Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Constructors in Java/5b1bae827becc0459dea5621

TUTORIAL

Constructors in Java

Topics

Constructor is a object creator. Objects are instances of classes that are created using constructors. Every class requires a constructor in order to create object instances. In fact, if a class does not declare a constructor, the compiler automatically creates a constructor for the class. However, in most cases, you won't rely on the compiler to create a constructor for you, and you'll declare your own constructors. The following is the syntax of a constructor declaration:

```
modifers ClassName(arguments) throws Clause
{
        // Constructor body
}
```

The modifiers, arguments, and throws clause are optional. Valid modifiers are public, protected, private, or none. No modifier means that the constructor can only be accessed within the package where the class is declared. The public, protected, and private modifiers are used in the same manner as methods. The arguments and throws clause are also defined in the same way as methods.

One important way in which constructors differ from methods is that they are not inherited. Each class must define its own constructors. However, if no constructor is defined for a class, the compiler will supply a default constructor. The default constructor will not have any arguments and will simply invoke the constructor of the class's super class to construct an object of the class.

We can have parameterized and non-parameterized constructors. It depends on object creation that which constructor needs to be called. Following example shows it:

Java

```
class Shape
{
  int length, width, height;
  Shape()       // Constructor 1 having 0 arguments
  {
    length = width = height = 0;
  }
  Shape(int val)        // Constructor 2 having 1 arguments
  {
    length = width = height = val;
  }
  Shape(int l, int w, int h)    // Constructor 3 having 3 arguments
  {
    length = l;
    width = w;
    height = h;
  }
  int area()    // member function of class
  {
    return length*width*height;
  }
}

class Main{
  public static void main(String[] args)
  {
    Shape s1 = new Shape();     // It will call constructor 1
    Shape s2 = new Shape(10,20,5);      // It will call constructor 3
    Shape s3 = new Shape(10);           // It will call constructor 2

    System.out.println("s1 area = "+s1.area());
```

```
32      System.out.println("s2 area = "+s2.area());
33      System.out.println("s3 area = "+s3.area());
34    }
35  }
```

this operator in constructor

While using parameterized constructors we might use the parameter names same as the member fields name such as in above example if we define the 3rd constructor as below: -

```
Shape(int length, int width, int height)
{
        // Statements.
}
```

Now it becomes problematic as assigning values to member fields will not happen as local variable overrides member fields. So if we write:

```
Shape(int length, int width, int height)
{
        length = length;
}
```

Although we presume that left side is member field and right side is parameter but compiler will not be able to differentiate, so in this case we can distinguish them using this keyword. this is a keyword associated with the current object. So wherever, we need to specify member field we can use this.field as below:

```
Shape(int length, int width, int height)
{
        this.length = length;
        this.width = width;
        this.height = height;
}
```

So in these cases, we need to use this keyword for differentiating the common names. It is always a good practice to name variables meaningfully and using this for ambiguity solution.