Method Overloading in Java

- **What is method overloading?**

Method overloading is a concept that allows to declare multiple methods with same name but different parameters in the same class.

Java supports method overloading and always occur in the same class(unlike method overriding).

Method overloading can be done by changing number of arguments or by changing the data type of arguments.

If two or more methods have same name and same parameter list but differs in return type can't be overloaded.

Note: Overloaded method can have different access modifiers and it does not have any significance in method overloading.

There are two different ways of method overloading.

Different datatype of arguments

Different number of arguments

- **Method overloading by changing data type of arguments.**

Example:

In this example, we have two sum() methods that take integer and float type arguments respectively.

Notice that in the same class we have two methods with same name but different types of parameters.

```
class Calculate

{

 void sum (int a, int b)

 {

  System.out.println("sum is"+(a+b)) ;

 }
 void sum (float a, float b)
 {
   System.out.println("sum is"+(a+b));
 }
 public static void main (String[] args)
 {
   Calculate cal = new Calculate();
   cal.sum (8,5);      //sum(int a, int b) is method is called.
   cal.sum (4.6f, 3.8f); //sum(float a, float b) is called.
 }
}
```

You can see that sum() method is overloaded two times. The first takes two integer arguments, the second takes two float arguments.

- **Method overloading by changing no. of argument.**

Example:

In this example, we have two methods

```
class Demo
{
 void multiply(int l, int b)
 {
   System.out.println("Result is"+(l*b)) ;
 }
 void multiply(int l, int b,int h)
 {
   System.out.println("Result is"+(l*b*h));
 }
 public static void main(String[] args)
 {
   Demo  ar = new Demo();
   ar.multiply(8,5);   //multiply(int l, int b) is method is called
   ar.multiply(4,6,2);   //multiply(int l, int b,int h) is called
 }
}
```

In this example the multiply() method is overloaded twice. The first method takes two arguments and the second method takes three arguments.

When an overloaded method is called Java look for match between the arguments to call the method and then it's parameters. This match need not always be exact, sometime when exact match is not found, Java automatic type conversion plays a vital role.

- **Example of Method overloading with type promotion.**

Java supports automatic type promotion, like int to long or float to double etc. In this example we are doing same and calling a function that takes one integer and second long type argument. At the time of calling we passed integer values and Java treated second argument as long type. See the below example.

```
class Demo
{
 void sum(int l,long b)
 {
  System.out.println("Sum is"+(l+b)) ;
 }
 void sum(int l, int b, int h)
 {
```

```
 System.out.println("Sum is"+(l+b+h));
 }
 public static void main (String[] args)
 {
  Demo  ar = new Demo();
  ar.sum(8,5);
 }
}
```

- **Method overloading if the order of parameters is changed**

We can have two methods with same name and parameters but the order of parameters is different.

Example:

In this scenario, method overloading works but internally JVM treat it as method having different type of arguments.

```
class Demo{
    public void get(String name,  int id)
   {
System.out.println("Company Name :"+ name);
  System.out.println("Company Id :"+ id);
    }
    public void get(int id, String name)
   {
System.out.println("Company Id :"+ id);
  System.out.println("Company Name :"+ name);
    }
}
```

```
class MethodDemo8{
    public static void main (String[] args) {
  Demo obj = new Demo();
  obj.get("Cherry", 1);
  obj.get("Jhon", 2);
    }
}
```

- **Overloading main Method**

In Java, we can overload the main() method using different number and types of parameter but the JVM only understand the original main() method.

Example:

In this example, we created three main() methods having different parameter types.

```
class MethodDemo10{
    public static void main(int args)
    {
System.out.println("Main Method with int argument Executing");
  System.out.println(args);
    }
    public static void main(char args)
    {
System.out.println("Main Method with char argument Executing");
  System.out.println(args);
    }
    public static void main(Double[] args)
    {
System.out.println("Main Method with Double Executing");
```

```
  System.out.println(args);
    }
    public static void main(String[] args)
    {
System.out.println("Original main Executing");
  MethodDemo10.main(12);
  MethodDemo10.main('c');
  MethodDemo10.main(1236);
    }
}
```

- **Method overloading and null error**

This is a general issue when working with objects, if same name methods having reference type parameters, then be careful while passing arguments.

Below is an example in which you will know how a null value can cause an error when methods are overloaded.

Example:

Null value is a default value for all the reference types. It created ambiguity to JVM that reports error.

```
class Demo
{
    public void test(Integer i)
    {
System.out.println("test(Integer ) ");
    }
    public void test(String name)
    {
```

```
System.out.println("test(String ) ");
   }
   public static void main(String [] args)
   {
      Demo obj = new Demo();
obj.test(null);
   }
}
```

Output: The method test(Integer) is ambiguous for the type Demo.

Reason:

The main reason for getting the compile time error in the above example is that here we have Integer and String as arguments which are not primitive data types in java and this type of argument do not accept any null value. When the null value is passed the compiler gets confused which method is to be selected as both the methods in the above example are accepting null.

However, we can solve this to pass specific reference type rather than value.

Example:

In this example, we are passing specific type argument rather than null value.

```
class MethodDemo9
{
   public void test(Integer i)
   {
System.out.println("Method ==> test(Integer)");
   }
   public void test(String name)
   {
System.out.println("Method ==> test(String) ");
```

```
    }
    public static void main(String [] args)
    {
        MethodDemo9 obj = new MethodDemo9 ();
  Integer a = null;
obj.test(a);
  String b = null;
obj.test(b);
    }
}
```