1 Methods in Java

A method is a collection of statements that perform some specific task and return result to the caller. A method can perform some specific task without returning anything. Methods allow us to reuse the code without retyping the code. In Java, every method must be part of some class which is different from languages like C, C++ and Python.

In general, method declarations has six components :

- **Modifier-**: Defines access type of the method i.e. from where it can be accessed in your application. In Java, there 4 type of the access specifiers.
  - public: accessible in all class in your application.
  - protected: accessible within the class in which it is defined and in its subclass(es)
  - private: accessible only within the class in which it is defined.
  - default (declared/defined without using any modifier) : accessible within same class and package within which its class is defined.
- **The return type** : The data type of the value returned by the the method or void if does not return a value.
- **Method Name** : the rules for field names apply to method names as well, but the convention is a little different.
- **Parameter list** : Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
- **Exception list** : The exceptions you expect by the method can throw, you can specify these exception(s).
- **Method body** : it is enclosed between braces. The code you need to be executed to perform your intended operations.

2 method types

```
1  class Test1
2  {
3    void m1() //instance method
4   {
5      System.out.println("instance m1
           method");
6   }
7   public static void main(String[] args)
8   {
9    Test1 t1= new Test1();
10   t1.m1();
11  }
12 }
```

Test1.java

```
//instance method

D:\textbackslash phani>javac Test1.java
D:\textbackslash phani>java Test1
instance m1 method
```

```
1  class Test2
2  {
3  static void m2() //static method
4   {
5    System.out.println("static m2 method");
6    }
7   public static void main(String[] args)
```

```
//static method


D:\phani>javac Test2.java

D:\phani>java Test2
static m2 method
```

```
 {

   Test2.m2();
 }
}
```

Test2.java

3 method values displaying

```
class Test3
{
 int x=100;
 void m1() //instance method
 {
    System.out.println("instance variable"+x);
  }
 public static void main(String[] args)
 {
  Test3 t1= new Test3();
   t1.x=200;
   t1.m1();
   Test3 t2= new Test3();
   t2.m1();
  }
}
```

Test3.java

```
//instance  method with instance varables

D:\ phani>javac Test3.java

D:\ phani>java Test3
instance variable 200
instance variable 100
```

```
class Test4
{
 static int x=400; //static variable
 static void m1() //static method
 {
   System.out.println("static variable "+x);
  }
 public static void main(String[] args)
 {
   Test4.m1();
   Test4.x=500;//assign value to static
       variable
   Test4.m1();
 }
}
```

Test4.java

```
//static  method with static varables


D:\ phani>javac Test4.java

D:\ phani>java Test4
static variable 400
static variable 500
```

duplicate methods are not allowed

inner methods are not allowed in java

4 Method signature

In Java, a method signature is part of the method declaration. It's the combination of the method name and the parameter list. The reason for the emphasis on just the method name and parameter list is because of overloading. It's the ability to write methods that have the same name but accept different parameters.

```
public void setMapReference(int xCoordinate, int yCoordinate)
{
  //method code
}
```

The method signature in the above example is `setMapReference(int, int)`. In other words, it's the method name and the parameter list of two integers.

```
1  class Test5
2  {
3      void m1() //instance method
4   {
5    System.out.println("instance variable ");
6
7    }
8      void m1() //instance method
9   {
10   System.out.println("instance variable ");
11
12   }
13  public static void main(String[] args)
14  {
15      Test5 t5=new Test5();
16       t5.m1();
17  }
18 }
```

Test5.java

```
//methods with same name or duplication is not allowed

D:\ phani>javac Test5.java
Test5.java:8: error: method m1() is already defined in class Test5
    void m1() //instance method
         ^
1 error
```

```
1  class Test6
2  {
3      void m1()
4   {
5    System.out.println("instance variable ");
6
7    }
8     static void m1()
9   {
10   System.out.println("instance variable ");
11
12   }
13  public static void main(String[] args)
14  {
15 Test6 t6=new Test6();
16    t6.m1();
17 Test6.m1();
18  }
19 }
```

Test6.java

```
//methods with same name with different modifiers also not allowed

D:\ phani>javac Test6.java
Test6.java:8: error: method m1() is already defined in class Test6
   static void m1()
               ^
Test6.java:17: error: non-static method m1() cannot be referenced fro
context
Test6.m1();
     ^
```

5 method parameters passsing

```
class Test7
{
   int a=123;
  void m1(int x)
 {
   System.out.println("local value "+x);
   System.out.println("instance value "+a);
   a=x;
  System.out.println("instance value after "+a)
      ;
  }

 public static void main(String[] args)
 {
  Test7 t7=new Test7();
  t7.m1(456);
 }
}
```

Test7.java

```
//parameters passing
D:\ phani>javac Test7.java

D:\ phani>java Test7
local value 456
instance value 123
instance value after 456
```

```
class Test8
{
 static int p=321;
 static void m1(int k)
 {
   System.out.println("local value "+k);
   System.out.println("static value "+p);
   p=k;
  System.out.println("after passing local value
       "+p);
  }

 public static void main(String[] args)
 {
  Test8 t8=new Test8();
  t8.m1(654);
 }
}
```

Test8.java

```
//parameters passing

D:\ phani>javac Test8.java

D:\ phani>java Test8
local value 654
static value 321
after passing local value 654
```

```java
class A
{
 void m11()
  {
   System.out.println("class A method m11");
  }
}
class B
{
void m12()
  {
   System.out.println("class B method m12");
  }
}

class Test9
{
  void m13(A a1)
 {
   System.out.println("instance method m13");
    A a13=new A();
    a13.m11();
    B b14=new B();
    b14.m12();
 }

 public static void main(String[] args)
 {
  Test9 t9=new Test9();
  A a2=new A();
  t9.m13(a2);
 }
}
```

Test9.java

```
//parameters passing as objects

D:\ phani>javac Test9.java

D:\ phani>java Test9
instance method m13
class A method m11
class B method m12
```

6 method return type

java method retun type is mandatory

```
class Test10
{
public static int square(int x)
 {
  return x*x;
  }
public static int cube(int x)
 {
  return x*x*x;
 }
public static void main(String[] args)
 {
  System.out.println("square method"+square(10)
      );
  System.out.println("cube method"+square(3));
 }
}
```

Test10.java

```
//method retun int values

D:\ phani>javac Test10.java

D:\ phani>java Test10
square method100
cube method 9
```

7 this keyword

Keyword THIS is a reference variable in Java that refers to the current object.

1. It can be used to refer instance variable of current class
2. It can be used to invoke or initiate current class constructor
3. It can be passed as an argument in the method call
4. It can be passed as argument in the constructor call
5. It can be used to return the current class instance

```
class Test11
{
 int x=10;
 int y=20;
 public void m1(int p,int q)
{
  this.x=p;
  this.y=q;
}
public static void main(String[] args)
{
 Test11 ta=new Test11();
  System.out.println("befor passing "+ta.x+" "+
      ta.y);
   ta.m1(123,456);
  System.out.println("after passing "+ta.x+" "+
```

```
//this key word
D:\phani>javac Test11.java

D:\phani>java Test11
befor passing 10 20
after passing 123 456
```

```
        ta.y);

  }
}
```

Test11.java