CS133: Developing Programming Principles

Lecture 23

Binary Files, Serialization

Class `File`

The class **`File`** is used to represent filenames in a system independent fashion:

```
new File(String fileName);
```

e.g.,

```
new File("C:\\hello.txt");
new File("/u/claclark/hi");
```

Methods in Class **`File`**

```
public boolean exists()
public boolean canRead()
public boolean canWrite()
public boolean delete()
public long length()
```


Binary Files

Binary files store data in the same format as the computer's internal memory.

Reading/writing binary files avoids expensive conversions to/from text formats.

Binary Files

- Reading/Writing/Appending
- Object Serialization

Example

```
public class Coordinate {
  private int row, col;

  public Coordinate(int row, int col)
  {
    this.row = row;
    this.col = col;
  }

  public int getRow() { return row; }

  public int getCol() { return col; }
```

Example - Saving

```
public void save(String fileName)
    throws IOException
{
  ObjectOutputStream out =
    new ObjectOutputStream(
      new FileOutputStream(fileName)
    );
  out.writeInt(row);
  out.writeInt(col);
  out.close();
}
```


Example - Restoring

```
public Coordinate(String fileName)
    throws IOException
{
  ObjectInputStream in =
    new ObjectInputStream(
      new FileInputStream(fileName)
    );
  row = in.readInt();
  col = in.readInt();
  in.close();
}
```

Example

```
public static void main(String[] args)
    throws Exception
{
  Coordinate x, y;
  String fileName = "C:\\test.dat";
  x = new Coordinate(4, 5);
  x.save(fileName);
  y = new Coordinate(fileName);
  System.out.println(y.getRow());
  System.out.println(y.getCol());
}
}
```


Output to Binary Files

```
ObjectOutputStream out =
  new ObjectOutputStream(
    new FileOutputStream(String fileName)
  );
```

May throw an **`IOException`** if the file can't be created.

ObjectOutPutStream

Methods:

```
public void writeInt(int n)
public void writeDouble(double x)
public void writeBoolean(boolean b)
```

All these methods throw **`IOException`** if an error occurs. (However, this is unusual.)

ObjectOutPutStream

`public void flush()`
Forces an actual physical write of any buffered data to the file.

`public void close()`
Flushes the output stream and closes the file.

Input from Binary Files

```
ObjectInputStream in =
  new ObjectInputStream(
    new FileInputStream(String fileName)
  );
```

May throw **`FileNotFoundException`** or some other **`IOException`**.

`FileNotFoundException`

Thrown when a file of the specified name does not exist.

It's often worth handling **`IOException`** and **`FileNotFoundException`** in separate **`catch`** statements.

ObjectInputStream

Methods:

```
public int readInt()
public double readDouble()
public boolean readBoolean()
```

All these methods may throw an **`EOFException`**, or some other **`IOException`**, if an error occurs.

EOFException

Thrown when an **`ObjectInputStream`** method tries to read beyond the end of the file.

May be used with a **`catch`** statement to break out of an input loop.

Example

```
public static int intCount(String fileName)
   throws IOException
{
  int n = 0;
  ObjectInputStream in = null;
  try {
    in = new ObjectInputStream(
      new FileInputStream(fileName));
    while (true) {
      in.readInt();
      n++;
    }
  } catch (EOFException e) {
    in.close();
  }
  return n;
}
```


Example

```
try {
  String fileName = "C:\\test.dat";
  ObjectOutputStream out =
    new ObjectOutputStream(
      new FileOutputStream(fileName));
  out.writeInt(10);
  out.writeInt(11);
  out.writeInt(2005);
  out.close();
  System.out.println(intCount(fileName));
} catch (IOException e) {
}
```

Object I/O

The **`writeObject`** method of the **`ObjectOutputStream`** class outputs objects to binary files.

The **`readObject`** method of the **`ObjectInputStream`** class inputs objects from binary files.

Object I/O

To read or write objects, the class must be declared as serializable:

```
import java.io.Serializable;
public class Name implements Serializable
{
   // Normal Java class
}
```

Example – Object I/O

```
import java.io.*;

public class Rectangle implements Serializable {
  private int height, width;

  public Rectangle(int height, int width) {
    this.height = height;
    this.width = width;
  }

  // etc.
}
```


Example - Output

```
Rectangle r0 = new Rectangle(2, 3);
Rectangle r1 = new Rectangle(8, 10);
String fileName = "C:\\test.dat";

try {
  ObjectOutputStream out =
      new ObjectOutputStream(
          new FileOutputStream fileName)
       );
  out.writeObject(r0);
  out.writeObject(r1);
  out.close();
} catch (IOException e) {
}
```

Example - Input

```
Rectangle r0, r1;
String fileName = "C:\\test.dat";

try {
  ObjectInputStream in =
      new ObjectInputStream(
          new FileInputStream(fileName)
      );
   r0 = (Rectangle)in.readObject();
   r1 = (Rectangle)in.readObject();
   in.close();
} catch (Exception e) {
}
```


Summary

- Class **`File`**
- Binary files
- Object I/O

ERROR: undefined
OFFENDING COMMAND:

STACK: