**Method Overloading in Java**

Whenever same method name is exiting multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as **method overloading**.

**Why use method Overloading in Java ?**

Suppose we have to perform addition of given number but there can be any number of arguments, if we write method such as a(int, int)for two arguments, b(int, int, int) for three arguments then it is very difficult for you and other programmer to understand purpose or behaviors of method they can not identify purpose of method. So we use method overloading to easily figure out the program. For example above two methods we can write sum(int, int) and sum(int, int, int) using method overloading concept.

**Syntax**

class class\_Name

{

Returntype method()

{.........}

Returntype method(datatype1 variable1)

{.........}

Returntype method(datatype1 variable1, datatype2 variable2)

{.........}

Returntype method(datatype2 variable2)

{.........}

Returntype method(datatype2 variable2, datatype1 variable1)

{.........}

}

**Different ways to overload the method**

There are two ways to overload the method in java

* By changing number of arguments or parameters
* By changing the data type

**By changing number of arguments**

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

**Example Method Overloading in Java**

**class** Addition

{

**void** sum(**int** a, **int** b)

{

System.**out**.println(a+b);

}

**void** sum(**int** a, **int** b, **int** c)

{

System.**out**.println(a+b+c);

}

**public** **static** **void** main(String args[])

{

Addition obj=**new** Addition();

obj.sum(10, 20);

obj.sum(10, 20, 30);

}

}

**Output**

30

60

**By changing the data type**

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two float arguments.

**Example Method Overloading in Java**

**class** Addition

{

**void** sum(**int** a, **int** b)

{

System.**out**.println(a+b);

}

**void** sum(**float** a, **float** b)

{

System.**out**.println(a+b);

}

**public** **static** **void** main(String args[])

{

Addition obj=**new** Addition();

obj.sum(10, 20);

obj.sum(10.05, 15.20);

}

}

**Output**

30

25.25

**Why Method Overloading is not possible by changing the return type of method?**

In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity. Let's see how ambiguity may occur:

because there was problem:

**Example of Method Overloading**

**class** Addition

{

**int** sum(**int** a, **int** b)

{

System.**out**.println(a+b);

}

**double** sum(**int** a, **int** b)

{

System.**out**.println(a+b);

}

**public** **static** **void** main(String args[])

{

Addition obj=**new** Addition();

**int** result=obj.sum(20,20); //Compile Time Error

}

}

**Explanation of Code**

**Example**

int result=obj.sum(20,20);

Here how can java determine which sum() method should be called

**Note:**The scope of overloading is within the class.

Any object reference of class can call any of overloaded method.

**Can we overload main() method ?**

Yes, We can overload main() method. A Java class can have any number of main() methods. But run the java program, which class should have main() method with signature as "public static void main(String[] args). If you do any modification to this signature, compilation will be successful. But, not run the java program. we will get the run time error as main method not found.

**Example of override main() method**

**Example**

**public** **class** mainclass

{

**public** **static** **void** main(String[] args)

{

System.**out**.println("Execution starts from Main()");

}

**void** main(**int** args)

{

System.**out**.println("Override main()");

}

**double** main(**int** i, **double** d)

{

System.**out**.println("Override main()");

**return** d;

}

}

**Outpot**

Execution starts from Main()