

# 5 Different Ways to Create Objects in Java

A list of five ways to create objects in Java, how they interact with constructors, and an example of how to utilize all of these methods.

by Naresh Joshi  MVB · Jul. 12, 16 · Java Zone · Opinion

**Verify, standardize, and correct the Big 4 + more**– name, email, phone and global addresses – try our Data Quality APIs now at Melissa Developer Portal!

Being Java developers, we usually create lots of objects daily, but we always use dependency management systems e.g. Spring to create these objects. However, there are more ways to create objects, which we will study in this article.

There are five total ways to create objects in Java, which are explained below with their examples followed by bytecode of the line which is creating the object.

Using new keyword	} → constructor gets called
Using newInstance() method of Class class	} → constructor gets called
Using newInstance() method of Constructor class	} → constructor gets called
Using clone() method	} → no constructor call
Using deserialization	} → no constructor call

If you will execute program given in end, you will see method 1, 2, 3 uses the constructor to create the object while 4, 5 doesn't call the constructor to create the object.

# 1. Using new keywords

It is the most common and regular way to create an object and a very simple one also. By using this method we can call whichever constructor we want to call (no-arg constructor as well as parameterized).


# 2. Using newInstance() method of Class class

We can also use the newInstance() method of a Class class to create an object. This newInstance() method calls the no-arg constructor to create the object.

We can create an object by newInstance() in the following way:

--

Or


# 4. Using newInstance() method of Constructor class

Similar to the newInstance() method of Class class, There is one newInstance() method in the java.lang.reflect.Constructor class which we can use to create objects. We can also call parameterized constructor, and private constructor by using this newInstance() method.


Both newInstance() methods are known as reflective ways to create objects. In fact newInstance() method of Class class internally uses newInstance() method of Constructor class. That's why the later one is preferred and also used by different frameworks like Spring, Hibernate, Struts etc. To know differences between both newInstance() methods read Creating objects through Reflection in Java with Example.

## 4. Using clone() method:

Whenever we call clone() on any object, the JVM actually creates a new object for us and copies all content of the previous object into it. Creating an object using the clone method does not invoke any constructor.

To use clone() method on an object we need to implement Cloneable and define the clone() method in it.

```
import java.io.*;
import java.util.*;

class Employee implements Cloneable {
    String name;
    int id;

    Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }

    @Override
    public Employee clone() throws CloneNotSupportedException {
        Employee obj = (Employee) super.clone();
        return obj;
    }
}
```

Java cloning is the most debatable topic in Java community and it surely does have its drawbacks but it is still the most popular and easy way of creating a copy of any object until that object is full filling mandatory conditions of Java cloning. I have covered cloning in details in a 3 article long Java Cloning Series which includes (Java Cloning And Types Of Cloning (Shallow And Deep) In Details With Example, Java Cloning - Copy Constructor Versus Cloning, Java Cloning - Even Copy Constructors Are Not Sufficient), go ahead and read them if you want to know more about cloning.

## 5. Using deserialization:

Whenever we serialize and deserialize an object, the JVM creates a separate object for us. In deserialization, the JVM doesn't use any constructor to create the object.

To deserialize an object we need to implement a Serializable interface in our class.

```
import java.io.*;

class Employee implements Serializable {
    String name;
    int id;

    Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }
}
```

As we can see in the above bytecode snippets, all 4 methods are called and get converted to invokevirtual (object creation is directly handled by these methods) except the first one, which got converted to two calls: one is new and other is invokespecial (call to constructor).

## Example

Let's consider an Employee class for which we are going to create the objects:

```
Employee obj1 = new Employee("John", 1);
Employee obj2 = new Employee("Jane", 2);
Employee obj3 = obj1.clone();
Employee obj4 = obj2.clone();
Employee obj5 = obj1.clone();
```

In the below Java program we are going to create Employee objects in all 5 ways. You can also find the source code at [GitHub](#).

In the below Java program we are going to create Employee objects in an array. You can also find the source code at Github.

This program will give the following output:

Developers! Quickly and easily gain access to the tools and information you need! Explore, test and combine our data quality APIs at **Melissa Developer Portal** – home to tools that save time and boost revenue. Our APIs verify, standardize, and correct the Big 4 + more – name, email, phone and global addresses – to ensure accurate delivery, prevent blacklisting and identify risks in real-time.

## Like This Article? Read More From DZone



Generating Millions of Objects with Random Values [Snippet]



Java Singletons Using Enum




How to Verify Equality Without Equals Method



Free DZone Refcard  
Getting Started With Vaadin 10

Topics: JAVA , CORE JAVA , REFLECTION API

Published at DZone with permission of Naresh Joshi , DZone MVB. [See the original article here.](#)   
Opinions expressed by DZone contributors are their own.

# Java Partner Resources

Learn more about Kotlin

JetBrains



Migrating to Microservice Databases

Red Hat Developer Program



Building Reactive Microservices in Java: Asynchronous and Event-Based Application Design

Red Hat Developer Program



A Reference Guide to Stream Processing

Hazelcast

