CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE251: Java PROGRAMMING

July – November 2020

Chapter – 2

Types of Method in Java

Devang Patel Institute of Advance Technology and Research

What is the purpose of method? (in real time)

```
class  Student
{
    int a=10;
    int b =20;
    S.O.P(a+b);
}
```

Inside the class directly business logic allowed or not?

2 types of method in Java, generally

1. Instance Method
2. Static Method

```
void method()
{
    int a =10;
    int b=20;
}
```

```
static void method()
{
    int a =10;
    int b=20;
}
```

Method syntax?

Modifiers ReturnType Method_Name
(Prameter_List) throws Exception

Method Signature?

Method_Name(Parameter_List)

Method Component/Parts

```
class MethodSample
{
        int a =100;
        int b = 200;
    void m1()                  // method declaration
    {
       // method limplementation
    }
    public static void main(String[] args)
    {
         MethodSample  m = new MethodSample();
        m.m1(); // method calling
    }
}
```

Method Without Parameters

```
class MethodSample
{
      void m1() {
            S.O.P("m1 method");
            }
      static void m2() {
            S.O.P("m2 method");
            }
      static
{
       S.O.P("static ");}
      public static void main(String[] args)
      {
             MethodSample  m = new MethodSample();
            m.m1();
             MethodSample.m2();
}}
```

Method With Parameters

```
class MethodSample
{
    void m1(int a, char ch) {
        S.O.P("m1 method"); S.O.P(a); S.O.P(ch);
        }
    static void m2(string str, double d) {
        S.O.P("m2 method");  S.O.P(str); S.O.P(d);
        }
    public static void main(String[] args)
    {
         MethodSample  m = new MethodSample();
        m.m1(10, 'M');
         MethodSample.m2("CHARUSAT", 10.5);
}}
```

Conclusion

If method expecting parameters then we need to pass parameters and also order of parameters is also important

At project level method is expecting objects not primitive data only

Method with expecting objects

```
class Admin{ }

class Employee{ }

class Department{ }

class Student{ }
```

```
class Management{
void m1( Admin a, Employee e){
S.O.P("m1 method");
}
static void m2( Department d, Student s){
S.O.P("m2method");
}
P.S.V.M.(String[] args)
{
Management m = new Management();
}
}
```

How to call methods?

```
Admin a =new Admin();
Employee e1 = new Employee();
m.m1(a,e1);

// m.m1(new Admin(), new Employee());
```

Exercise- create 2 classes

```
class Circle{
// create 2 instance variable name as pi=3.14 & radius
// create two methods name as
void setRadius{      }
double calculateArea{    }
}
```

```
class CallMethod{
P.S.V.M(){
 // create local variable area
// create object of Circle class
// call setRadius() method with required argument
// call calculateArea() method & store return value
// print the result
}}
```

Duplicate methods are not allowed

```
class MethodSample
{
      void m1() {
            S.O.P("m1 method");
            }
      void m1() {
            S.O.P("m1 method");
            }
      public static void main(String[] args)
      {
             MethodSample  m = new MethodSample();
            m.m1();

}}
```

Two methods with same signature is not allowed

Will it work without error?

```
class MethodSample
{
      void m1() {
            S.O.P("m1 method");
            }
      void m1(int a) {
            S.O.P("m1 method with one argument ");
            }
      public static void main(String[] args)
      {
             MethodSample  m = new MethodSample();
            m.m1();

}}
```

Two methods with different signature are allowed

Will it work without error?

```
class MethodSample
{
    m1() {
        S.O.P("m1 method");
        }
    public static void main(String[] args)
    {
         MethodSample  m = new MethodSample();
        m.m1();

}}
```

In Java, method return type is mandatory

Inner Method

```
class InnerMethod
{
        void m1() {
                S.O.P("m1 method");
                void m2() {
                        S.O.P("m2 method ");
                        }

                }
        public static void main(String[] args)
        {
                 InnerMethod  m = new InnerMethod();
                m.m1();

}}
```

In Java, inner method is not allowed

Will it work without error?

```
class MethodSample
{
      void m1() {
            S.O.P("m1 method");
            m2()
            }
      void m2() {
            m3();
            }
      void m3() {    S.O.P("m3 method");       }
      public static void main(String[] args)
      {
             MethodSample  m = new MethodSample();
            m.m1();

}}
```

Inside method, instance method calling is allowed

Local variable name as an instance variable name

```
class Test
{
int a =100;
int b = 200;
void m1(int x, int y)
{      S.O.P(x+y);
              S.O.P(a+b);
}
P.S.V.M(String[] args)
{
    Test  t = new Test();
     t.m1(10,20);
}}
```

```
class Test
{
int x =100;
int y = 200;
void m1(int x, int y)
{      S.O.P(x+y);
              S.O.P(x+y);
}
P.S.V.M(String[] args)
{
    Test  t = new Test();
     t.m1(10,20);
}}
```

What will be the output here?

Output in second case

30

30

Why??

Local variable having higher priority over instance variable

How to print instance variable??

To represent instance variable use this keyword

How??

S.O.P(this.x + this.y);

Output: 300

What happen if method is static type??

```
class Test
{
      int x =100;
      int y = 200;
      static void m1(int x, int y)
      {      S.O.P(this.x+this.y);
                  S.O.P(x+y);
      }
      P.S.V.M(String[] args)
      {
            Test  t = new Test();
             t.m1(10,20);
      }
}
```

Will it work without error??

Compilation error:-
non static variable this
can not be referenced
from a static context

Inside the static method this keyword is not allowed

At project level method is expecting Array type parameters also

Exercise- Swap 2 number using array and method

Solution

```
public class SwapNumber {

  public static void main(String[] args) {
    int[] num = { 1, 2 };
   System.out.println("Before  swap");
    System.out.println("#1: " + num[0]);
    System.out.println("#2: " + num[1]);

    swap(num);

    System.out.println("After  swap");
    System.out.println("#1: " + num[0]);
    System.out.println("#2: " + num[1]);
  }
}
```

```
public static void swap(int[] source) {

  if (source != null && source.length == 2) {
     int temp = source[0];
     source[0] = source[1];
     source[1] = temp;
   }
  }
```

Array Parameter Reference

```
import java.util.Arrays;
public class ArrayReference {

public static void main(String[] args) {
   int[] origNum = { 1, 2, 3 };
   System.out.println("Before method  call:" + Arrays.toString(origNum));

    tryArrayChange(origNum);

   System.out.println("After method  call:" + Arrays.toString(origNum));
  }

  public static void tryArrayChange(int[] num) {
   System.out.println("Inside method-1:" + Arrays.toString(num));

   num = new int[] { 10, 20 };

   System.out.println("Inside method?2:" + Arrays.toString(num));
  }
} https://www.youtube.com/watch?v=BHtfb3lfc-g&t=282s
```

Conclusion

Because an array is an object, a copy of its reference is passed to a method.

If the method changes the array parameter, the actual parameter is not affected.

```
Class test
{
public static void changeContent(int[] arr)
 {
arr[0] = 10;
}
public static void changeRef(int[] arr) {
   arr = new int[2];
  arr[0] = 15;
}
public static void main(String[] args) {
    int [] arr = new int[2];
    arr[0] = 4;
    arr[1] = 5;
changeContent(arr);
    System.out.println(arr[0]);
    changeRef(arr);
  System.out.println(arr[0]);
}
}
```

Elements of the Array Parameter

```
import java.util.Arrays;

public class Main {
  public static void main(String[] args) {
    int[] origNum = { 1, 2, 3 };
    String[] origNames = { "Java", "SQL" };
    S.O.P("Before method  call, origNum:"
       + Arrays.toString(origNum));
    S.O.P("Before method  call, origNames:"
       + Arrays.toString(origNames));

    tryElementChange(origNum);
    tryElementChange(origNames);

    S.O.P("After method  call, origNum:"
       + Arrays.toString(origNum));
    S.O.P("After method  call, origNames:"
       + Arrays.toString(origNames));
  }}
```

```
public static void
tryElementChange(int[] num) {
   if (num != null && num.length > 0) {
     num[0] = -1;
   }
 }
```

```
public static void
tryElementChange(String[] names) {
   if (names != null && names.length > 0) {
     names[0] = "T";
   }
 }
```

Output

```
Before method  call, origNum:[1, 2, 3]
Before method  call, origNames:[Java, SQL]
After method  call, origNum:[-1, 2, 3]
After method  call, origNames:[T, SQL]
```

Conclusion

The values stored in the elements of an array parameter can always be changed inside a method.

What is the output here??

```
class Item {
  private double price;
  private String name;

  public Item(String name, double initialPrice) {
    this.name = name;
    this.price = initialPrice;
  }

  public double getPrice() {
    return this.price;
  }

  public void setPrice(double newPrice) {
    this.price = newPrice;
  }

  public String toString() {
    return "[" + this.name + ", " + this.price + "]";
  }
}
```

Continue

```
public class MainArray {
  public static void main(String[] args) {

    Item[] myItems = { new Item("Pen", 2.11), new Item("Pencil", 0.10) };
    System.out.println("Before method  call  #1:" + myItems[0]);
    System.out.println("Before method  call  #2:" + myItems[1]);

    tryStateChange(myItems);

    System.out.println("After method  call  #1:" + myItems[0]);
    System.out.println("After method  call  #2:" + myItems[1]);
  }

  public static void tryStateChange(Item[] allItems) {
    if (allItems != null && allItems.length > 0) {
      allItems[0].setPrice(0.38);
    }  }}
```

Output

```
Before method  call  #1:[Pen, 2.11]
Before method  call  #2:[Pencil, 0.1]
After method  call  #1:[Pen, 0.38]
After method  call  #2:[Pencil, 0.1]
```

Method return type

Java return type is optional or mandatory??

Its mandatory, at least we need to write void

Return type may be

Primitive
Class type
Array type
Enum type

Return type at primitive level

```
class MethodSample
{
    int m1() {    return 10;      }
    float m2() {  return 10.5f;}
    static char m3() {    return 'M';    }
    public static void main(String[] args)
    {
        MethodSample  m = new MethodSample();
        int a = m.m1();
        S.O.P("m1 method return value is = "+a);
       float f = m.m2();
        S.O.P("m2 method return value is = "+f);
       char c = MethodSample.m3();
        S.O.P("m3 method return value is = "+c);
}}
```

Return type at class & Object level

```
class Employee{
}

class Student{
 }
```

```
class Management{
Employee m1(){
S.O.P("m1 method is called");
Employee e = new Employee();
return e;
}
Student m2(){
S.O.P("m2method");
return new Student();
}
P.S.V.M.(String[] args)
{
Management m = new Management();
Employee e = m.m1();
Student s = m.m2();
}}
```

Return same class object

```
class Test{
    Test m1(){
        S.O.P("m1 method is called");
        Test t = new Test();
        return t;
        }
    Test m2(){
        S.O.P("m2method");
        return new Test();
        }
    P.S.V.M.(String[] args)
        {
            Test t = new Test();
            Test t1 = t.m1();
            Test t2 = t.m2();
}}
```

Will it work without error??

Use this keyword also

```
class Test{
    Test m1(){
        S.O.P("m1 method is called");
        Test t = new Test();
        return t;
        }
    Test m2(){
        S.O.P("m2method");
        return this;
        }
    P.S.V.M.(String[] args)
        {
            Test t = new Test();
            Test t1 = t.m1();
            Test t2 = t.m2();
}}
```

Few cases

```
int a =100;
int m1(int a)
{
    return a;
}
```

```
int a =100;
int m1()
{
    return a;
}
```

```
int a =100;
int m1(int a)
{
    return this.a;
}
```

Any Question