Different ways to create objects in Java

Last Updated : 14 Sep, 2023

There are several ways by which we can create objects of a class in java as we all know a class provides the blueprint for objects, you create an object from a class. This concept is under-rated and sometimes proves to be beneficial as this concept is bypassed by many programmers and sometimes even do ask from interview perceptive.

Methods:

There are many different ways to create objects in Java. Let us list them later discussing later taking individually with the help of programs to illustrate internal working by which we can create objects in Java.

1. Using new keyword
2. Using new instance
3. Using clone() method
4. Using deserialization
5. Using newInstance() method of Constructor class

Let us discuss them one by one and implement the same by appending a clean java program for the same.

Method 1: Using new keyword

Using the new keyword in java is the most basic way to create an object. This is the most common way to create an object in java. Almost 99% of objects are created in this way. By using this method we can call any constructor we want to call (no argument or parameterized constructors).

Example

Java

```
// Java program to Illustrate Creation of Object
// Using new keyword

// Main class
class GFG {

    // Declaring and initializing string
    // Custom input string
    String name = "GeeksForGeeks";

    // 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
        GFG obj = new GFG();

        // Print and display the object
        System.out.println(obj.name);
    }
}
```

Output

```
GeeksForGeeks
```

Method 2: Using new instance

If we know the name of the class & if it has a public default constructor we can create an object **Class.forName**. We can use it to create the Object of a Class. Class.forName actually loads the Class in Java but doesn't create any Object. To create an Object of the Class you have to use the new Instance Method of the Class.

Example

Java

```
// Java program to Illustrate Creation of Object
// Using new Instance

// Main class
class GFG {

    // Declaring and initializing string
    String name = "GeeksForGeeks";

    // Main driver method
    public static void main(String[] args)
    {
        // Try block to check for exceptions
        try {

            Class cls = Class.forName("GFG");

            // Creating object of main class
            // using instance method
            GFG obj = (GFG)cls.newInstance();

            // Print and display
            System.out.println(obj.name);
        }

        // Catch block to handle the exceptions

        // Catch block 1
        // Handling ClassNotFound Exception
        catch (ClassNotFoundException e) {

            // Display the exception along with line number
            // using printStacktrace() method
            e.printStackTrace();
        }

        // Catch block 2
        // Handling InstantiationException
        catch (InstantiationException e) {

            e.printStackTrace();
        }

        // Catch block 2
        // Handling IllegalAccessException
        catch (IllegalAccessException e) {

            e.printStackTrace();
        }
```

```
    }
}
```

Output:

```
GeeksForGeeks
```

Method 3: Using clone() method

Whenever clone() is called on any object, the JVM actually creates a new object and copies all content of the previous object into it. Creating an object using the clone method does not invoke any constructor. In order to use the clone() method on an object we need to implement Cloneable and define the *clone() method* in it.

Example

Java

```
// Java program to Illustrate Creation of Object
// Using clone() method

// Main class
// Implementing Cloneable interface
class GFG implements Cloneable {

    // Method 1
    @Override
    protected Object clone()
        throws CloneNotSupportedException
    {
        // Super() keyword refers to parent class
        return super.clone();
    }

    // Declaring and initializing string
    String name = "GeeksForGeeks";

    // Method 2
    // main driver method
    public static void main(String[] args)
    {
        GFG obj1 = new GFG();

        // Try block to check for exceptions
```

```
        try {

            // Using the clone() method
            GFG obj2 = (GFG)obj1.clone();

            // Print and display the main class object
            // as created above
            System.out.println(obj2.name);
        }

        // Catch block to handle the exceptions
        catch (CloneNotSupportedException e) {

            // Display the exception
            // using printStackTrace() method
            e.printStackTrace();
        }
    }
}
```

Output

```
GeeksForGeeks
```

Note:

- *Here we are creating the clone of an existing Object and not any new Object.*
- *Class need to implement Cloneable Interface otherwise it will throw **CloneNotSupportedException**.*

Method 4: Using deserialization

Whenever we serialize and then deserialize an object, JVM creates a separate object. In **deserialization**, JVM doesn't use any constructor to create the object. To deserialize an object we need to implement the Serializable interface in the class.

Example 1

Java

```java
// Java Program Illustrate Serializing an Object

// Importing input output classes
import java.io.*;

// Main class
// Implementing the Serializable interface
class GFG implements Serializable {

    // Member variables
    private String name;
    GFG(String name)
    {
        // This keyword refers to current object itself
        this.name = name;
    }

    // Main driver method
    public static void main(String[] args)
    {
        // Try block to check for exceptions
        try {
            // Creating object of class in main() method
            GFG d = new GFG("GeeksForGeeks");

            FileOutputStream f
                = new FileOutputStream("file.txt");
            ObjectOutputStream oos
                = new ObjectOutputStream(f);
            oos.writeObject(d);
            oos.close();

            // Freeing up memory resources
            f.close();
        }

        // Catch block to handle the exceptiona
        catch (Exception e) {
            // Display the exception along with line number
            // using printStacktrace() method
            e.printStackTrace();
        }
    }
}
```

Output:

```
GeeksForGeeks
```

Object of DeserializationExample class is serialized using writeObject() method and written to file.txt file.

Example 2

Java

```
// Java Program Illustrate Creation of Object
// Using Deserialization

// Importing input output classes
import java.io.*;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Try block to check for exceptions
        try {

            GFG d;

            // Creating FileInputStream class object
            FileInputStream f
                = new FileInputStream("file.txt");

            // Creating ObjectInputStream class object
            ObjectInputStream oos
                = new ObjectInputStream(f);
            d = (DeserializationExample)oos.readObject();
        }

        // Catch block to handle the exceptions
        catch (Exception e) {

            // Display the exception on console
            // using printStacjtrace() method
            e.printStackTrace();
```

```
        }

        System.out.println(d.name);
    }
}
```

Output:

```
GeeksForGeeks
```

Method 5: Using newInstance() method of the constructor class

This is similar to the newInstance() method of a class. There is one newInstance() method in the **java.lang.reflect.Constructor class** which we can use to create objects. It can also call the 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 internally uses newInstance() method of Constructor class.

Example

Java

```
// Java program to illustrate creation of Object
// using newInstance() method of Constructor class

// Importing required classes from java.lang package
import java.lang.reflect.*;

// Main class
class GFG {

    // Member variables of this class
    private String name;

    // Constructor of this class
    GFG() {}

    // Method 1
    // To set name ofthe string
    public void setName(String name)
    {
        // This method refers to current object itself
```

```
        this.name = name;
    }

    // Main driver method
    public static void main(String[] args)
    {
        // Try block to check for exceptions
        try {
            Constructor<GFG> constructor
                = GFG.class.getDeclaredConstructor();

            GFG r = constructor.newInstance();

            // Custom passing
            r.setName("GeeksForGeeks");
            System.out.println(r.name);
        }

        // Catch block to handle the exceptions
        catch (Exception e) {

            // Display the exception on console
            // using printStackTrace() method
            e.printStackTrace();
        }
    }
}
```

Output:

```
GeeksForGeeks
```

Start your **Java programming** journey today with our **Java Programming Online Course,** designed for both beginners and advanced learners. With self-paced lessons covering everything from **basic syntax to advanced concepts**, you'll gain the skills needed to excel in the world of programming.

Take the **Three 90 Challenge**! Complete **90% of the course** in **90 days**, and **earn a 90% refund.** Track your progress and stay