CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

Serialization

Overview

1. Introduction
2. Serializable interface
3. Deserialization in Java
4. Why do we need Object Serialization in Java?
5. How to Perform Object Serialization/Deserialization?
6. Serialization of Static & Transient Variables
7. Non-serializable Data Fields
8. Example: Employee class
9. Object Serialization & Inheritance (IS-A relationship)
10. Object Serialization & Aggregation (HAS-A relationship)
11. Java Serialization with Array or Collections
12. Common Uses of Serialization in Java
13. Can we customize the serialization process in Java?

Introduction

- **Serialization in Java** is the process of writing the state of an object to a byte stream.
- In other words, it is the process by which we can store (or save) the state of an object by converting it to a byte stream.
- Once Java object state is converted into byte stream, it can be saved or stored into a hard disk, socket, file, or send over a network.
- At a later time, we may restore (or retrieve) these object states by using the process of deserialization.
- The class whose objects need to be serialized must implement the `Serializable` interface of `java.io` package. Then we pass objects to `ObjectOutputStream`, which is connected to a file, and store objects to a file.
- If any one of the objects is not serializable, it will throw an exception named *`NotSerializableException`*.

Serializable Interface

- `Serializable` interface is a marker interface that defines no members. It does not have any method also. Since it has no methods, we do not need to add additional code in the class that implements `Serializable` interface.

- `Serializable` interface is used to mark class objects as serializable so that they can be written into a file. If a class is serializable then all of its subclasses are also serializable.

- By default, String class and all the wrapper classes implement `Serializable` interface.

- So, if we want to send the state of an object over a network or file, we must implement `Serializable` interface.

- If `Serializable` interface is not implemented by a class, storing that class objects into a file will generate `NotSerializableException`.

Deserialization in Java

- The process of converting byte stream (data) generated through serialization to object is called **deserialization** in java. In simple words, the reverse operation of serialization is called deserialization.

- Once the state of objects is stored in a file, they can be retrieved by reading back objects from the file and used as and when needed.

- Both serialization and deserialization process is platform-independent. So, we can serialize an object in a platform and deserialize it in different platforms.

Why do we need Object Serialization in Java?

1. So far, we have created programs where we have stored only text into files and retrieved the same text from the files. These text files are useful when we do not want to perform any calculations on the data.

Suppose we want to store some structured data in files. For example, we want to store employee details like **employee** name (String type), id number (int type), salary (float type), and date of joining job (Date type) in a file.

This data is well structured and of different types. To store such well-structured data in a file, use serialization. We need to create a class **Employee** with instance variables name, id, salary, doj.

The code is as follows:

Why do we need Object Serialization in Java?

```
import java.io.Serializable;
import java.util.Date;

public class Employee implements Serializable {
   String name;
   int id;
   float salary;
   Date doj;

   Employee(String name, int id, float salary, Date doj) {
       this.name = name;
       this.id = id;
       this.salary = salary;
       this.doj = doj;
   }
}
```

In the above code, `Employee` class implements `Serializable` interface. Therefore, its object states can be converted into a byte stream.

Why do we need Object Serialization in Java?

2. Suppose we want to store an `ArrayList` object. To perform it, we need to store all the elements in the list. Each element is an object that may contain other objects. So, it would be a very tiresome process.

To make it easy, Java provides a built-in mechanism to automate the process of writing the state of objects. This process is known as objects serialization in Java, which is implemented by `ObjectOutputStream`.

In contrast, the process of reading the state of objects is known as objects deserialization in java, which is implemented by `ObjectInputStream`.

How to Perform Object Serialization/Deserialization?

`Serializable` interface must be implemented by a class whose objects are to be stored in a file. Follow all the important steps below to store (write) objects in a file:

1. First, connect `objfile.dat` file to `FileOutputStream.` It will write data into `objfile.dat` file.

```
FileOutputStream fos = new FileOutputStream("objfile.dat");
```

2. Then, connect `FileOutputStream` to `ObjectOutputStream` by code below:

```
ObjectOutputStream oos = new ObjectOutputStream(fos);
```

3. Now, call `writeObject()` method of `ObjectOutputStream` to write objects to `FileOutputStream`, which stores them into `objfile.dat` file.

Writing (i.e. storing) objects into a file like this is called ***objects serialization in java***.

How to Perform Object Serialization/Deserialization?

The reverse process in which objects are retrieved (i.e. reading) back from a file is called **objects *deserialization***. To read objects from `objfile.txt file`, follow all the steps below:

1. Connect `objfile.dat` file to `FileInputStream`. It will read objects from `objfile.dat` file. The code is as:

```
FileInputStream fis = new FileInputStream("objfile.dat");
```

2. Connect `FileInputStream` to `ObjectInputStream` to retrieve objects from `FileInputStream`.

```
ObjectInputStream ois = new ObjectInputStream(fis);
```

3. Now, call `readObject()` method of `ObjectInputStream` class to read objects by code below:

```
// Here, Employee is class that implements Serializable interface.
Employee e = (Employee) ois.readObject();
```

How to Perform Object Serialization/Deserialization?

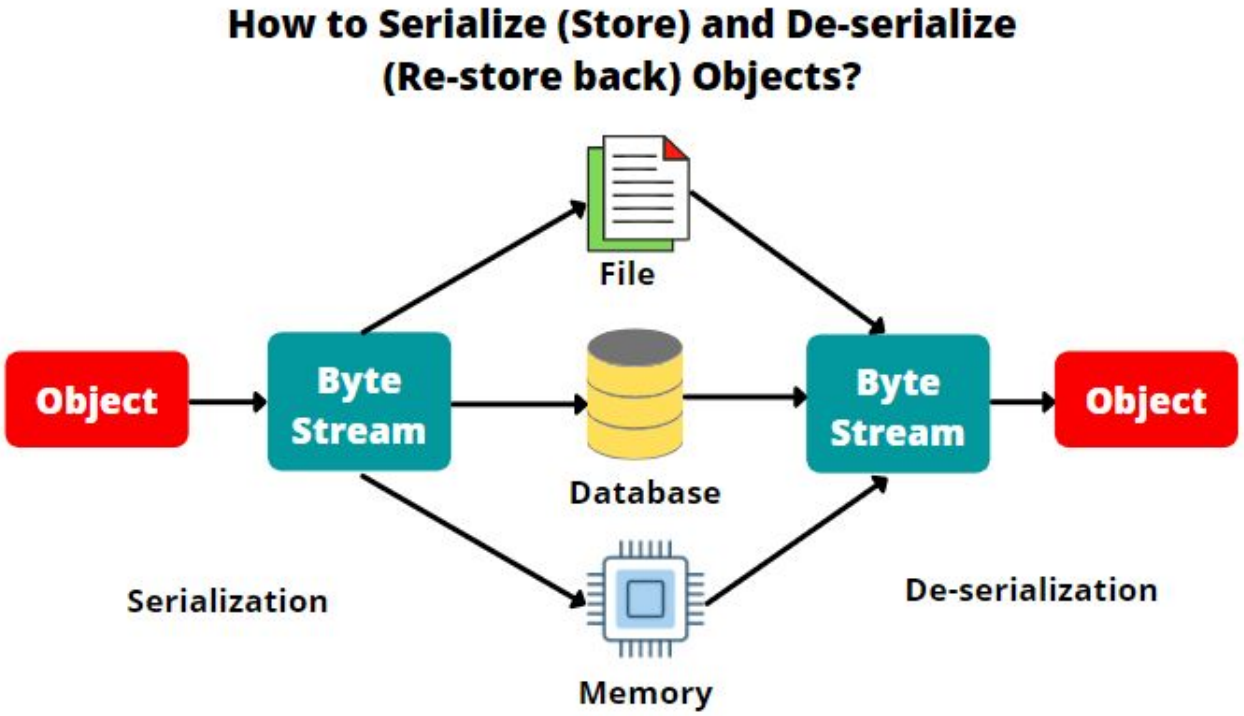

Fig: Serialization vs De-serialization process

Serialization of Static and Transient Variables

Any `static` and `transient` variables declared inside a class cannot be serialized.

For example:

```
static int x = 30;
transient String str = "myPassword";
```

These variables cannot be stored in a file. Static is the part of class, not object.

Key point:

If you don’t want to serialize a data member of a class, declare it with a `transient` keyword.

Non-serializable Data fields

Suppose a class that implements `Serializable` interface also contains non-serializable object data fields.

Can non-serializable data fields be serialized?

The answer is **No**.

To enable the object **not** to be serialized, declare these data fields with `transient` keyword to tell JVM to ignore them when writing objects to the byte stream.

Consider the following example below:

Non-serializable Data fields

```
public class A implements Serializable {
   private int x;
   private static float y;
   private transient B b = new B();
}

class B {    } // B is not serializable.
```

When an object of class `A` is serialized, only variable `x` will be serialized.

Variable `y` will **not** be serialized because it is a `static` variable.

The variable `b` will **not** be serialized because it is marked `transient`. If `b` is **not** marked `transient`, a `java.io.NotSerializableException` would occur.

Example 1: Serialize Employee

In this example program, we will serialize objects of `Employee` class. The `writeObject()` method of `ObjectOutputStream` class provides the functionality to serialize the states of object (i.e. object data fields). We will store the states of the object in the file named `employeefile.dat.`

Look at the source code step by step to understand better.

Example 1: Serialize Employee

```
import java.io.Serializable;
import java.util.Date;

public class Employee implements Serializable {
   String name;
   int id;
   double salary;
   Date doj;

   Employee(String name, int id, double salary, Date doj) {
       this.name = name;
       this.id = id;
       this.salary = salary;
       this.doj = doj;
   }

   public String toString(){
        return String.format("Name:%s, id:%d, salary:$%f, hired:(%s)",name,id,salary,doj);
   }

}
```

Example 1: Serialize Employee

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.Date;

public class SerializeEmployee {
   public static void main(String[ ] args) throws IOException {
       // Create an object of Employee class.
       Employee emp = new Employee("Ted", 12345, 300.00, new Date());

       // Create an object of FileOutputStream class to connect employeefile.dat file.
       FileOutputStream fos = new FileOutputStream("./src/employeefile.dat");

       // Create an object of ObjectOutputStream class to connect with fos.
       ObjectOutputStream oos = new ObjectOutputStream(fos);

       // Call writeObject() method of ObjectOutputStream class.
       oos.writeObject(emp);
       oos.flush();
       oos.close();
       System.out.println("Serialization is done successfully...");
   }
}
```

Example 1: Read/Write a Student object

Output:

```
Serialization is done successfully...
```

Example 2: Deserialize Employee

Let’s create a program where we will retrieve by reading back the data (objects) from the `employeefile.dat` file and displays them on the console.

Example 2: Deserialize Employee

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class DeserializeEmployee {
   public static void main(String[] args) throws IOException, ClassNotFoundException {
       // Create an object of FileInputStream class to connect employeefile.dat file.
       FileInputStream fis = new FileInputStream("./src/employeefile.dat");

       // Create an object of ObjectInputStream to connect with fis.
       ObjectInputStream ois = new ObjectInputStream(fis);

       Object obj = ois.readObject(); // Reading objects.
       Employee e = (Employee)obj;    // Converting to Employee.

       System.out.println(e);
   }
}
```

Example 2: Deserialize Employee

Output:

```
Name:Ted, id:12345, salary:$300.000000, hired:(Mon Jul 18 07:55:43 CDT 2022)
```

Objects Serialization with Inheritance (Is-A relationship)

If a class implements `Serializable` interface, all its subclasses will also be serializable.
Look at the following example given below:

```
import java.io.Serializable;

class Person implements Serializable {
   int id;
   String name;

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

```
class Student extends Person {
   String course;
   int fee;

   public Student(int id, String name, String course, int fee) {
       super(id, name);
       this.course = course;
       this.fee = fee;
   }
}
```

Now we can serialize objects of `Student` class that extends the `Person` class which is `Serializable`.
Superclass properties are inherited to subclasses so if superclass is `Serializable`, subclass would also be.

Objects Serialization & Aggregation (HAS-A Relationship)

If a class has a reference to another class, all the references must be `Serializable`. Otherwise, the serialization process will not be done. In such a case, at runtime, an exception named `NotSerializableException` will be thrown.

```
import java.io.Serializable;

public class Resident implements Serializable {
   String name;
   Address address; // Has-A relationship.

   public Resident(String name) {
       this.name = name;
   }
}
```

```
class Address {
   String addressLine, city, state;
   public Address(String street, String city, String state) {
       this.addressLine = street;
       this.city = city;
       this.state = state;
   }
}
```

Since `Address` is not `Serializable`, we can **not** serialize the instance of `Resident` class.

Key point: All objects within an object must be `Serializable`.

Java Serialization with Array or Collection

An array or collection is serializable if all its elements are serializable. If any element is **not** serializable, the serialization process will be failed.

Example 3: Serialize/Deserialize Array

Let’s take a simple example program where we will store an array of five int values and an array of four strings, writes it to a file and reads them back from the file, and then display them on the console.

Look at the source code.

Example 3: Serialize/Deserialize Array

```
//Imports from IO for object serialization & file processing
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
```

Example 3: Serialize/Deserialize Array

```
public class SerializeArray {
   public static void main(String[] args) throws FileNotFoundException, IOException, ClassNotFoundException {
       int[ ] numbers = {10, 20, 30, 40, -50};
       String[ ] strings = {"John", "Ted", "Deep", "Kim"};

       // Create an output stream for file objfile.dat.
       ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("./src/arrayfile.dat"));

       // Write arrays to the object output stream.
       oos.writeObject(numbers); // write serialized array to file.
       oos.writeObject(strings); // write serialized array to file.

       oos.flush(); // flushing object output stream.
       oos.close(); // closing object output stream.

       // Create an input stream for file objfile.dat.
       ObjectInputStream ois = new ObjectInputStream(new FileInputStream("./src/arrayfile.dat"));

       int[ ] newNumbers = (int[])(ois.readObject()); // Reading array back from the file.
       String[ ] newStrings = (String[])(ois.readObject()); // Reading array back from the file.
       ois.close();

       // Display arrays contents.
       for (int i = 0; i < newNumbers.length; i++)
           System.out.print(newNumbers[i] + " "); // Displaying array contents.
       System.out.println();

       for (int i = 0; i < newStrings.length; i++)
           System.out.print(newStrings[i] + " "); // Displaying array contents.
   }
}
```

Example 3: Serialize/Deserialize Array

Output:

```
10 20 30 40 -50
John Ted Deep Kim
```

Common Use of Serialization in Java

The advantages of using serialization process in java are as follows:

1. When an object is to be sent over the network, objects need to be serialized.

2. If the state of an object is to be saved into a hard disk, or file, objects need to be serialized.

Can we customize the serialization process in Java?

Yes, we can customize or control the serialization process by implementing the `Externalization` interface.

This interface contains two methods: `readExternal()` and `writeExternal()`. Implement these two methods and write logic to control the serialization process.

END