Method overloading

Method Overloading is a feature that allows a class to have **more than one method having the same name**, if their **argument lists are different**. When I say argument list it means the parameters that a method has: For example the argument list of a method **add(int a, int b)** having two parameters is different from the argument list of the method **add(int a, int b, int c)** having three parameters.

There are three ways to overload a method:

In order to overload a method, the argument lists of the methods must differ in either of these:

1. Number of parameters.

For example: This is a valid case of overloading

- area(int, int)
- area(int, int, int)

2. Data type of parameters.

For example:

- area(int, float)
- area(int, int)

3. Sequence of Data type of parameters.

For example:

- area(int, float)
- area(float, int)

Invalid case of method overloading:

When I say argument list, I am not talking about **return type of the method**, for example if two methods have **same name**, **same parameters** and have **different return type**, then this is **not a valid method overloading** example. This will throw **compilation error**.

For example: **int area(int, int)**

float area(int, int)

Method overloading is an example of **Static Polymorphism**. We will look into polymorphism in deep in further classes.

- Static Polymorphism is also known as **compile time binding or early binding**.
- Static 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.

Now let's see through code..

We are trying to find the area of rectangle with all possible inputs. And just to demonstrate the different number of parameters rule, 3rd parameter is included.


```
class Rectangle
{
    int area(int l,int b)
    {
        return l*b;
    }
    float area(int l,float b) //data types are different
    {
        return l*b;
    }
    float area(float l,int b) // order is different
    {
        return l*b;
    }
    double area(int l,double b)
    {
        return l*b;
    }
    double area(float l,double b)
    {
        return l*b;
    }
    double area(double l,double b)
    {
        return l*b;
    }
    float area(int l,float b,int h) //number of parameters differ
    {
        return l*b*h;
    }
}
```

```
class Demo
{
    public static void main(String[] args)
    {
        int a=10,b=20;
        float m=30.3f,n=40.4f;
        double p=50.5,q=60.6;
        Rectangle ref = new Rectangle();
        System.out.println(ref.area(a,b));
        System.out.println(ref.area(m,b));
        System.out.println(ref.area(a,n));
        System.out.println(ref.area(a,n,b));

    }
}
```

Output:

```
200
606.0
404.0
8080.0
```

Method overloading and Type Promotion

When a **data type** of **smaller size** is **promoted to the data type of bigger size** than this is called **type promotion**, for example: byte data type can be promoted to short, a short data type can be promoted to int, long, double etc.

What it has to do with method overloading?

Well, it is very important to understand type promotion else you will think that the program will throw compilation error but in fact that program will run fine because of type promotion.

Let's take an example to see what I am talking here:

```java
class Rectangle
{

    float area(int l,float b) //closest match
    {
        return l*b;
    }
    double area(double l,double b)
    {
        return l*b;
    }
}
class Demo
{
    public static void main(String[] args)
    {
        Rectangle ref = new Rectangle();
        int a=10,b=20;
        System.out.println(ref.area(a,b));
    }
}
```

Output: **200.0**

Type Promotion table:

The **data type** on the **left side** can be **promoted to** the any of the data type present in the **right side** of it.

```
byte → short → int → long
short → int → long
int → long → float → double
float → double
long → float → double
```