CONSTRUCTORS IN JAVA

Constructors are used to initialize the state of an object when it is created. Constructors are invoked while creating objects, usually after the new keyword. A child class may also invoke a super constructor using the super keyword to initialize the parent object.

Constructors are similar in syntax to methods, but without any return type and have the same name as that of the class (case sensitive).

```
MyClass{

  MyClass(){
  }

}
```

Rules for defining constructors

1. Constructors have the same name as that of the class.
2. Constructors are similar to methods in syntax, but constructors do not have a return type; should not even specify void.

1. If you specify return type, then the supposed to be constructor will become a method and will not be considered as a constructor.

3. You can have a method with the same name as a constructor.

1. If you specify return type, then it will be considered a method and will not be invoked during initialization; you need to invoke it like just any other method.

4. Constructors can also be overloaded just like methods.

1. You may invoke the correct one based on arguments supplied.

5. Constructors can be invoked only during object creation or from other constructors using this keyword.

1. Only one constructor is invoked based on which overloaded version we specify. This constructor can then call other constructors if required, using this keyword (e.g. this() or this(2) etc.).

6. A default constructor is a constructor that takes no arguments, and mostly does nothing.
7. A default constructor with no parameters is automatically added by Java for a class if there are no user defined constructors.

1. Hence, even if we don't have a constructor for our class, we can instantiate a class using a no-argument constructor (e.g. new MyClass()).

8. If we provide at least one constructor, the default constructor is no longer added by Java.

9. One constructor can call another constructor using "this" keyword (e.g. this() or this(2)); arguments should match a constructor signature similar to method. You use the method name to specify a method whereas you use the "this" keyword for a constructor. This is called constructor chaining.

10. Java won't allow us to create an infinite loop by calling one constructor from another and then the other calling it back either directly or indirectly. Compilation will fail with message "Recursive constructor invocation". To know more about "this" keyword, refer to keyword-this-in-java.

11. During inheritance, subclass constructor should call a super class constructor. We use super keyword ti invoke a super class constructor (e.g. super(), super(1) etc).

12. First line of a constructor should be a call to super() or this(). If you don't provide any, constructor will add a call to default super (e.g. super()). So if your parent class doesn't have a visible default constructor, default super call will not work. You either need to add a default constructor in the parent or call any other visible constructors explicitely. This also means that if there are no visible constructors for a class, we cannot inherit that class. For instance, if a class contain only private contructors, you cannot inherit that class or instantiate that class from outside. Refer to the note on singleton for a use case where constructor is made private so that objects can be created only from within the class.

Examples

Example: Constructor look-alike method

```
MyClass{

  void MyClass(){}
}
```

Here MyClass is a method, not a contructor.

Example: Constructor overloading

```
MyClass{

  MyClass(){
  }
  MyClass(int i){
  }
}
```

Example: Default constructor

To understand the previous rule, create a class without any constructors and instantiate using the default no-parameter constructor.

```
MyClass{

  public static void main(String[] args)

  {

    MyClass mc = new MyClass();

  }

}
```

Now add a constructor with parameter and try to execute the same old instantiation:

```
MyClass{

  MyClass(int i)

  {

  }

  public static void main(String[] args)

  {

    MyClass mc = new MyClass();

  }

}
```

The code will not compile now as we have provided a constructor with parameter and Java does not provide the default constructor. To compile the code you need to either invoke the new constructor as:

```
MyClass mc = new MyClass(1);
```

Or create a no-argument constructor like the default one. To avoid confusion, it is always good have an explicit no-argument constructor.

Example: Invalid Invocation

```
public class MyClass {

  MyClass()

  {

  }

  public static void main(String[] args) {

    MyClass c = new MyClass();

    c.MyClass();

  }

}
```

Compilation will fail because of the line c.MyClass(). Here MyClass() is a constructor name and we cannot call a constructor as c.MyClass(). Constructors can be invoked only during object creation or from other constructors using this keyword.

Example: Not constructor, not method

```
public class MyClass{

  Myclass()

  {

  }

  public static void main(String[] args) {

    MyClass c = new MyClass();

  }

}
```

Compilation will fail. Java identifiers are case sensitive and hence MyClass is not same as Myclass. Myclass is not a valid constructor as the name is not same as that of class and not a valid method as it doesn't have a return type.

Source : http://javajee.com/constructors-in-java