Dynamic Polymorphism :
---
* The polymorphism which exist at runtime is called Dynamic Polymorphism.

* In Dynamic Polymorphism, Compiler does not have any idea regarding method call, At runtime JVM will decide which method is invoked based on the **Object type.**

* The binding of the method is done at runtime by the JVM by using Object type so, It is also known as Late Binding.

* We can achieve Dynamic Polymorphism by using "Method Overriding".

Method Overloading :
---
* It allows us to write two or more methods in the same class as well as in the **super and sub class** in such a way that **method name must be same but differ in :**

a) Number of Parameters [Example 3]
b) Data type of the parameter [Example 1 and Example 2]
c) Order of the parameter (int - long and long - int) [Example 4]

Example 1 :

```
public void makePayment(Cash cash)
{
}

public void makePayment(CreditCard card)
{
}

public void makePayment(UPI upi)
{
}
```

Example 2 :

```
public void unlockPhone(Pattern pattern)
{
}

public void unlockPhone(PIN pin)
{
}

public void unlockPhone(Face face)
{
}
```

Example 3:

```
public void add(int x)
{
}
public void add(int x, int y)
{
}
```

Example 4 :

```
public void add(int x, long y)
{
}

public void add(long x, int y)
{
}
```

* While overloading a method, We can change the return type of the method.

* If we only change the return type but parameters are same then **it is not a** valid Method overload.

* Method overloading is possible in the same class as well as with super and sub class.

Advantage of Method Overloading :
---
* With MOL we can provide **different functionality with same method name,** Actually it is the refinement of the method.
* The best example of method overloading from Java API (Application Programming interface) is **println() method of** System.out.println() statement.

IQ :

Can we overload static method OR main method ?
---
* We can overload static method.
* We can also overload main (static method) method but JVM always execute the main method which accept String array as a parameter.

//Programs :

```
package com.ravi.overloading;

class Addition
{
    {
        System.out.println("NSB");
    }


    public Addition(int x, int y)
    {
        this(2.3F, 8.9F);
        System.out.println("Sum of two integer is :"+(x+y));
    }

    public Addition(float x, float y)
    {
        this("Data","base");
        System.out.println("Sum of two Float is :"+(x+y));
    }

    public Addition(String  x, String y)
    {
        System.out.println("Concatenation is :"+x.concat(y));
    }
}
public class ConstructorOverloading
{
    public static void main(String[] args)
    {
        new Addition(100, 200);
    }

}
```

//Program on Method Overloading by changing the return type :

```
package com.ravi.overloading;

class Sum
{
    public int add(int x, int y, int z)
    {
        return x+y+z;
    }

    public Double add(Double x, Double y)
    {
        return x+y;
    }
}

public class Overloading
{
    public static void main(String[] args)
    {
        Sum s1 = new Sum();
        System.out.println("Sum of two integer is :"+s1.add(100, 200, 300));
        System.out.println("Sum of two double is :"+s1.add(5D, 5d));

    }

}
```

Var-Args in java :
---
* It is a new concept introduced from JDK 1.5V.
* It is represented by exactly ... (3 dots).
* By using var args we can accept 0 to n number of arguments of same type OR different type (by using Object class)

* Actually Internally it is an array variable that is the reason It can accept 0 to n number of arguments