Session - 11

Serialization in JAVA

Storing Objects/Structures in Files

- Many programs need to save information between program runs

- Alternately, we may want one program to save information for later use by another program

Serialization of Objects

- Java provides a way to save objects directly
- Saving an object with this approach is called *serializing* the object
- Serialization in other languages can be *very* difficult, because objects may contain references to other objects. Java makes serialization (almost) easy
- Any object that you plan to serialize must implement the `Serializable` interface

Conditions for serializability

- If an object is to be serialized:
  - The class must be declared as public
  - The class must implement **Serializable**
  - If the object is a sub type of another class, the parent class must have a no-argument constructor
  - All fields of the class must be **serializable**: either primitive types or **serializable** objects

Implementing Serializable

- To "implement" an interface means to define all the methods declared by that interface, but…
- The **Serializable** interface does not define any methods!
  - Question: What possible use is there for an interface that does not declare any methods?
  - Answer: **Serializable is used as flag to tell Java it needs to do extra work with this** class
  - When an object implements **Serializable, its state is converted to a byte stream to be written to a file so that the byte stream can be converted back into a copy of the object when it is read from the file.**

The `Serializable` Interface

- The **Serializable** interface is a marker interface.
- It has no methods, so you don't need to add additional code in your class except that the class must implement **Serializable**.
- You must also import **java io** which contains all the streams needed.

The Object Streams

- You need to use the **ObjectOutputStream** class for storing objects and the **ObjectInputStream** class for restoring objects.

- These two classes are built upon several other classes.

A Serializable Version of a Circle Class

```
package ch09.circles;

import java.io.*;

public class SCircle implements Serializable
{
   public int xValue;
   public int yValue;
   public float radius;
   public boolean solid;
}
```

A Program to Save a `SCircle` Object

```
import java.io.*;
import ch09.circles.*;

public class SaveSCircle
{
  public static void main(String[] args) throws IOException
  {
    SCircle c1 = new SCircle();
    c1.xValue = 5;
    c1.yValue = 3;
    c1.radius = 3.5f;
    c1.solid = true;

    ObjectOutputStream out = new ObjectOutputStream(new
                                    FileOutputStream("objects.dat"));
    out.writeObject(c1);
    out.close();
  }
}
```

Cont….

```
public class FlightRecord2 implements Serializable
    {

       private String flightNumber;     // ex. = AA123
      private String origin;          // origin airport; ex. = Khi
      private String destination;      // destination airport; ex. = Isl
      private int numPassengers;       // number of passengers
      private double avgTicketPrice;   // average ticket price

//  Constructor
  public FlightRecord2 (String startFlightNumber, String startOrigin, String
    startDestination, int startNumPassengers,  double startAvgTicketPrice )
      {
       flightNumber = startFlightNumber;

         origin = startOrigin;
       destination = startDestination;

        numPassengers = startNumPassengers;
       avgTicketPrice = startAvgTicketPrice;
      }
```

Flight Record class

```
public String toString( )
  {
    return "Flight " + flightNumber
           + ": from " + origin
           + " to " + destination
           + "\n\t" + numPassengers + " passengers";
  }
  // accessors, mutators, and other methods …
}
```

```
import java.io;

public class WritingObjects
{
  public static void main( String [] args )
  {
   // instantiate the objects
   FlightRecord2 fr1 = new FlightRecord2( "AA31", “Khi", “Lhr",200, 13500 );
   FlightRecord2 fr2 = new FlightRecord2( "CO25", “Lhr", “Isl",225, 11500);
   FlightRecord2 fr3 = new FlightRecord2( "US57", “Khi", “Isl"175, 17500 );

   try
   {    FileOutputStream fos = new FileOutputStream( objects.dat );

    ObjectOutputStream oos = new ObjectOutputStream( fos );

    // write the objects to the file
    oos.writeObject( fr1 );
    oos.writeObject( fr2 );
    oos.writeObject( fr3 );

    // release resources associated with the objects file
    oos.close( );
   }
```

```
catch( FileNotFoundException e )
      {
        System.out.println( "Unable to write to objects" );
      }
     catch( IOException e )
      {
         ioe.printStackTrace( );
  }
  }
 }
```

Saving Hierarchical Objects

- Ensure that each of the objects involved implements the `Serializable` interface

```
import java.io*;
public class SPoint implements Serializable
{
    public int xValue;  // this is for example only
    public int yValue;
}
import java.io*;
public class SNewCircle implements Serializable
{
public SPoint location;
public float radius;
public boolean soldi;
}
// initialize location's xValue and yValue
```

Reading Objects from a file

- **ObjectInputStream** reads objects from a file. The **readObject()** method reads the next object from the file and returns it.

- Because it returns a generic object, the returned object must be cast to the appropriate class.

- When the end of file is reached, it throws an **EOFException** versus when reading from a text file where a **null String** is returned.

Reading objects from a file

```
ObjectInputStream objectIn = new
ObjectInputStream( new BufferedInputStream(
        new FileInputStream(fileName)));

myObject = (itsType) objectIn.readObject( );
//  some code
objectIn.close( );
```

```
import java.io.;

public class GetCircle
  {
    public static void main( String [] args )
     {
       SCircle s2 = new SCircle();
        ObjectInputStream in =new ObjectInputStream( new
        BufferedInputStream(new FileInputStream(Objects.dat)));
       try {
              s2   = (SCircle) in.readObject();
            }
     catch (Exception e) { System.out.println (“ Error in reading “ + e)
     }
    System.out.println( “ The value of xvalue is “ + s2.xValue;
    System.out.println( “ The value of yvalue is “ + s2.yValue;
   }
  in.close();
 }
```

```
import java.io.ObjectInputStream;

public class ReadingObjects
{
  public static void main( String [] args )
  {
   try
   {
     FileInputStream fis = new FileInputStream( objects.dat );
     ObjectInputStream ois = new ObjectInputStream( fis );
     try
     {
       while ( true )
       {
         // read object, type cast returned object to FlightRecord
         FlightRecord2 temp = ( FlightRecord2 ) ois.readObject( );
         // print the FlightRecord2 object read
         System.out.println( temp );
       }
     } // end inner try block
     catch( EOFException eofe )
     {
      System.out.println( "End of the file reached" );
     }
```

```
catch( ClassNotFoundException e )
    {
     System.out.println( cnfe.getMessage( ) );
    }
    finally
    {
     System.out.println( "Closing file" );
     ois.close( );
    }
  } // end outer try block
  catch( FileNotFoundException e )
  {
    System.out.println( "Unable to find objects" );
  }
  catch( IOException ioe )
  {
     ioe.printStackTrace( );
  }
 }
}
```

Reading Objects from a file.

- The while loop runs until the end of file is reached and an exception is thrown

- Control goes to the catch block and will always execute in a normal program run.

- The **EOFException** catch block must come before **IOException** as it is subclass of **IOException**. Otherwise the program will not produce the correct stack trace.

Output from reading objects

----jGRASP exec: java ReadingObjects

Flight AA31: from Khi to Lhr
200 passengers; average ticket price: 13500
Flight CO25: from Lhr to Isl
225 passengers; average ticket price: 11500
Flight US57: from Khi to Isl
175 passengers; average ticket price: 17500

End of the file reached // EOF exception caught
Closing file

Example-Serialization

```
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);
   }
}
```

Cont….

```
import java.io.*;
public class SerializeDemo
{
  public static void main(String [] args)
  {
    Employee e = new Employee();
    e.name = “Muhammad Shafan";
    e.address = “DHA, Karachi";
    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.println("Serialized data is saved in /tmp/employee.ser");
    }catch(IOException i)
    {
       i.printStackTrace();
    }
  }
}
```

Example-Deserialization

```
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();
         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);
   }
}
```