Lecture - 19

Object Serialization

Arpit Rana

21st April 2022

Serialization

Object serialization allows an object to be transformed into a sequence of bytes that can later be re-created (deserialized) into the original object.

- After deserialization, the object has the same state as it had when it was serialized, barring any data members that were not serializable.
- This mechanism is generally known as *persistence*.

Non-Serializable Objects Example

```
import java.io.Serializable;

//public class Wheel implements Serializable {                    // (1)
public class Wheel {                                              // (1a)
  private int wheelSize;

  Wheel(int ws) { wheelSize = ws; }

  public String toString() { return "wheel size: " + wheelSize; }
}
```

```
import java.io.Serializable;

public class Unicycle implements Serializable {                   // (2)
  private Wheel wheel;                                            // (3)
//transient private Wheel wheel;                                  // (3a)

  Unicycle (Wheel wheel) { this.wheel = wheel; }

  public String toString() { return "Unicycle with " + wheel; }
}
```

Non-Serializable Objects Example

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

public class SerialClient {

  public static void main(String args[])
                      throws IOException, ClassNotFoundException {
    SerialClient demo = new SerialClient();
    demo.writeData();
    demo.readData();
  }
```

Non-Serializable Objects Example

```
void writeData() throws IOException {                                    // (4)
  // Set up the output stream:
  FileOutputStream outputFile = new FileOutputStream("storage.dat");
  ObjectOutputStream outputStream = new ObjectOutputStream(outputFile);

  // Write the data:
  Wheel wheel = new Wheel(65);
  Unicycle uc = new Unicycle(wheel);
  System.out.println("Before writing: " + uc);
  outputStream.writeObject(uc);

  // Close the stream:
  outputStream.flush();
  outputStream.close();
}
```

Non-Serializable Objects Example

```
  void readData() throws IOException, ClassNotFoundException {     // (5)
    // Set up the input streams:
    FileInputStream inputFile = new FileInputStream("storage.dat");
    ObjectInputStream inputStream = new ObjectInputStream(inputFile);

    // Read data.
    Unicycle uc = (Unicycle) inputStream.readObject();

    // Write data on standard output stream.
    System.out.println("After reading: " + uc);

    // Close the stream.
    inputStream.close();
  }
}
```

Non-serializable Objects

If we run the program with the following declarations for the `Wheel` and the `Unicycle` classes, where both classes are serializable:

```
class Wheel implements Serializable {                          // (1)
  private int wheelSize;
  ...
}

class Unicycle implements Serializable {                       // (2)
  private Wheel wheel;                                         // (3)
  ...
}
```

we get the following output, showing that both serialization and deserialization were successful:

```
Before writing: Unicycle with wheel size: 65
After reading: Unicycle with wheel size: 65
```

Non-serializable Objects

If we make the `wheel` field of the `Unicycle` class *transient*, (3a):

```
class Wheel implements Serializable {                          // (1)
  private int wheelSize;
  ...
}

class Unicycle implements Serializable {                       // (2)
  transient private Wheel wheel;                               // (3a)
  ...
}
```

we get the following output, showing that the `wheel` field of the `Unicycle` object was not serialized:

```
Before writing: Unicycle with wheel size: 65
After reading: Unicycle with null
```

As noted earlier, static fields are not serialized, as these are not part of the state of an object.

Non-serializable Objects

If the class Wheel is not serializable, (1a):

```
class Wheel {                                                    // (1a)
  private int wheelSize;
  ...
}

class Unicycle implements Serializable {                         // (2)
  private Wheel wheel;                                           // (3)
  ...
}
```

we get the following output when we run the program, i.e., a `Unicycle` object can*not* be serialized because its constituent `Wheel` object is not serializable:

```
>java SerialClient
Before writing: Unicycle with wheel size: 65
Exception in thread "main" java.io.NotSerializableException: Wheel
  at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1156)
  at java.io.ObjectOutputStream.defaultWriteFields(ObjectOutputStream.java:1509)
  at java.io.ObjectOutputStream.writeSerialData(ObjectOutputStream.java:1474)
  at java.io.ObjectOutputStream.writeOrdinaryObject(ObjectOutputStream.java:1392)
  at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1150)
  at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:326)
  at SerialClient.writeData(SerialClient.java:25)
  at SerialClient.main(SerialClient.java:12)
```

Customizing Object Serialization

The class of the object must implement the Serializable interface if we want the object to be serialized.

- If this object is a compound object, then all its constituent objects must also be serializable.

Customizing Object Serialization

Customizing object serialization is the mechanism to deal with objects or values that should not or cannot be serialized by the default methods of the object streams.

The basic idea behind the scheme is to use default serialization as much as possible, and provide “hooks” in the code to call specific methods.

For example:

While writing to the stream:

```
oos.defaultWriteObject();
oos.writeInt(wheel.getWheelSize());
```

While reading from the stream:

```
ois.defaultReadObject();
int wheelSize = ois.readInt();
wheel = new Wheel(wheelSize);
```

Customized Serialization Example

```
public class Wheel {                                                    // (1a)
  private int wheelSize;

  Wheel(int ws) { wheelSize = ws; }

  int getWheelSize() { return wheelSize; }

  public String toString() { return "wheel size: " + wheelSize; }
}
```

Customized Serialization Example

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

public class Unicycle implements Serializable {                    // (2)
  transient private Wheel wheel;                                   // (3a)

  Unicycle(Wheel wheel) { this.wheel = wheel; }

  public String toString() { return "Unicycle with " + wheel; }
```

Customized Serialization Example

```
  private void writeObject(ObjectOutputStream oos) {              // (3b)
    try {
      oos.defaultWriteObject();
      oos.writeInt(wheel.getWheelSize());
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  private void readObject(ObjectInputStream ois) {                // (3c)
    try {
      ois.defaultReadObject();
      int wheelSize = ois.readInt();
      wheel = new Wheel(wheelSize);
    } catch (IOException e) {
      e.printStackTrace();
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
}
```

Serialization and Inheritance

The inheritance hierarchy of an object also determines what its state will be after it is deserialized.

- An object will have the same state at deserialization as it had at the time it was serialized if all its superclasses are also serializable.
- This is because the normal object creation procedure using constructors is not run during deserialization.

Serialization and Inheritance

If any superclass of an object is not serializable,

- then the normal creation procedure using constructors is run, starting at the first non-serializable superclass, all the way up to the Object class.
- This means that the state at deserialization might not be the same as at the time the object was serialized, because super constructors run during deserialization may have initialized the object's state.

Customized Serialization Example

```
// A superclass
// public class Person implements Serializable {              // (1a)
public class Person  {                                        // (1b)
  private String name;

  Person() {}                                                 // (2)
  Person(String name) { this.name = name; }
  public String getName() { return name; }
}
```

Customized Serialization Example

```
import java.io.Serializable;

//public class Student extends Person {                          // (1a)
public class Student extends Person implements Serializable {  // (1b)

  private long studNum;

  Student(String name, long studNum) {
    super(name);
    this.studNum = studNum;
  }

  public String toString() {
    return "Student state(" + getName() + ", " + studNum + ")";
  }
}
```

Customized Serialization Example

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

public class SerialInheritance {
  public static void main(String[] args)
                    throws IOException, ClassNotFoundException {
    SerialInheritance demo = new SerialInheritance();
    demo.writeData();
    demo.readData();
  }

  void writeData() throws IOException {                          // (1)
    // Set up the output stream:
    FileOutputStream outputFile = new FileOutputStream("storage.dat");
    ObjectOutputStream outputStream = new ObjectOutputStream(outputFile);

    // Write the data:
    Student student = new Student("Pendu", 1007);
    System.out.println("Before writing: " + student);
    outputStream.writeObject(student);

    // Close the stream:
    outputStream.flush();
    outputStream.close();
  }
```

Customized Serialization Example

```
    void readData() throws IOException, ClassNotFoundException {   // (2)
      // Set up the input stream:
      FileInputStream inputFile = new FileInputStream("storage.dat");
      ObjectInputStream inputStream = new ObjectInputStream(inputFile);

      // Read data.
      Student student = (Student) inputStream.readObject();

      // Write data on standard output stream.
      System.out.println("After reading: " + student);

      // Close the stream.
      inputStream.close();
    }
}
```

Next lecture -
Course Evaluation Viva