

CE251: Java PROGRAMMING
July – November 2020

Chapter – 2

Java Constructor

Lecture map today

- Defining a Constructor in Java
- Constructor Overloading - Multiple Constructors for a Java Class
- Default, no-arg Constructor
- Constructor Parameters
- Calling a Constructor
- Calling a Constructor From a Constructor
- Calling Constructors in Superclasses cover in Inheritance
- Java Constructor Access Modifiers

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?

```
class CSE {  
  
    // Declaring and initializing string  
    // Custom input string  
    String name = "CHARUSAT";  
  
    // Main driver method  
    public static void main(String[] args)  
    {  
        // As usual and most generic used we will  
        // be creating object of class inside main()  
        // using new keyword  
        CSE obj = new CSE();  
  
        // Print and display the object  
        System.out.println(obj.name);  
    }  
}
```

What are the rules to declare constructor in java?

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

2. User define Constructor

- (ii) Zero argument constructor
- (iii) Parameterized constructor

NOTE:

Default Constructor generated by compiler and executed by JVM

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);
```

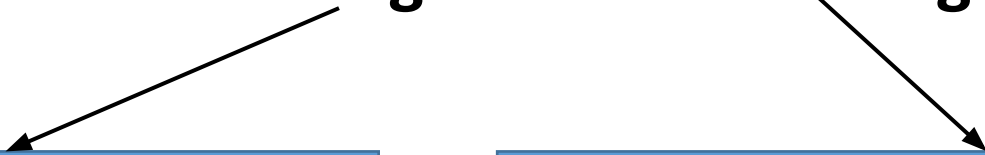
```
    }
```

```
}
```

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

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

Will it compile or not?

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

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

- Java garbage collection means----unreferenced objects
- Process of reclaiming the runtime unused memory automatically

- Advantages:
 - 1) Makes java memory efficient
 - 2) Automatically done

How can an object be unreferenced?

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

- By nulling reference : A a= new A();
A=null;

By assigning a reference to another
A b= new A();
a=b;

By anonymous object
New A();

Methods used in garbage collection

- Finalize(): invoked each time before object is garbage collected.
 - Method of object class
 - Performs clean up processing
-
- Gc(): Used to invoke garbage collection
 - Method of system class

Example

Class c1

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

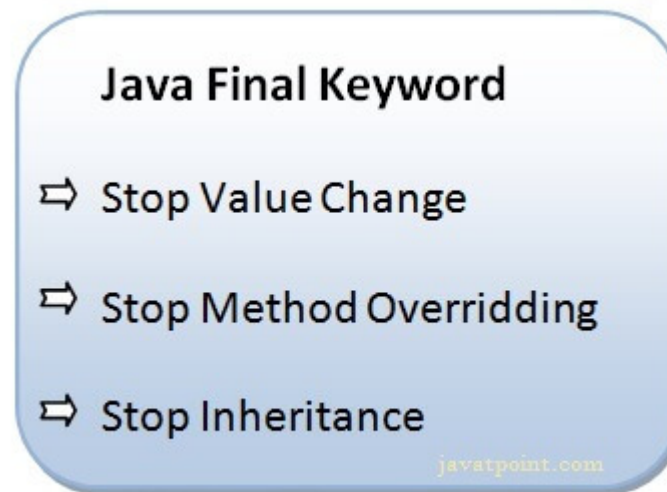
```
{  
C1 a = new C1();  
C1 b = new C1();  
a=null;  
b=null;  
System.gc();
```

```
}
```

```
Public void finalize()  
{  
System.out.println("G.C")  
}  
}
```

Final Keyword

- The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:
- variable
- method
- class



Final Variable

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

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

Output: Compile Time Error

Java final method

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

```
class Vehicle{  
    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 you make any class as final, you cannot extend it.

```
final class Vehicle{}
```

```
class Honda1 extends Vehicle{  
    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 you cannot override it. For Example:

- **class** Vehicle{
 - **final void** run(){System.out.println("running...");}
 - }
- **class** Honda2 **extends** Vehicle{
 - **public static void** main(String args[]){
 - **new** Honda2().run();
 - }
- }

Output: Running

Any Question?

What is the output?

```
class Student{
    int sno;
    String sname;
    public Student(int sno, String 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

Examine the following code: the constructor is preventing it from compiling. What is wrong?

```
class Cards {  
    String suit;  
  
    public DeckofCards(String suit) {  
        this.suit = suit;  
    }  
}
```

- 1.The this keyword is misused
- 3.Not enough arguments
- 4.Cannot have arguments
- 5.The name doesn't match the class

Examine the following code for a constructor. Which option will create a new instance of the Trees class and pass the parameter of elm to the constructor?

```
public class Trees {  
    String species;  
  
    public Trees(String trees) {  
        this.species = species;  
    }  
}
```

1.

```
Trees = new "elm";
```

2

```
new Trees("elm")
```

3

```
Trees tree = new Trees("elm");
```

4

```
Trees "elm" = new Tree;
```

6. Which of the following is not applicable for a constructor function?

(A) It has the same name as the class.

(B) It has no return-type

(C) It is usually used for initialisation

(D) It can be invoked using an object like any other member function

Reference

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