

Method Signature

A method signature consists only of the name of the method, the parameter types, and their order. The modifiers, return type, and throws clause are not part of the signature.

`public static int add(int i, int j)` → called method declaration

`int add(int i,int j)` // method prototype

`add(int,int)`→**Method Signature**

| Method | Method Signature |
|--|---|
| • <code>void setData(String n,double b,int c)</code> | <code>--setData(String, double, int)</code> |
| • <code>void addNum(double d,float f)</code> | <code>—addNum(double,float)</code> |
| • <code>public void addNum(float f, double d)</code> | <code>—addNum(float,double)</code> |

class [MSignature](#)

```
{  
  
    public void meth1(int i)  
    {  
        System.out.println("method 1");  
    }  
  
    public void meth2(String s)  
    {  
        System.out.println("method 2");  
    }  
    public static void main(String args[])  
    {  
        MSignature m=new MSignature();// created one object  
  
        m.meth1(4);// calling method ?  
  
        m.meth2("Saurabh");// calling method ?  
  
    }  
}
```

```
C:\Windows\System32\cmd.exe

F:\Java Code 2020>javac MSignature.java

F:\Java Code 2020>java MSignature
method 1
method 2
```

Compiler is going to be maintained in the method signature table.

- **meth1(int)**
- **meth2(String)**

Is there any method **meth1**, which can take int argument, present in MSignature's method table?
Answer:

Yes

Is there any method meth2 that can take the String argument, present in MSignature's method table?

Answer:

Yes

```
class MSignature
{
    public void meth1(int i)
    {
        System.out.println("method 1");
    }

    public void meth2(String s)
    {
        System.out.println("method 2");
    }

    public static void main(String args[])
    {
        MSignature m=new MSignature();
        m.meth1(4);
        m.meth2("Saurabh");
        m.meth3(10.5);//meth3(double)
    }
}
```

```
}  
}
```

m of type MSignature

Is there any meth3 method with double type argument, present in MSignature's method table?

Answer :

no

So following the compiler time error, you are going to get:

```
F:\Java Code 2020>javac MSignature.java  
MSignature.java:18: error: cannot find symbol  
        m.meth3(10.5);  
            ^  
  symbol:   method meth3(double)  
  location: variable m of type MSignature  
1 error
```

Question: Who is going to use the method signature?

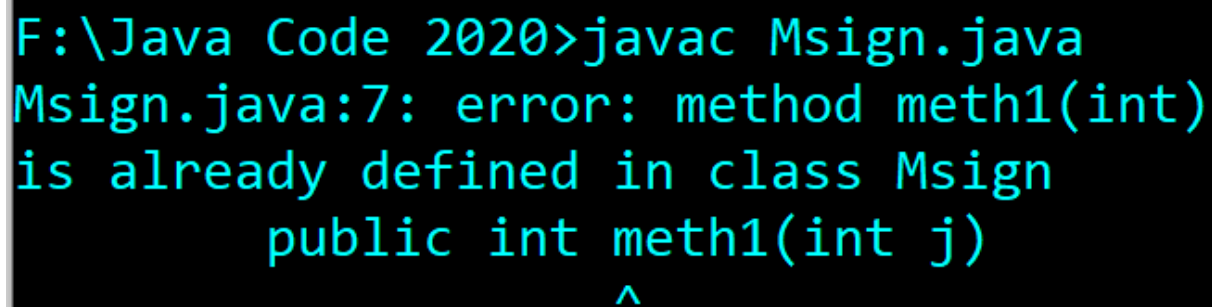
Answer: compiler

Question: When the compiler will use the method signature?

Answer: while calling/invoking methods, for resolving method calls.

```
class Msign
{
    public void meth1(int i) // meth1(int)
    {
        SOP("hello");
    }
    public int meth1(int j) //meth1(int)
    {
        return 10;
    }
}
is it valid or not???
```

Answer: Not valid.

A screenshot of a terminal window with a black background and cyan text. It shows the command 'javac Msign.java' being executed. The output is an error message: 'Msign.java:7: error: method meth1(int) is already defined in class Msign'. Below the error message, the code snippet 'public int meth1(int j)' is shown with a caret '^' pointing to the parameter 'j', indicating the conflict with the first method's parameter 'i'.

```
F:\Java Code 2020>javac Msign.java
Msign.java:7: error: method meth1(int)
is already defined in class Msign
        public int meth1(int j)
                        ^
```

Within a class, two methods with the same signature are not allowed.

Method Overloading:

Two methods are said to be overloaded if and only if both methods have the same name but different argument types.

OR

Defining a function again with the same name, in the same class, and with different arguments is known as method overloading.

If we have to perform only one operation, having same name of the methods increases the **readability of the program**.

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

```
class MOverload
{
    public void m1()
    {
        System.out.println("no args method");
    }

    public void m1(int i)
    {
        System.out.println("int args method");
    }

    public void m1( double d)
    {
        System.out.println("double args method");
    }

    public static void main(String args[])
    {
        MOverload m=new MOverload();
        m.m1();//m1()
        m.m1(5);//m1(int)
        m.m1(7.5);//m1(double)
    }
}
```

```
C:\Windows\System32\cmd.exe

F:\Java Code 2020>javac MOverload.java

F:\Java Code 2020>java MOverload
no args method
int args method
double args method
```

In overloading, the compiler always takes care of method resolution based on the reference type. Hence, overloading is also considered to compile time polymorphism, static polymorphism, and early **binding**.

Different ways to overload the method

There are two ways to overload the method in java

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

BY CHANGING THE NUMBER OF ARGUMENTS

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

```
1. class Calculation{
2.     void sum(int a, int b) //sum(int,int)
3.     {
4.         System.out.println(a+b);
5.     }
6.     void sum(int a,int b,int c)//sum(int,int,int)
7.     {
8.         System.out.println(a+b+c);
9.     }
10.
11.     public static void main(String args[]){
12.         Calculation obj=new Calculation();
13.         obj.sum(10,10,10); // sum(int,int,int)→30
14.         obj.sum(20,20); //sum(int,int) →40
15.
16.     }
```

```
17. }
```

```
Output:30
       40
```

BY CHANGING THE DATA TYPE OF THE ARGUMENT

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 double arguments.

```
1. class Calculation
2. {
3.     void sum(int a,int b)// sum(int,int)
4.     {
5.         System.out.println(a+b);
6.     }
7.     void sum(double a,double b)// sum(double,double)
8.     {
9.         System.out.println(a+b);
10.    }
11.    public static void main(String args[]){
12.        Calculation obj=new Calculation();
13.        obj.sum(10.5,10.5);//
14.        obj.sum(20,20);
15.
16.    }
17. }
```

```
Output:21.0
       40
```

```
1. class Calculation{
2.     int sum(int a,int b)//sum(int,int)----1
3.     {
4.         System.out.println(a+b);
5.     }
6.
7.     double sum(int a,int b)// sum(int,int)----2
8.     {
9.         System.out.println(a+b);
10.    }
11.
12.    public static void main(String args[]){
13.        Calculation Obj=new Calculation();
14.        Obj.sum(20,20); //
15.    }
16. }
```

int result=obj.sum(20,20); //Here how can java determine which sum() method should be called

In java, method overloading is not possible by changing the return type of the method because there may be ambiguity.

Cases:

```
class MOverCase1
{
public void m1(int i)
{
System.out.println("Int arg");
}
public void m1(float f)
{
System.out.println("Float arg");
}
public static void main(String args[])
{
MOverCase1 c=new MOverCase1();
c.m1(10);//
c.m1(20.5f);//
c.m1('a');// c.m1(97)
c.m1(10l);//long value    m1(long)

}
}
```

```
F:\Java Code 2020>javac MOverCase1.java
```

```
F:\Java Code 2020>java MOverCase1
```

```
Int arg
```

```
Float arg
```

```
Int arg
```

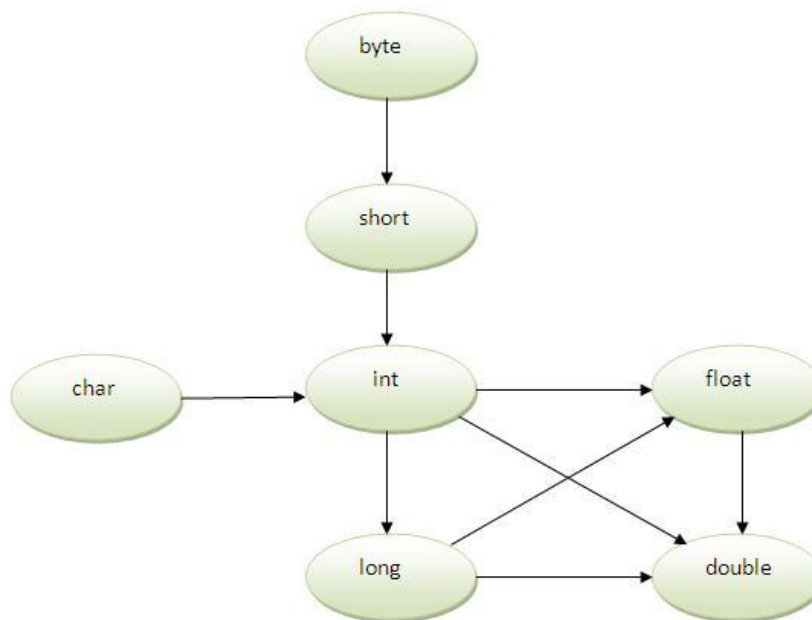
```
Float arg
```

c.m1(20.5);//Compile time error -- m1(double)

```
F:\Java Code 2020>javac MOverCase1.java
MOverCase1.java:18: error: no suitable method found for m1(double)
  c.m1(20.5);
  ^
    method MOverCase1.m1(int) is not applicable
      (argument mismatch; possible lossy conversion from double to int)
    method MOverCase1.m1(float) is not applicable
      (argument mismatch; possible lossy conversion from double to float)
1 error
```

Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found. Let's understand the concept by the figure given below:



As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int, long, float or double and so on.

EXAMPLE OF METHOD OVERLOADING WITH TYPEPROMOTION

```
1. class Calculation{
2.     void sum(int a,long b)        sum(int,long)
3.     {System.out.println(a+b);
4.     }
5.     void sum(int a,int b,int c)    sum(int,int,int)
6.     {System.out.println(a+b+c);
7.     }
8.
9.     public static void main(String args[]){
10.    Calculation obj=new Calculation();
11.    obj.sum(20,20);//sum(int,int) -not present
12.    obj.sum(20,20,20);
13.
14. }
15. }
```

Output:40
60

If there are matching type arguments in the method, type promotion is not performed.

```
1. class Calculation{
2.     void sum(int a,int b){System.out.println("int arg method invoked");}
3.     void sum(long a,long b){System.out.println("long arg method invoked");}
4.
5.     public static void main(String args[]){
6.    Calculation obj=new Calculation();
7.    obj.sum(20,20);
8.    }
9. }
```

Output:int arg method invoked

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

```
1. class Calculation{
2.     void sum(int a,long b){System.out.println("a method invoked");} // sum(int,long)
3.     void sum(long a,int b){System.out.println("b method invoked");} //sum(long,int)
4.
5.     public static void main(String args[]){
6.         Calculation obj=new Calculation();
7.         obj.sum(20,20);// sum(int,int)-not present
8.     }
9. }
```

- a) a method invoked
- b) b method invoked
- c) both a and b
- d) compile time error

Output: Compile Time Error

Case:

```
class MOverCase2
{
public void m1(String s)
{
System.out.println("String arg");
}
public void m1(Object o)
{
System.out.println("Object arg");
}
public static void main(String args[])
{
MOverCase2 c=new MOverCase2();
c.m1(new Object());
c.m1("Saurabh");
c.m1(null);
}
}
```

```
F:\Java Code 2020>javac MOverCase2.java
```

```
F:\Java Code 2020>java MOverCase2
```

```
Object arg
```

```
String arg
```

```
String arg
```

```
1  class MOverCase2
2  {
3  /*public void m1(String s)
4  {
5  System.out.println("String arg");
6  }
7  */
8  public void m1(Object o)
9  {
10 System.out.println("Object arg");
11 }
12 public static void main(String args[])
13 {
14 MOverCase2 c=new MOverCase2();
15 c.m1(new Object());
16 c.m1("Saurabh");
17 c.m1(null);
18 }
19 }
```

```
D:\Java Code 2k23>java MOverCase2
Object arg
Object arg
Object arg
```

Object class in Java

The Object class is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

