IT251: Java PROGRAMMING
July – November 2022

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();
}}
```

```
Class a
{
Static
{
        s.o.p("hello");
}
}
```

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

Method Overloading in Java

- If a class has multiple methods having same name but different in parameters, it 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.
- Method overloading *increases the readability of the program*.

Ways to overload the method

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

Method Overloading: changing no. of arguments

```
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 TestOverloading
        {
        public static void main(String[] args)
        {
                System.out.println(Adder.add(11,11));
                System.out.println(Adder.add(11,11,11));
        }
}
```

Output:

```
22
33
```

Method Overloading: changing data type of arguments

```
class Adder{
static int add(int a, int b)
        {
        return a+b;
        }
static double add(double a, double b)
        {
        return a+b;
        }
}
class TestOverloading
        {
        public static void main(String[] args)
        {
        System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(12.3,12.6));
        }
}
```

Output:

```
22
24.9
```

What will be the output of the following program?

```
class Adder
{
static int add(int a,int b)
        {
        return a+b;
        }
static double add(int a,int b)
        {
        return a+b;
        }
}
class TestOverloading
{
        public static void main(String[] args)
        {
        System.out.println(Adder.add(11,11));
        }
}
```

Output:
Compile time Error

Is it possible to overload main method in java?

What will be the output of the following program?

```
class Test {
public static void main(String[] args)
{
System.out.println("main with String[]");
}
public static void main(String args)
{
System.out.println("main with String");
}
public static void main()
{
System.out.println("main without args");
}
}
```

Can We overload static method??

YES

How???

Points to remember while working with static method overloading

Method signature must be different

Example

```
public class OverloadStaticMethodExample1
{
	public static void display()
	{
		System.out.println("Static method called.");
	}
	public static void display(int x)
	{
		System.out.println("An overloaded static method called.");
	}
	public static void main(String args[])
	{
		OverloadStaticMethodExample1.display();
		OverloadStaticMethodExample1.display(160);
	}
}
```

Output:

```
Static method called
An overloaded static method called.
```

can we overload the methods if they are only different by static keyword?

NO

Why?

```
public class OverloadStaticMethodExample2
{
	public static void sum(int a, int b)
	{
		int c=a+b;
		System.out.println("The sum is: "+c);
	}
	public void sum(int a, int b)
	{
		int c=a+b;
		System.out.println("The sum is: "+c);
	}
	public static void main(String args[])
	{
		OverloadStaticMethodExample2.sum(12, 90);
	}
}
```

Output:

```
Error: method sum(int, int) is
already defined in class
```

Can we override a static method?

NO

Why?

- we cannot override static methods because method overriding is based on dynamic binding at runtime and the static methods are bonded using static binding at compile time.
- The calling of method depends upon the type of object that calls the static method. It means:
  - If we call a static method by using the parent class object, the original static method will be called from the parent class.
  - If we call a static method by using the child class object, the static method of the child class will be called.

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));
  }
}
```

Array Parameter Reference

```
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]);
}
}
```

Explanation

```
public class Main {

    public static void main(String[] args) {
        Foo f = new Foo("f");
        changeReference(f);
modifyReference(f);
}

    public static void changeReference(Foo a) {
        Foo b = new Foo("b");
        a = b;
    }

    public static void modifyReference(Foo c) {
        c.setAttribute("c");
    }

}
```

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.

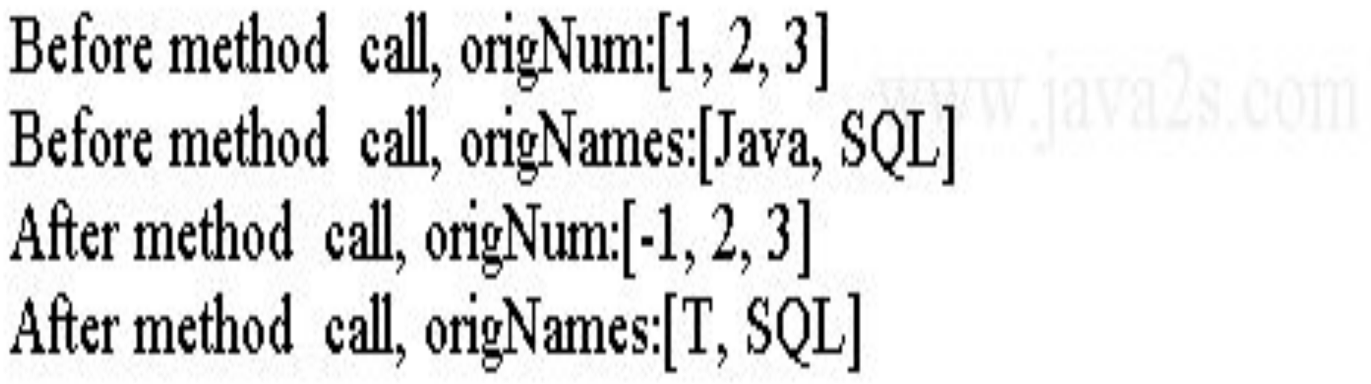
Elements of the Array Parameter

```
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

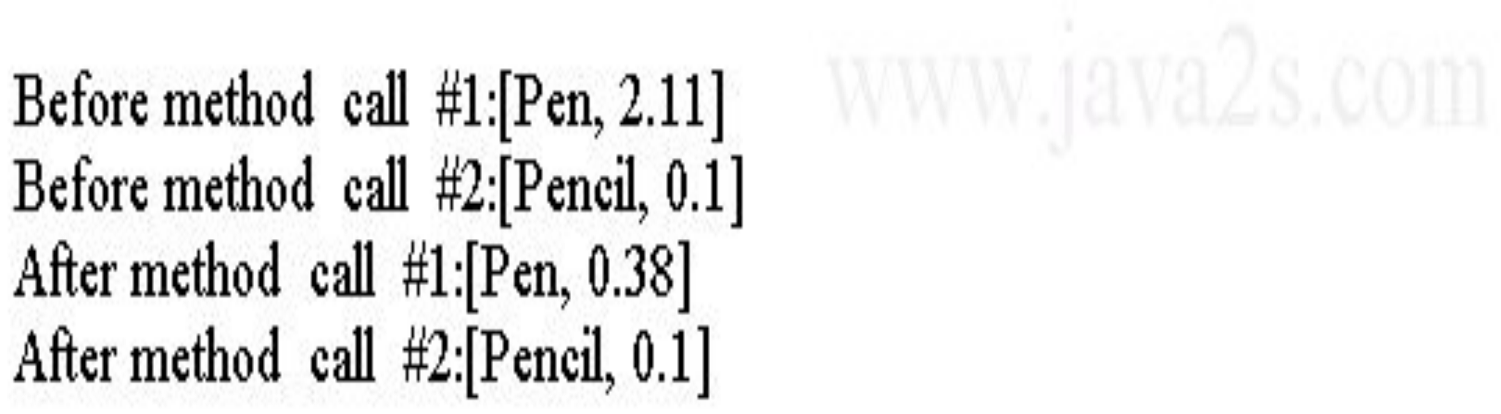
The values stored in the elements of an array parameter can always be changed inside a method.

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 +
"]";
  }
}
```

```
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

Example

```
class Test
{
    // logic here
}
```

Create object of a Test class **Test t = new Test();**

NOTE: During the object creation constructor will be executed

Instantiation vs. Initialization in context of object

Instantiation //

```
Test   t ;
```

Initialization //

```
t  = new Test();
```

How many different approach to create an object in java?

What are the rules to declare constructor in java?

- constructor cannot have a return type.
- A constructor must have the same name as that of the Class.
- Constructors cannot be marked static.
- A constructor cannot be marked abstract.
- A Constructor cannot be overridden.
- A Constructor cannot be final.

Few Question must have in your mind about constructor rules.

Why constructor name is same as class name?

Why return type is not allowed for constructor?

Can we declare constructor as public, private, protected?

Example

```
class Test
{
      void m1()
      {
        S.O.P.("M1 method");
      }
      P.S.V.M(String[] args){
            Test t = new Test();
      }
}
```

```
/*
  Test()
  {
    // empty
implementation
  }
*/
```

Zero argument constructor

Type of default constructor

Types of constructor

1. Default Constructor
   (i) Always Zero argument constructor

NOTE:
Default Constructor generated by compiler and executed by JVM

2. User define Constructor
   (ii) Zero argument constructor
   (iii) Parameterized constructor

User define constructor example (Constructor Overloading)

```
class Test
{
void m1()
{
 S.O.P.("M1 method");
  }
          Test()
          {
          S.O.P.("0-argument constructor");
           }
          Test(int a)
          {
           S.O.P.("1-argument constructor");    }

          P.S.V.M(String[] args){
                    Test t = new Test();
                     Test t1 = new Test(10);
                    t.m1(); t1.m1();
          }
}
```

Will it compile or not?

```
class Test
{
      Test(int a)
      {
      S.O.P.("1-argument constructor");
       }
      P.S.V.M(String[] args){
            Test t = new Test();
             Test t1 = new Test(10);
      }
}
```

Default constructor will not be
Generated in this case
Compiler error

Design a program to calculate Rectangle area and perimeter

1. Create two classes name as **Rectangle** and **CalRectangle**

1. Declare instance variable as length & width
2. Create Rectangle constructor to initialize instance variable
3. Create setLength & setWidth method
4. Create area and perimeter method and return values to main method

1. Initialize object here (main method)
2. Set the length and width
3. Print area and perimeter

Topic to discuss

- Instance block
- Static Block

Instance block

```
{
    System.out.println(“This is instance block");
 }
```

What is the output here?

```
class InstanceBlock
{
        {
        System.out.println(“Instance block called");
       }

      InstanceBlock()
    {
        System.out.println("Constructor called");
    }
    public static void main(String[] args)
    {
       InstanceBlock a = new InstanceBlock();
    }
}
```

Multiple Instance Block

```
class InstanceBlock
{
        {
        System.out.println(“Instance block-1 called");
       }
      {
        System.out.println(“Instance block -2 called");
       }
      {
        System.out.println(“Instance block-3  called");
      }

      InstanceBlock()
     {
        System.out.println("Constructor called");
     }
    public static void main(String[] args)
    {
         InstanceBlock a = new InstanceBlock();
    }
}
```

Instance Block Use to Initialize the Variable

```
Class Emp
{
 int eid;
      {
            eid=111;
       }
      void display()
      {
                S.O.P(eid);
      }
P.S.V.M.()
{
     new Emp().display();
}
}
```

Static block

```
static {
    System.out.println("static block called ");
  }
```

What is the output here

```
class Test {
    static int i;
    int j;
    static {
        i = 10;
            System.out.println("static block called ");
    }
    Test(){
            System.out.println("Constructor called");
    }
}
 class Main {
    public static void main(String args[]) {
      Test t1 = new Test();
      Test t2 = new Test();
    }
}
```

Case Study

Assume we have 3 classes and each class contain static block but only 1 class contain main method.

I want to execute all 3 class static block then How??

Solution

Using forName() method

Example-

```
Class.forName(“ClassName”);
```

Passing and Returning Objects in Java

- While creating a variable of a class type, we only create a reference to an object. Thus, when we pass this reference to a method, the parameter that receives it will refer to the same object as that referred to by the argument.

```
class ObjectPassDemo
{
   int a, b;
   ObjectPassDemo(int i, int j)
   {
      a = i;
      b = j;
   }
    boolean equalTo(ObjectPassDemo o)
   {
      return (o.a == a && o.b == b);
   }
}
public class Test
{
   public static void main(String args[])
   {
      ObjectPassDemo ob1 = new ObjectPassDemo(100, 22);
      ObjectPassDemo ob2 = new ObjectPassDemo(100, 22);
      ObjectPassDemo ob3 = new ObjectPassDemo(-1, -1);
      System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));
      System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));
   }
}
```

Output:

```
Ob1 == ob2: true
Ob1 == ob3: false
```

Returning Objects

```
class ObjectReturnDemo
{
   int a;
   ObjectReturnDemo(int i)
   {
      a = i;
   }
ObjectReturnDemo incrByTen()
   {
      ObjectReturnDemo temp =
           new ObjectReturnDemo(a+10);
      return temp;
   }
}
public class Test
{
   public static void main(String args[])
   {
      ObjectReturnDemo ob1 = new ObjectReturnDemo(2);
      ObjectReturnDemo ob2;
      ob2 = ob1.incrByTen();
      System.out.println("ob1.a: " + ob1.a);
      System.out.println("ob2.a: " + ob2.a);
   }
}
```

Output:

```
Ob1.a: 2
Ob2.a: 12
```

What is the purpose of constructor in program?

Two purpose or two advantages of constructor

1. Constructor are user to write business logic of application those logic are executed during object creation.
2. To understand second advantage, we must know use of **this** keyword

Case-01

```
class Emp{
   int eid;
   String ename;
   float esal;
   void display()
{
 S.O.P.("Emp id is : ", +eid);
                           S.O.P.("Emp name is :
", +ename);                              S.O.P.("Emp
salary is : ", +esal);
 }
    P.S.V.M(String[] args)
{
         Emp e = new Emp();
                       e.display();
}
}
```

Output:

```
0
Null
0.0
```

Case-02

```
class Emp
{
   int eid;
   String ename;
   float esal;
   Emp()
 {
eid=123; ename = “XYZ”, esal= 1000;
}
   void display()
{
 S.O.P.(“Emp id is : ”, +eid);
 S.O.P.(“Emp name is : ”, +ename);
 S.O.P.(“Emp salary is : ”, +esal);
}
    P.S.V.M(String[] args){
      Emp e = new Emp();
                  e.display();
}
 }
```

Output:

```
123
XYZ
1000
```

Constructor uses to initialize some values to instance variable during object creation

What is the problem in case-02?

- With the given logic, for all the object of **Emp** class, same value will be initialize
- Means different object but same value will initialize

What is the solution?

Parameterize constructor

```
Emp(int eid, String ename, float esal)
{
  this.eid = eid;
  this.ename = ename;
  this.esal = esal;
}
```

```
Emp e1 = new Emp(111, “abc”, 1000);
Emp e2 = new Emp(222, “xyz”, 2000);
```

How one constructor call to another constructor?

Constructor calling

Example-1

```
class Test
{     Test()
      {        this(10);
                    S.O.P.("0-argument constructor");    }
      Test(int a)
      {        this(10,20);
                    S.O.P.("1-argument constructor");    }
      Test(int a, int b)
      {     S.O.P.("2-argument constructor");    }

      P.S.V.M(String[] args){
            Test t = new Test();
      }}
```

Will it compile or not?

```
class Test
{     Test()
      {     S.O.P.("0-argument constructor");
                        this(10);   }
      Test(int a)
      {          S.O.P.("1-argument constructor");
            this(10,20);   }
      Test(int a, int b)
      {     S.O.P.("2-argument constructor");    }

      P.S.V.M(String[] args){
            Test t = new Test();
      }}
```

Compiler error

this() must be the first statement in constructor

What is the conclusion from constructor calling?

- One constructor is able to call only **one** constructor at a time because one **this** statement is allowed

Can we create object inside constructor?

Will it work or not?

```
class Test
{
   Test()
   {
      new Test();
   }
   public static void main(String[] args)
   {
      System.out.println("Main Method");
      new Test();
   }
}
```

It is possible to create object inside constructor.

It will not give compile time error, but it is not recommended to create.

If we create object inside constructor, it invokes the constructor recursively.

Hence it cannot complete the process of Object creation.

Can we define a method with same name of class?

Will it compile or not?

```
class Test
{
        void Test()
        {
                System.out.println("This is a method not a constructor");
        }
        public static void main(String[] args)
        {
                System.out.println("Hello World!");
                Test t = new Test();
                t.Test();
        }
}
```

YES

If we place return type in constructor prototype will it leads to Error?

NO

Reason- No, because compiler and JVM considers it as a method.

How compiler and JVM can differentiate constructor and method **definitions** of both have same class name?

Ans- A compiler and a JVM differentiates constructor and method invocations by using 'new' keyword. If 'new' keyword is used in calling then a constructor is executed else method is executed.

Will it compile or not?

```
class Test
{
	Test()
	{
		System.out.println("0-arg Constructor");
	}
	void Test()
	{
		System.out.println("THis is method not constructor");
	}
	public static void main(String[] args)
	{
		System.out.println("Hello World!");
		Test t = new Test();
		t.Test();
	}
}
```

Compile and run successfully

By return type

How compiler and JVM can differentiate constructor and method **invocations** of both have same class name?

by **new** Keyword

What is the difference between constructor and method in java?

write it in your own

Java Constructor Access Modifiers

Constructors can have any of the **access modifiers**: public, protected, private, or none.

Constructors cannot be abstract , final , native , static , or synchronized .

Copy Constructor

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

WHAT ?

A copy constructor is a constructor that takes only one argument which is of the type as the class in which the copy constructor is implemented.

For example, let us assume a class namely Car and it has a constructor called copy constructor which expects only one argument of type Car.

WHY ?

- Copy constructors are widely used for creating a duplicates of objects known as cloned objects.
- Duplicate object in the sense the object will have the same characteristics of the original object from which duplicate object is created.
- But we have to ensure that both original and duplicate objects refer to different memory locations.

Example

```
class CopyConstructor{
    int id;
    String name;
    CopyConstructor(int i,String n){
    id = i;
    name = n;
    }

    CopyConstructor(CopyConstructor s){
    id = s.id;
    name =s.name;
    }
    void display()
     {
          System.out.println(id+" "+name);
     }
```

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

CopyConstructor s1 = new
CopyConstructor(111,"Karan");

CopyConstructor s2 = new CopyConstructor(s1);

s1.display();

s2.display();

}
}
```

Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object.

Example

```
class CopyConstructor{
   int id;
   String name;
   CopyConstructor(int i,String n){
   id = i;
   name = n;
   }
CopyConstructor(){}

void display()
{
       System.out.println(id+" "+name);
}
```

```
 public static void main(String args[]){
   CopyConstructor s1 = new
CopyConstructor(111,"Karan");
    CopyConstructor s2 = new CopyConstructor();

        s2.id=s1.id;
        s2.name=s1.name;

        s1.display();
        s2.display();
   }
}
```

Concept of destructor in JAVA

What is the aim of destructor in any OOPs language?

1. Free up the memory (c++ suffer from memory allocation / deallocation)
2. Clean up any other resources (like closing of open file stream)

Java take cares of all and hence there is no destructor in Java.

With the help of Garbage collection

Garbage Collector in java

- Dereferencing of the object can be done
  - By nulling the reference
  - By assigning a reference to another
  - By anonymous object etc.

finalize() method

```
class Thing {
    public  int  number_of_things = 0;
    public String what;

public Thing (String what) {
        this.what = what;
        number_of_things++;
        }

   public void finalize ()  {
        number_of_things--;
        }
   }
```

```
public class TestDestructor {

    public static void main(String[] args)

    {
       Thing obj = new Thing("Test App");

     }
}
```

Final Keyword in java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context.

It can be use with

1) Variables
2) Methods
3) Class

Java final variable

- If we make any variable as final, we cannot change the value of final variable(It will be constant).

```
class Bike
{
 final int speedlimit=90;
 void run()
{
  speedlimit=400;
 }
 public static void main(String args[])
{
 Bike obj=new  Bike();
 obj.run();
 }
}
```

Output:
Compile time error

Java final method

- If we make any method as final, we cannot override it.

```
class Bike{
 final void run()
{
System.out.println("running");
}
}
class Honda extends Bike
{
  void run()
{
System.out.println("running safely with 100kmph");
}
  public static void main(String args[]){
  Honda honda= new Honda();
  honda.run();
  }
}
```

```
Output:
Compile time Error
```

Java final class

- If we make any class as final, we cannot extend it.

```
final class Bike
{}
class Honda1 extends Bike
{
  void run()
{
System.out.println("running safely with 100kmph");
}
  public static void main(String args[]){
  Honda1 honda= new Honda1();
  honda.run();
  }
}
```

Output:
Compile Time Error

Is final method inherited?

Yes, final method is inherited but we cannot override it

What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

Can we initialize blank final variable?

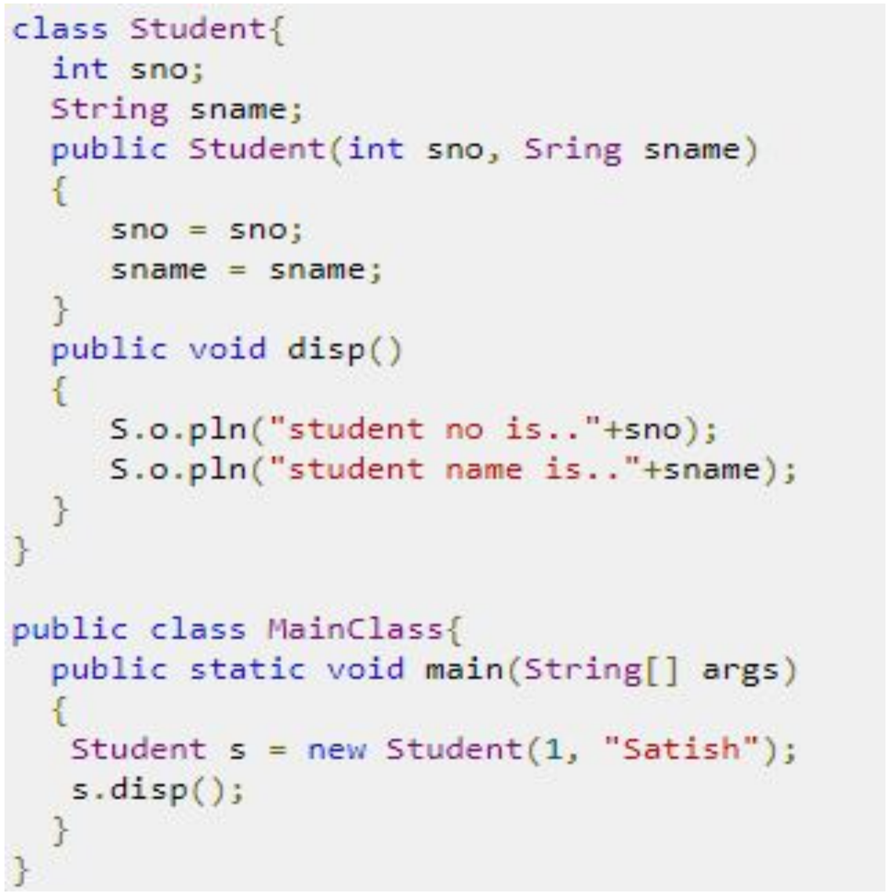
Yes, but only in constructor

Can we declare a constructor final?

No, because constructor is never inherited.

Any Question?

What is the output?



```
class Student{
  int sno;
  String sname;
  public Student(int sno, Sring sname)
  {
     sno = sno;
     sname = sname;
  }
  public void disp()
  {
     S.o.pln("student no is.."+sno);
     S.o.pln("student name is.."+sname);
  }
}

public class MainClass{
  public static void main(String[] args)
  {
   Student s = new Student(1, "Satish");
   s.disp();
  }
}
```

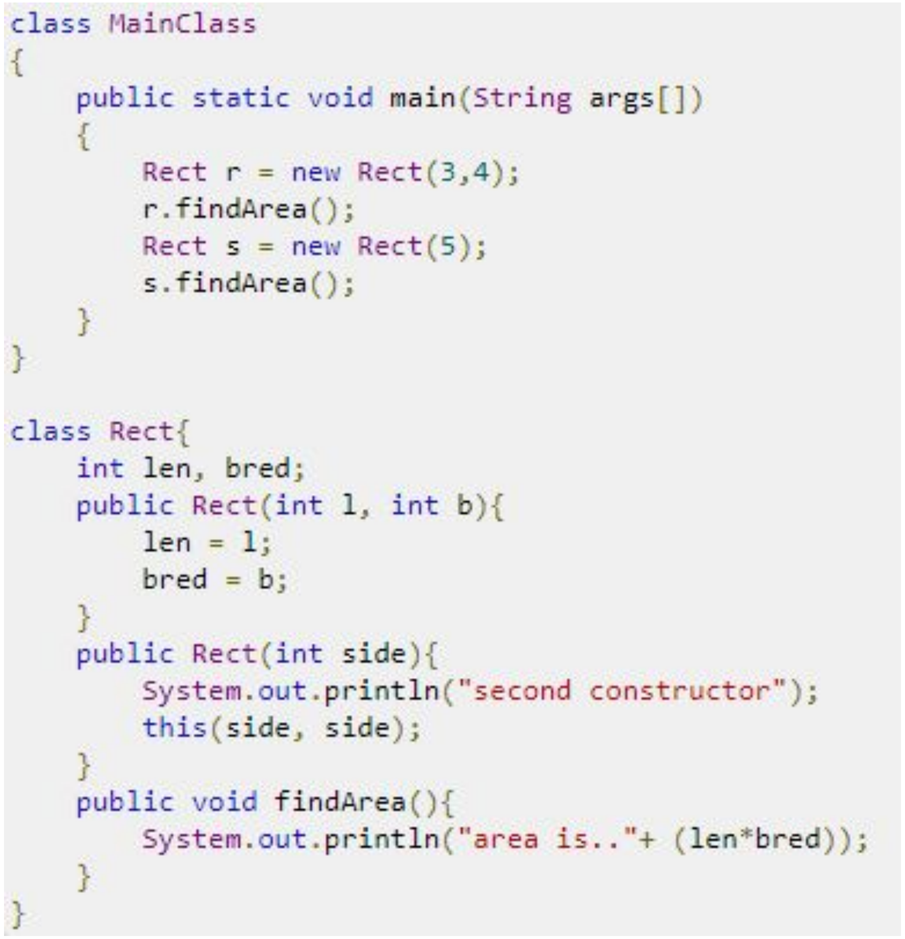
A. Student no is.. 1 student name is.. Satish

B. Compile time error in constructor

C. Run time error in constructor

D. Student no is.. 0 student name is.. null

What is the output?



```
class MainClass
{
    public static void main(String args[])
    {
        Rect r = new Rect(3,4);
        r.findArea();
        Rect s = new Rect(5);
        s.findArea();
    }
}

class Rect{
    int len, bred;
    public Rect(int l, int b){
        len = l;
        bred = b;
    }
    public Rect(int side){
        System.out.println("second constructor");
        this(side, side);
    }
    public void findArea(){
        System.out.println("area is.."+ (len*bred));
    }
}
```

A. compile time error

B. Run time error

C. It prints area is.. 12 and area is.. 25

D. It prints area is.. 25 and area is.. 25

What is constructor overriding?

A. Constructors can not be overridden in Java
B. Since constructor are special function which can't be inherited to derived classes, so it is not possible for derived classes to override a constructor of base class
C. Derived classes can call base class constructor by using super keyword
D. All are correct

Reference

https://deepeshdarshan.wordpress.com/2013/12/05/copy-constructors-in-java/