**Method Overloading in Java with examples**

Method Overloading is a feature that allows a class to have two or more methods having same name, if their argument lists are different. In the last tutorial we discussed [**constructor overloading**](http://beginnersbook.com/2013/05/constructor-overloading/) that allows a class to have more than one constructors having different argument lists.

**Argument lists could differ in –**  
1. Number of parameters.  
2. Data type of parameters.  
3. Sequence of Data type of parameters.

**Method overloading** is also known as **Static Polymorphism**.

**Points to Note:**  
1. [**Static Polymorphism**](http://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/) is also known as compile time binding or early binding.  
2. [**Static binding**](http://beginnersbook.com/2013/04/java-static-dynamic-binding/) happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

**Method Overloading examples:**

As discussed above, method overloading can be done by having different argument list. Lets see examples of each and every case.

**Example 1: Overloading – Different Number of parameters in argument list**

When methods name are same but number of arguments are different.

class DisplayOverloading

{

public void disp(char c)

{

System.out.println(c);

}

public void disp(char c, int num)

{

System.out.println(c + " "+num);

}

}

class Sample

{

public static void main(String args[])

{

DisplayOverloading obj = new DisplayOverloading();

obj.disp('a');

obj.disp('a',10);

}

}

**Output:**

a

a 10

In the above example – method disp() has been overloaded based on the number of arguments – We have two definition of method disp(), one with one argument and another with two arguments.

**Example 2: Overloading – Difference in data type of arguments**

In this example, method disp() is overloaded based on the data type of arguments – Like example 1 here also, we have two definition of method disp(), one with char argument and another with int argument.

class DisplayOverloading2

{

public void disp(char c)

{

System.out.println(c);

}

public void disp(int c)

{

System.out.println(c );

}

}

class Sample2

{

public static void main(String args[])

{

DisplayOverloading2 obj = new DisplayOverloading2();

obj.disp('a');

obj.disp(5);

}

}

Output:

a

5

**Example3: Overloading – Sequence of data type of arguments**

Here method disp() is overloaded based on sequence of data type of arguments – Both the methods have different sequence of data type in argument list. First method is having argument list as (char, int) and second is having (int, char). Since the sequence is different, the method can be overloaded without any issues.

class DisplayOverloading3

{

public void disp(char c, int num)

{

System.out.println("I’m the first definition of method disp");

}

public void disp(int num, char c)

{

System.out.println("I’m the second definition of method disp" );

}

}

class Sample3

{

public static void main(String args[])

{

DisplayOverloading3 obj = new DisplayOverloading3();

obj.disp('x', 51 );

obj.disp(52, 'y');

}

}

**Output:**

I’m the first definition of method disp

I’m the second definition of method disp

**Lets see few Valid/invalid cases of method overloading**

Case 1:

int mymethod(int a, int b, float c)

int mymethod(int var1, int var2, float var3)

Result: Compile time error. Argument lists are exactly same. Both methods are having same number, data types and same sequence of data types in arguments.

Case 2:

int mymethod(int a, int b)

int mymethod(float var1, float var2)

Result: Perfectly fine. Valid case for overloading. Here data types of arguments are different.

Case 3:

int mymethod(int a, int b)

int mymethod(int num)

Result: Perfectly fine. Valid case for overloading. Here number of arguments are different.

Case 4:

float mymethod(int a, float b)

float mymethod(float var1, int var2)

Result: Perfectly fine. Valid case for overloading. Sequence of the data types are different, first method is having (int, float) and second is having (float, int).

Case 5:

int mymethod(int a, int b)

float mymethod(int var1, int var2)

Result: Compile time error. Argument lists are exactly same. Even though return type of methods are different, it is not a valid case. Since return type of method doesn’t matter while overloading a method.

Guess the answers before checking it at the end of programs:  
**Question 1 – return type, method name and argument list same.**

class Demo

{

public int myMethod(int num1, int num2)

{

System.out.println("First myMethod of class Demo");

return num1+num2;

}

public int myMethod(int var1, int var2)

{

System.out.println("Second myMethod of class Demo");

return var1-var2;

}

}

class Sample4

{

public static void main(String args[])

{

Demo obj1= new Demo();

obj1.myMethod(10,10);

obj1.myMethod(20,12);

}

}

**Answer:**  
It will throw a compilation error: More than one method with same name and argument list cannot be defined in a same class.

**Question 2 – return type is different. Method name & argument list same.**

class Demo2

{

public double myMethod(int num1, int num2)

{

System.out.println("First myMethod of class Demo");

return num1+num2;

}

public int myMethod(int var1, int var2)

{

System.out.println("Second myMethod of class Demo");

return var1-var2;

}

}

class Sample5

{

public static void main(String args[])

{

Demo2 obj2= new Demo2();

obj2.myMethod(10,10);

obj2.myMethod(20,12);

}

}

**Answer:**  
It will throw a compilation error: More than one method with same name and argument list cannot be given in a class even though their return type is different. **Method return type doesn’t matter in case of overloading.**