National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Spring 2022, Lab Manual – 04

Course Code: CL-217	**Course : Object Oriented Programming Lab**
Instructor(s) :	**Abeer Gauher, Hajra Ahmed, Syed Zain ul Hassan**

<u>LAB - 4</u>

Constructors and Garbage Collection

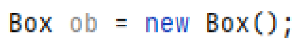
Constructor

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the ***new*** keyword, at least one constructor is called.

Example: Calling constructor for creating Box class object



```java
Box ob = new Box();
```

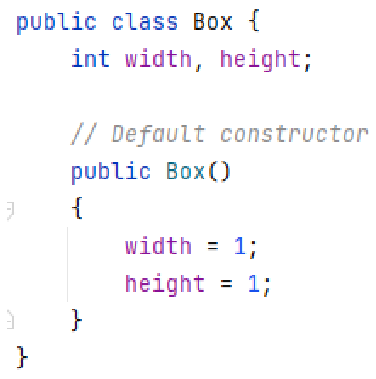
Types of Constructors

There are two types of constructors in Java: default (no parameter) constructor, and parameterized constructor.

Default Constructor

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

- Example of Default (no parameter) Constructor



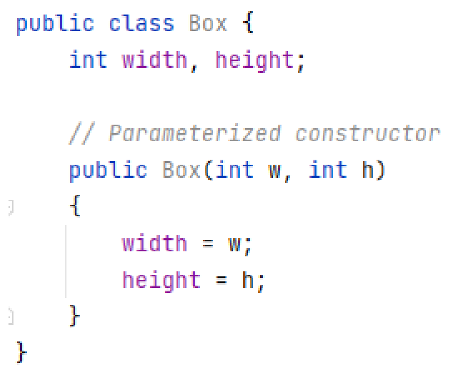
```java
public class Box {
    int width, height;

    // Default constructor
    public Box()
    {
        width = 1;
        height = 1;
    }
}
```

Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.
The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

- Example of Parameterized Constructor



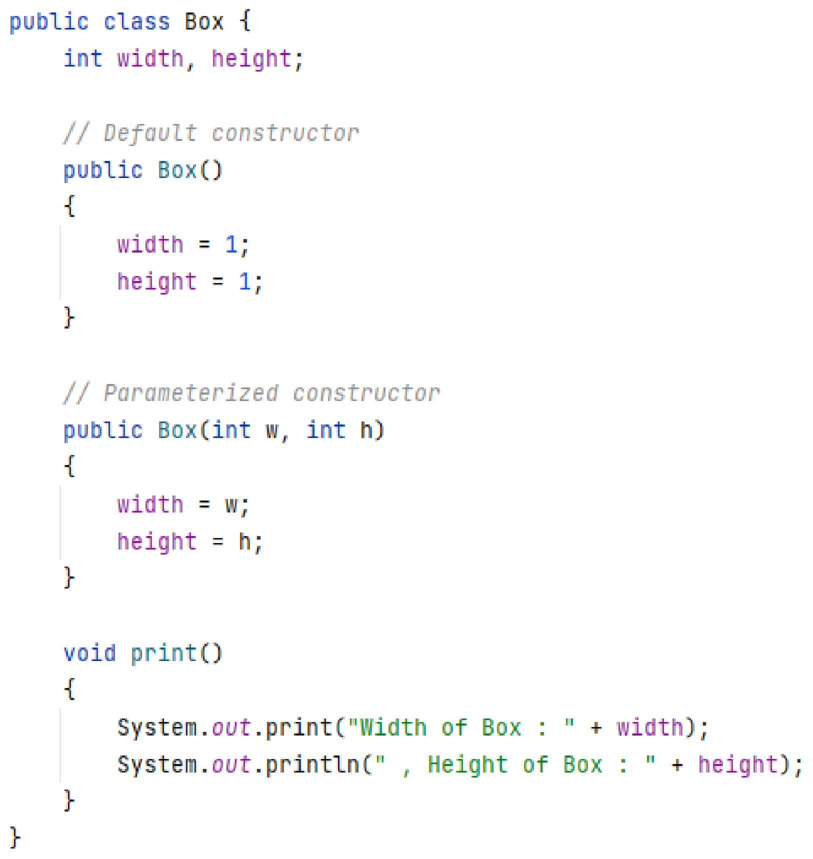
```java
public class Box {
    int width, height;

    // Parameterized constructor
    public Box(int w, int h)
    {
        width = w;
        height = h;
    }
}
```

Rules for Constructors

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. If there is no constructor in a class, compiler automatically creates a default constructor.

Complete Example



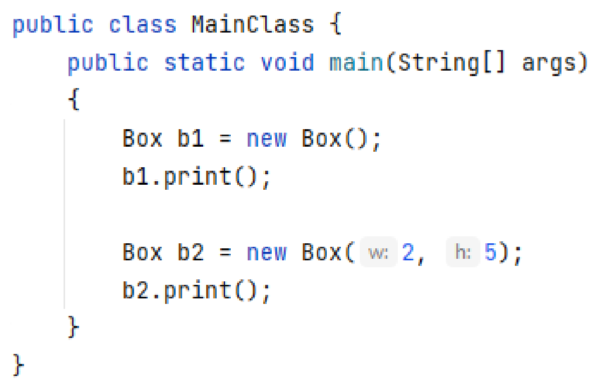
```java
public class Box {
    int width, height;

    // Default constructor
    public Box()
    {
        width = 1;
        height = 1;
    }

    // Parameterized constructor
    public Box(int w, int h)
    {
        width = w;
        height = h;
    }

    void print()
    {
        System.out.print("Width of Box : " + width);
        System.out.println(" , Height of Box : " + height);
    }
}
```

// Using these constructors from main class:



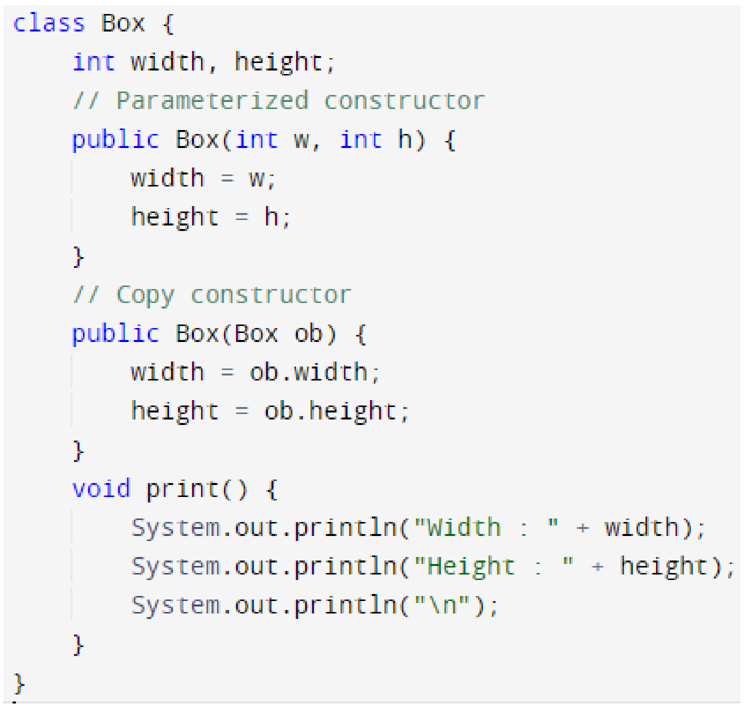
```java
public class MainClass {
    public static void main(String[] args)
    {
        Box b1 = new Box();
        b1.print();

        Box b2 = new Box( w: 2,  h: 5);
        b2.print();
    }
}
```

Copy Constructor

We can use one object to initialize another object in Java by passing the object as parameter to a constructor.

Example:



```java
class Box {
    int width, height;
    // Parameterized constructor
    public Box(int w, int h) {
        width = w;
        height = h;
    }
    // Copy constructor
    public Box(Box ob) {
        width = ob.width;
        height = ob.height;
    }
    void print() {
        System.out.println("Width : " + width);
        System.out.println("Height : " + height);
        System.out.println("\n");
    }
}
```

```
class MainClass {
    public static void main(String[] args) {
        Box b1 = new Box(2, 4);
        Box b2 = b1;

        System.out.println("Box 1 :- ");
        b1.print();
        System.out.println("Box 2:- ");
        b2.print();
    }
}
```

Output:

```
Box 1 :-
Width : 2
Height : 4


Box 2:-
Width : 2
Height : 4
```

Object Passing using Function

We can also pass objects as parameters to functions in Java just like other primitive type arguments.

Example:

```
class Square {
    int dim;
    public Square(int d) {
        dim = d;
    }
    public void print() {
        System.out.println("Sides: " + dim * dim);
    }
}
```

```
class Test {
    static void printer(Square o)
    {
        o.print();
    }

    public static void main(String[] args) {
        Square sq = new Square(5);
        printer(sq);
    }
}
```

Output:

```
Sides: 25
```

Garbage Collection

In Java, when we create an object of the class it occupies some space in the memory (heap). If we do not delete these objects, it remains in the memory and occupies unnecessary space that is not upright from the aspect of programming. To resolve this problem, we use the **Destructor**.

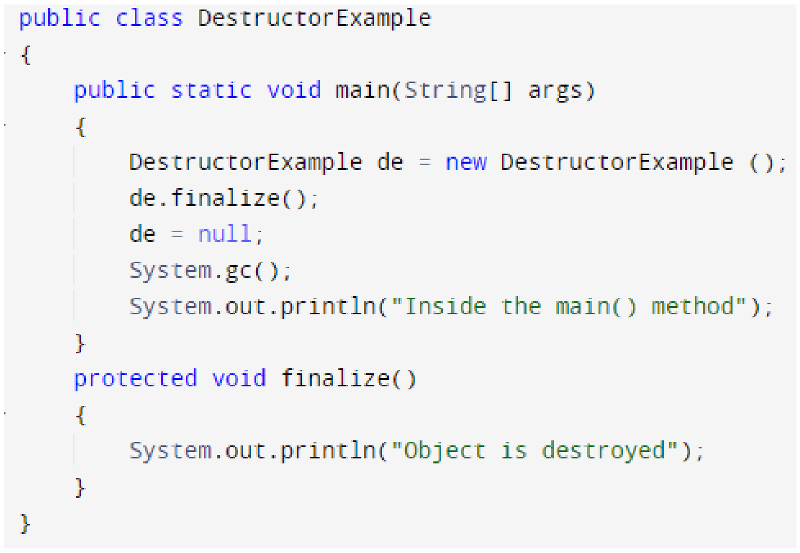
Unlike C++, which has explicit destructor, Java uses **finalize( )** method for this purpose. Since it cannot be predicted when the Java garbage collection occurs.

The finalize( ) Method

It is difficult for the programmer to forcefully execute the garbage collector to destroy the object. But Java provides an alternative way to do the same. The Java Object class provides the finalize() method that works the same as the destructor.

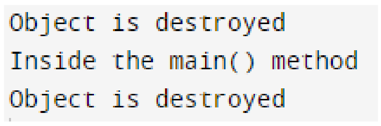
- It is a protected method of Object class
- It can be called only once.
- We need to call the finalize() method explicitly if we want to override the method.
- The gc() is a method of JVM executed by the Garbage Collector. It invokes when the heap memory is full and requires more memory for new arriving objects.

Example



```java
public class DestructorExample
{
    public static void main(String[] args)
    {
        DestructorExample de = new DestructorExample ();
        de.finalize();
        de = null;
        System.gc();
        System.out.println("Inside the main() method");
    }
    protected void finalize()
    {
        System.out.println("Object is destroyed");
    }
}
```

Output:



```
Object is destroyed
Inside the main() method
Object is destroyed
```

Lab Tasks

1. Modify the "**Box**" class given in example of this manual and implement the constructors as per the following requirements:
   a) A constructor that receives both height and width as parameter to create new **Box** object only if width and height are both positive values
   b) A constructor that receives only height as parameter and takes width as input from the user
   c) A constructor that receives no parameter and takes both width and height as user input

2. Create a new class **Student** with the following attributes:
   - studentID (int)
   - name (string)
   - cgpa (float)
   - department (string)

   Implement setter/getter functions for these variables and implement the following constructors:
   a) A constructor that receives parameters to initialize all variables
   b) A constructor that receives only studentID as input and initialize other variables by taking input from the user
   c) A constructor that receives only cgpa as input and does not initialize other variables.

   Create objects of **Student** class in main function using each constructor.

3. Implement a copy constructor for Student class and demonstrate its use in main function.

4. Create a class called **Tank** that can be filled and emptied, and has a termination condition that it must be empty when the object is cleaned up. Write a **finalize**( ) that verifies this termination condition.
Hint: You can create an int variable capacity that represents the water level of **Tank**.

5. Create a class containing an uninitialized String reference. Demonstrate that this reference is initialized by Java to null. Create a class with a String field that is initialized at the point of definition, and another one that is initialized by the constructor.