Method Overloading in Java

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

- Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int, int) for two parameters, and b(int, int ,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.
- So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

- Method overloading *increases the readability of the program*.

Different ways to overload the method

- By changing number of arguments
- By changing the data type

Method Overloading: changing no. of arguments

```
class Adder
{
static int add(int a, int b)
{
return a+b;
}
static int add(int a, int b, int c)
{
return a+b+c;}
}
```

```
class TestOverloading1
{
public static void main(String[] args)
{
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}
}
```

- we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

Method Overloading: changing data type of arguments

- **class** Adder
- {
- **static int** add(**int** a, **int** b)
- {**return** a+b;}
- **static double** add(**double** a, **double** b)
- {**return** a+b;}
- }

```
class TestOverloading2
{
public static void main(String[] args)
{
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}
}
```

- we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

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

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

```
class Adder
{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}}
```

Compile Time Error: method add(int, int) is already defined in class Adder

System.out.println(Adder.add(11,11)); //Here, how can java determine which sum() method should be called?

Can we overload java main() method?

- Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

```
class TestOverloading4
{
public static void main(String[] args)
{System.out.println("main with String[]");}
public static void main(String args)
{System.out.println("main with String");}
public static void main()
{System.out.println("main without args");}
}
```

Method Overriding in Java

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.
- In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

- method must have same name as in the parent class
- method must have same parameter as in the parent class.
- must be IS-A relationship (inheritance).

Understanding the problem without method overriding

- **class** Vehicle{
- **void** run(){System.out.println("Vehicle is runn ing");} }
- **class** Bike **extends** Vehicle{
- **public static void** main(String args[]){
- Bike obj = **new** Bike();
- obj.run();
- } }

Example of method overriding

- **class** Vehicle{
- **void** run(){System.out.println("Vehicle is running");} }
- **class** Bike2 **extends** Vehicle{
- **void** run(){System.out.println("Bike is running safely");}

- **public static void** main(String args[]){
- Bike2 obj = **new** Bike2();
- obj.run();
- }

super keyword in java

- The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

Usage of java super Keyword

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

super is used to refer immediate parent class instance variable.

- We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

- **lass** Animal{
- String color="white";
- }
- **class** Dog **extends** Animal{
- String color="black";
- **void** printColor(){
- System.out.println(color);//prints color of Dog class
- System.out.println(**super**.color);//prints color of Animal class
- } }

```
class TestSuper1
{
public static void main(String args[])
{
Dog d=new Dog();
d.printColor();
}}
```

super can be used to invoke parent class method

- The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
} }
```

```
class TestSuper2
{
public static void main(String args[])
{
Dog d=new Dog();
d.work();
}
}
```

super is used to invoke parent class constructor.

- **class** Animal{
- Animal(){System.out.println("animal is created");}
- }
- **class** Dog **extends** Animal{
- Dog(){
- **super**();
- System.out.println("dog is created");
- }
- }

- **class** TestSuper3
- {
- **public static void** main(String args[])
- {
- Dog d=**new** Dog();
- }
- }