

[Home](#) [Programming](#) [Java](#) [Web](#) [Databases](#) [Academic](#) [Management](#) [Quality](#) [Telecom](#) [More...](#)



Java Basics

- [Java - Home](#)
- [Java - Overview](#)
- [Java - Environment Setup](#)
- [Java - Basic Syntax](#)
- [Java - Object & Classes](#)
- [Java - Basic Datatypes](#)
- [Java - Variable Types](#)
- [Java - Modifier Types](#)
- [Java - Basic Operators](#)
- [Java - Loop Control](#)
- [Java - Decision Making](#)
- [Java - Numbers](#)
- [Java - Characters](#)
- [Java - Strings](#)
- [Java - Arrays](#)
- [Java - Date & Time](#)
- [Java - Regular Expressions](#)
- [Java - Methods](#)
- [Java - Files and I/O](#)
- [Java - Exceptions](#)

Java Object Oriented

- [Java - Inheritance](#)
- [Java - Overriding](#)
- [Java - Polymorphism](#)

Java - Serialization

Advertisements



[Previous Page](#)

[Next Page](#)

Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.

After a serialized object has been written into a file, it can be read from the file and deserialized. The type information and bytes that represent the object and its data can be used to recreate the object in memory.

Most impressive is that the entire process is JVM independent, meaning an object can be serialized on one platform and deserialized on an entirely different platform.

Classes **ObjectInputStream** and **ObjectOutputStream** are high-level streams that contain methods for serializing and deserializing an object.

The **ObjectOutputStream** class contains many write methods for writing various data types. One method in particular stands out:

```
public final void writeObject(Object x) throws IOException
```

The above method serializes an **Object** and sends it to the output stream. The **ObjectInputStream** class contains the following method for deserializing an object:

```
public final Object readObject() throws IOException,  
ClassNotFoundException
```

This method retrieves the next **Object** out of the stream and deserializes it. The return value you will need to cast it to its appropriate data type.

To demonstrate how serialization works in Java, I am going to use the **Employee** class that I introduced early on in the book. Suppose that we have the following **Employee** class, which implements the **Serializable** interface:

```
public class Employee implements java.io.Serializable
{
    public String name;
    public String address;
    public transient int SSN;
    public int number;
    public void mailCheck()
    {
        System.out.println("Mailing a check to " + name
                           + " " + address);
    }
}
```

Java - Abstraction

Java - Encapsulation

Java - Interfaces

Java - Packages

Java Advanced

Java - Data Structures

Java - Collections

Java - Generics

Java - Serialization

Java - Networking

Java - Sending Email

Java - Multithreading

Java - Applet Basics

Java - Documentation

Java Useful Resources

Java - Quick Guide

Java - Library Classes

Java Useful Resources

Java - Examples

Selected Reading

Developer's Best Practices

Effective Resume Writing

Computer Glossary

Who is Who

Notice that for a class to be serialized successfully, two conditions must be met:

The class must implement the `java.io.Serializable` interface.

All of the fields in the class must be serializable. If a field is not serializable, it is marked as `transient`.

If you are curious to know if a Java Standard Class is serializable or not, check the documentation. The test is simple: If the class implements `java.io.Serializable`, then it is serializable; otherwise, it is not.

Serializing an Object:

The `ObjectOutputStream` class is used to serialize an Object. The following program instantiates an `Employee` object and serializes it to a file.

When the program is done executing, a file named `employee.ser` is created. The program does not generate any output, but study the code and try to determine what the program is doing.

Note: When serializing an object to a file, the standard convention in Java is to give the file a `.ser` extension.

```
import java.io.*;

public class SerializeDemo
{
    public static void main(String [] args)
    {
        Employee e = new Employee();
        e.name = "Reyan Ali";
        e.address = "Phokka Kuan, Ambehta Peer";
        e.SSN = 11122333;
        e.number = 101;
        try
        {
            FileOutputStream fileOut =
                new FileOutputStream("/tmp/employee.ser");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(e);
            out.close();
            fileOut.close();
            System.out.printf("Serialized data is saved in /tmp/employee.ser");
        } catch (IOException i)
        {
            i.printStackTrace();
        }
    }
}
```

Deserializing an Object:

The following `DeserializeDemo` program deserializes the `Employee` object created in the `SerializeDemo` program. Study the program and try to determine its output:

```
import java.io.*;

public class DeserializeDemo
{
    public static void main(String [] args)
    {
        Employee e = null;
        try
        {
            FileInputStream fileIn = new FileInputStream("/tmp/employee.ser");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            e = (Employee) in.readObject();
        } catch (IOException i)
        {
            i.printStackTrace();
        }
    }
}
```

```
e = (Employee) in.readObject();
in.close();
fileIn.close();
} catch (IOException i)
{
    i.printStackTrace();
    return;
} catch (ClassNotFoundException c)
{
    System.out.println("Employee class not found");
    c.printStackTrace();
    return;
}
System.out.println("Deserialized Employee...");
System.out.println("Name: " + e.name);
System.out.println("Address: " + e.address);
System.out.println("SSN: " + e.SSN);
System.out.println("Number: " + e.number);
}
```

This would produce the following result:

```
Deserialized Employee...
Name: Reyan Ali
Address: Phokka Kuan, Ambehta Peer
SSN: 0
Number: 101
```

Here are following important points to be noted:

The try/catch block tries to catch a `ClassNotFoundException`, which is declared by the method. For a JVM to be able to deserialize an object, it must be able to find the class. If the JVM can't find a class during the deserialization of an object, it throws a `ClassNotFoundException`.

Notice that the return value of `readObject()` is cast to an `Employee` reference.

The value of the SSN field was 11122333 when the object was serialized, but because it is transient, this value was not sent to the output stream. The SSN field of the deserialized object is 0.

[Previous Page](#)

[Print Version](#)

[PDF Version](#)

[Next Page](#)

Advertisements



Online Java Classes

Easy Online Java Courses.
Free 10 Day Trial. Sign Up
Now!



[ASP.NET](#) | [jQuery](#) | [AJAX](#) | [ANT](#) | [JSP](#) | [Servlets](#) | [log4j](#) | [iBATIS](#) | [Hibernate](#) | [JDBC](#) | [Struts](#) | [HTML](#)

Copyright © 2014 by tutorialspoint. All Rights Reserved.