

Method Overloading in Oops

PREVIEW / RECAP:

We have seen how to create or call a method, static/non-static methods. Now let's understand the concept of Method Overloading

What is 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, 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**.

Advantage of method overloading

- Method overloading increases the readability of the program.

Different ways to overload the method

There are two ways to overload the method in java

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

Note: In java, Method Overloading is not possible by changing the return type of the method only.

1) Method Overloading: changing no. of arguments

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

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
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));
    }
}
```

2) Method Overloading: changing data type of arguments

In this example, 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.

```
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));
    }
}
```

Invalid case of method overloading:

If two methods have the same name, same parameters and have different return types, then this is not a valid method overloading example. This will throw a compilation error.

E.g.

```
int add(int, int)
float add(int, int)
```

Method overloading is an example of Static Polymorphism.

POLYMORPHISM CONCEPT IS EXPLAINED BRIEFLY UNDER POLYMORPHISM TOPIC

Points to Note:

1. Static Polymorphism is also known as compile time binding or early binding.
2. 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 we have a basic idea of what is method overloading and what is use of it, but there is one more concept of Type Promotion in Method Overloading. Let's have a look at it...

Method Overloading and Type Promotion

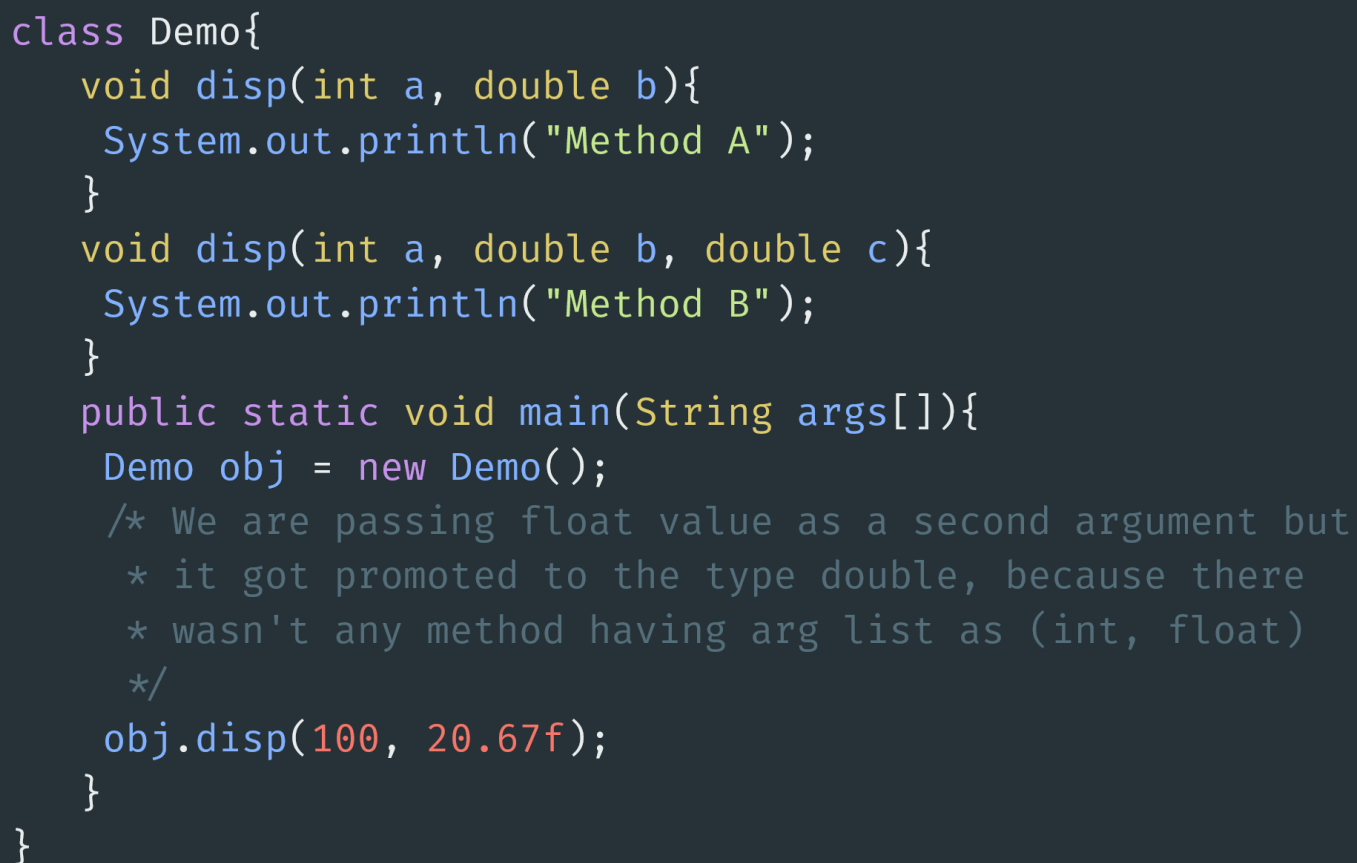
What is Type Promotion?

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 we are talking here:

E.g.



```
class Demo{
    void disp(int a, double b){
        System.out.println("Method A");
    }
    void disp(int a, double b, double c){
        System.out.println("Method B");
    }
    public static void main(String args[]){
        Demo obj = new Demo();
        /* We are passing float value as a second argument but
        * it got promoted to the type double, because there
        * wasn't any method having arg list as (int, float)
        */
        obj.disp(100, 20.67f);
    }
}
```

Output:

Method A

As you can see that we have passed the float value while calling the disp() method but it got promoted to the double type as there wasn't any method with argument list as (int, float)

But this type promotion doesn't always happen, let's see another example:

E.g.

```
class Demo{
    void disp(int a, double b){
        System.out.println("Method A");
    }
    void disp(int a, double b, double c){
        System.out.println("Method B");
    }
    void disp(int a, float b){
        System.out.println("Method C");
    }
    public static void main(String args[]){
        Demo obj = new Demo();
        /* This time promotion won't happen as there is
         * a method with arg list as (int, float)
         */
        obj.disp(100, 20.67f);
    }
}
```

Output:

Method C

As you see that this time type promotion didn't happen because there was a method with matching argument type.

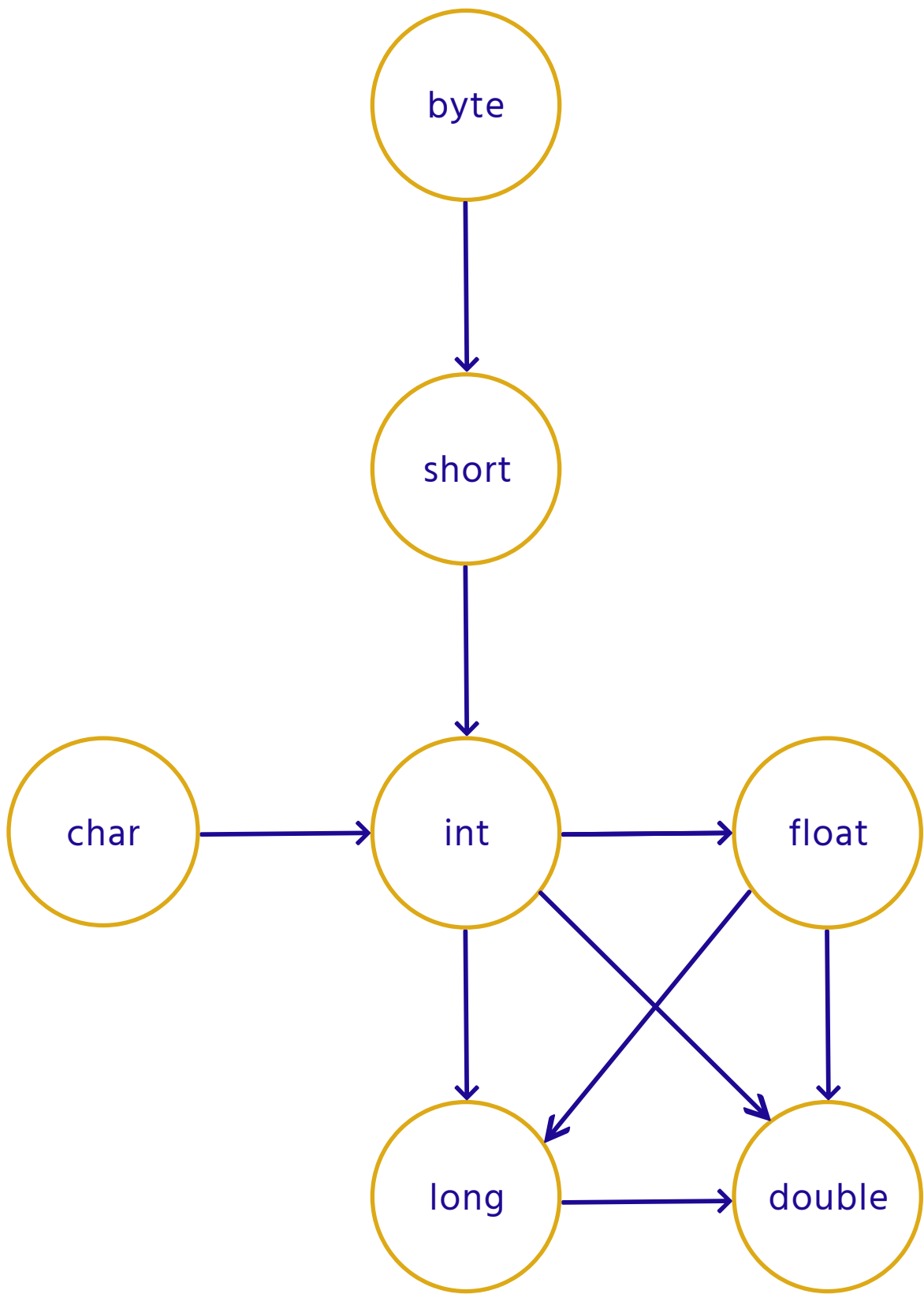
Type Promotion table:

The data type on the left side can be promoted to 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


Note: One type is not de-promoted implicitly for example double cannot be depromoted to any type implicitly.

Let’s have a look at pictorial representation of this type promotion.



Example of Method Overloading with Type Promotion in case of ambiguity

If there are no matching type arguments in the method, and each method promotes a similar number of arguments, there will be ambiguity.



```
class OverloadingCalculation3{
    void sum(int a,long b){System.out.println("a method invoked");}
    void sum(long a,int b){System.out.println("b method invoked");}

    public static void main(String args[]){
        OverloadingCalculation3 obj=new
        OverloadingCalculation3();
        obj.sum(20,20);//now ambiguity
    }
}
```

Output:

Compile Time Error