output
D:\Decode Java1.0Batch>javac Test.java
D:\Decode Java1.0Batch>java Test
String version...
Object version...
String version...


Case4:
class Sample
{
	public void methodOne(String s){
		System.out.println("String version...");
	}

	public void methodOne(StringBuffer o){
		System.out.println("StringBuffer version...");
	}
	public void methodOne(Object o){
		System.out.println("Object version...");
	}
}

class Test
{
	public static void main(String[] args)
	{
		Sample s =new Sample();
		s.methodOne(new String("sachin"));//String
		s.methodOne(new StringBuffer("sachin"));//StringBuffer
		s.methodOne(null);//CE: ambigous call
	}
}

Case5:In case of methodoverloading,compiler will bind the method call based on the
reference type but not on the runtime object

class Animal{}
class Monkey extends Animal
{
}
class AnimalApp
{
```

```
	public void m1(Monkey m){
		System.out.println("Monkey version...");
	}
	public void m1(Animal a){
		System.out.println("Animal version...");
	}
}
class Test
{
	public static void main(String[] args)
	{
		AnimalApp a = new AnimalApp();

		Monkey m = new Monkey();
		a.m1(m);//m(Monkey) ----> Monkey

		Animal animal = new Animal();
		a.m1(animal);//animal ---> Animal

		Animal an = new Monkey();
		a.m1(an);//an(Animal) ----> Animal

	}
}
Output
D:\Decode Java1.0Batch>javac Test.java

D:\Decode Java1.0Batch>java Test
Monkey version...
Animal version...
Animal version...

Var-args in java
++++++++++++++++
 => This mechanism is available in java from JDK1.5V
 => In case of var-args all the arugments should be of same datatype
 => U can call var-args by passing arguments from 0....n

eg#1.
class Calculator
{
	//Method Overloading : same argument type,but different argument count
	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(int a,int b,int c,int d){
		System.out.println(a+b+c+d);
	}
	public void add(int a,int b,int c,int d,int e){
		System.out.println(a+b+c+d+e);
	}
}
class AdvancedCalculator{
	//Var-Args:: 0 to n
	public void add(int... args){
		int sum = 0;
```

```
            for (int data : args )
            {
                  sum+=data;
            }
            System.out.println(sum);
      }
}

class Test {
      public static void main(String[] args) {
            Calculator c = new Calculator();
            c.add(10,20);
            c.add(10,20,30);
            c.add(10,20,30,40);
            c.add(10,20,30,40,50);

            System.out.println();

            AdvancedCalculator ac = new AdvancedCalculator();
            ac.add();
            ac.add(10);
            ac.add(10,20);
            ac.add(10,20,30);
            ac.add(10,20,30,40);
            ac.add(10,20,30,40,50);
      }
}
Output
D:\Decode Java1.0Batch>javac Test.java

D:\Decode Java1.0Batch>java Test
30
60
100
150

0
10
30
60
100
150

Var-Arg vs Overloaded method
++++++++++++++++++++++++++
class Demo
{
      //Exact Match : One-Argument
      public void methodOne(int i){
            System.out.println("General method");
      }

      //Var-Args : 0 .... n
      public void methodOne(int... i){
            System.out.println("Var-Arg method");
      }
}
class Test {
```

```
	public static void main(String[] args) {
		Demo d1 = new Demo();
		d1.methodOne();//Var-Arg
		d1.methodOne(10);//General method
		d1.methodOne(10,20);//Var-Arg method
	}
}
Output
D:\Decode Java1.0Batch>javac Test.java
D:\Decode Java1.0Batch>java Test
Var-Arg method
General method
Var-Arg method
```