

# CS1092: Object-Oriented Programming with Java

## Binary Input and Output with Files

Howard Barringer

Room KB2.20/22: email: [howard.barringer@manchester.ac.uk](mailto:howard.barringer@manchester.ac.uk)

February 2005

## Supporting and Background Material

- Copies of key slides (will appear in Resource Centre later today)
- Chapter 18 of the JTL book (follows this presentation) — [lecture notes](#)
- Recommended reading books for course
  - Chapter 9 (9.4) of Savitch “JAVA: An introduction to Computer Science and Programming”, 3rd Edition
  - Chapter 17 (17.11) of Liang “Introduction to JAVA Programming”, International Edition

## Supporting and Background Material

- Copies of key slides (will appear in Resource Centre later today)
- Chapter 18 of the JTL book (follows this presentation) — [lecture notes](#)
- Recommended reading books for course
  - Chapter 9 (9.4) of Savitch “JAVA: An introduction to Computer Science and Programming”, 3rd Edition
  - Chapter 17 (17.11) of Liang “Introduction to JAVA Programming”, International Edition

## Supporting and Background Material

- Copies of key slides (will appear in Resource Centre later today)
- Chapter 18 of the JTL book (follows this presentation) — [lecture notes](#)
- Recommended reading books for course
  - Chapter 9 (9.4) of Savitch “JAVA: An introduction to Computer Science and Programming”, 3rd Edition
  - Chapter 17 (17.11) of Liang “Introduction to JAVA Programming”, International Edition

# Outline

## Binary Input and Output

ObjectOutputStream & ObjectInputStream

A Graphing Task

Summary



# Outline

## Binary Input and Output

ObjectOutputStream & ObjectInputStream

A Graphing Task

Summary



# Textual versus Binary Output

- Why use text files ....



# Textual versus Binary Output

- Why use text files ....
- What use binary files ....



## Simple Binary Output

- For “binary” output, we use the `ObjectOutputStream` class

## Simple Binary Output

- For “binary” output, we use the `ObjectOutputStream` class
- Methods such as `writeInt`, `writeDouble`, `writeBoolean`, ..., are provided



## Simple Binary Output

- For “binary” output, we use the `ObjectOutputStream` class
- Methods such as `writeInt`, `writeDouble`, `writeBoolean`, ..., are provided
- To write to a file, set up a `FileOutputStream` for the given filename



## Simple Binary Output

- For “binary” output, we use the `ObjectOutputStream` class
- Methods such as `writeInt`, `writeDouble`, `writeBoolean`, ..., are provided
- To write to a file, set up a `FileOutputStream` for the given filename
- Then wrap with an `ObjectOutputStream` to be able to use the appropriate writing methods

## Let's generate some data

```
import java.io.ObjectOutputStream;  
import java.io.FileOutputStream;  
import java.io.PrintWriter;  
import java.io.FileWriter;  
  
public class GenerateRandomData {  
    public static void main(String [] args) throws Exception {  
  
    }  
}
```

## Let's generate some data

```
import java.io.ObjectOutputStream;  
import java.io.FileOutputStream;  
import java.io.PrintWriter;  
import java.io.FileWriter;  
  
public class GenerateRandomData {  
    public static void main(String [] args) throws Exception {  
        ObjectOutputStream outputStream =  
            new ObjectOutputStream( new FileOutputStream( args[2] ) );  
  
        outputStream.close();  
    }  
}
```

## Let's generate some data

```
import java.io.ObjectOutputStream;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.io.FileWriter;

public class GenerateRandomData {
    public static void main(String [] args) throws Exception {
        ObjectOutputStream outputStream =
            new ObjectOutputStream( new FileOutputStream( args[2] ) );
        PrintWriter outputWriter =
            new PrintWriter( new FileWriter( args[3] ) );

        outputStream.close();
        outputWriter.close();
    }
}
```

## Let's generate some data

```
import java.io.ObjectOutputStream;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.io.FileWriter;

public class GenerateRandomData {
    public static void main(String [] args) throws Exception {
        ObjectOutputStream outputStream =
            new ObjectOutputStream( new FileOutputStream( args[2] ) );
        PrintWriter outputWriter =
            new PrintWriter( new FileWriter( args[3] ) );

        // : --- see next slide
        outputStream.close();
        outputWriter.close();
    }
}
```



## The generation part

```
int numberOfRandoms = Integer.parseInt(args[0]);
int xStep = Integer.parseInt(args[1]);
outputStream.writeInt(numberOfRandoms);
outputStream.writeInt(xStep);
outputWriter.println(numberOfRandoms);
outputWriter.println(xStep);

double randomNumber;
for (int i = 0; i < numberOfRandoms; i++)
{
    outputStream.writeInt(i*xStep);
    outputStream.writeDouble(randomNumber = Math.random()*1000000);
    outputWriter.println("" + i*xStep + " " + randomNumber);
}
```

## The generation part

```
int numberOfRandoms = Integer.parseInt(args[0]);
int xStep = Integer.parseInt(args[1]);
outputStream.writeInt(numberOfRandoms);
outputStream.writeInt(xStep);
outputWriter.println(numberOfRandoms);
outputWriter.println(xStep);

double randomNumber;
for (int i = 0; i < numberOfRandoms; i++)
{
    outputStream.writeInt(i*xStep);
    outputStream.writeDouble(randomNumber = Math.random()*1000000);
    outputWriter.println("" + i*xStep + " " + randomNumber);
}
```

## The generation part

```
int numberOfRandoms = Integer.parseInt(args[0]);
int xStep = Integer.parseInt(args[1]);
outputStream.writeInt(numberOfRandoms);
outputStream.writeInt(xStep);
outputWriter.println(numberOfRandoms);
outputWriter.println(xStep);

double randomNumber;
for (int i = 0; i < numberOfRandoms; i++)
{
    outputStream.writeInt(i*xStep);
    outputStream.writeDouble(randomNumber = Math.random()*1000000);
    outputWriter.println("" + i*xStep + " " + randomNumber);
}
```

## Actual Comparison of Output

Add the following four lines after the close of files

```
File first = new File(args[2]);  
File second = new File(args[3]);  
System.out.println(first.length()  
    + " bytes written to file " + args[1]);  
System.out.println(second.length()  
    + " bytes written to file " + args[2]);
```

## Actual Comparison of Output

Add the following four lines after the close of files

```
File first = new File(args[2]);  
File second = new File(args[3]);  
System.out.println(first.length()  
    + " bytes written to file " + args[1]);  
System.out.println(second.length()  
    + " bytes written to file " + args[2]);
```

And we obtain



## Actual Comparison of Output

Add the following four lines after the close of files

```
File first = new File(args[2]);  
File second = new File(args[3]);  
System.out.println(first.length()  
    + " bytes written to file " + args[1]);  
System.out.println(second.length()  
    + " bytes written to file " + args[2]);
```

And we obtain

```
$ java GenerateRandomData 100 5 random.dat random.txt
```



## Actual Comparison of Output

Add the following four lines after the close of files

```
File first = new File(args[2]);  
File second = new File(args[3]);  
System.out.println(first.length()  
    + " bytes written to file " + args[1]);  
System.out.println(second.length()  
    + " bytes written to file " + args[2]);
```

And we obtain

```
$ java GenerateRandomData 100 5 random.dat random.txt  
1219 bytes written to file random.dat  
2472 bytes written to file random.txt
```

```
$
```



## Size difference

A `double` requires 8 bytes of storage

When printed, a `double` may use around 17 to 20 characters

An `int` in the range 0..500 takes 4 bytes but typically 3 bytes textually

Hence, we get an approximate halving in file size for the binary version.





## Reading it back

- Data written to a file through an `ObjectOutputStream`



## Reading it back

- Data written to a file through an `ObjectOutputStream`
- can be read back from that file through an `ObjectInputStream`



## Reading it back

- Data written to a file through an `ObjectOutputStream`
- can be read back from that file through an `ObjectInputStream`
- Data must be read back in the same order as written.



## Reading it back

- Data written to a file through an `ObjectOutputStream`
- can be read back from that file through an `ObjectInputStream`
- Data must be read back in the same order as written.
- For the above, we can use the `readInt` and `readDouble` methods.



## Reading it back

- Data written to a file through an `ObjectOutputStream`
- can be read back from that file through an `ObjectInputStream`
- Data must be read back in the same order as written.
- For the above, we can use the `readInt` and `readDouble` methods.
- These methods throw an `EOFException` when they attempt to read past the end of file.

# Outline

## Binary Input and Output

ObjectOutputStream & ObjectInputStream

A Graphing Task

Summary

# Graph Data Manipulation

Our task is to read & process time-stamped sensor data.

We can assume the data in the form generated above:-

1. an integer: the number of time-stamped data items
2. an integer: the usual time step between data items
3. a series of pairs of values:
  - 3.1 an integer time-stamp value
  - 3.2 a double for the sensor data value

Some time-stamped data may be missing. One required task is thus to interpolate for the missing data items.

# The Graph Class

```
public class Graph {  
  
    private int maxDataPointPairs;  
    private int noOfDataPointPairs;  
    private int [] xValues;  
    private double [] yValues;  
    private int normalXStep;  
  
    public Graph(int size, int xStep) {  
        noOfDataPointPairs = size;  
        maxDataPointPairs = (int)Math.round(size*1.1);  
        xValues = new int [maxDataPointPairs];  
        yValues = new double [maxDataPointPairs];  
        normalXStep = xStep;  
    } // Graph  
  
    :  
} // class Graph
```





## Reading raw values into the graph

```
public void readValues(ObjectInputStream rawValues)
    throws GraphException
{
    int i = 0;

    while (true) {
        xValues[i] = rawValues.readInt();
        yValues[i] = rawValues.readDouble();
        i++;
    }
}
```



## Reading raw values into the graph

```
public void readValues(ObjectInputStream rawValues)
    throws GraphException
{
    int i = 0;
    try {
        while (true) {
            xValues[i] = rawValues.readInt();
            yValues[i] = rawValues.readDouble();
            i++;
        }
    } catch (IndexOutOfBoundsException e){
        throw new GraphException("Too much graph data input"); }
}
```



## Reading raw values into the graph

```
public void readValues(ObjectInputStream rawValues)
    throws GraphException
{
    int i = 0;
    try {
        while (true) {
            xValues[i] = rawValues.readInt();
            yValues[i] = rawValues.readDouble();
            i++;
        }
    } catch (IndexOutOfBoundsException e){
        throw new GraphException("Too much graph data input"); }
    catch (EOFException e){
        if (i != noOfDataPointPairs)
            throw new GraphException("Insufficient graph data input"); }
}
```



## Reading raw values into the graph

```
public void readValues(ObjectInputStream rawValues)
    throws GraphException
{
    int i = 0;
    try {
        while (true) {
            xValues[i] = rawValues.readInt();
            yValues[i] = rawValues.readDouble();
            i++;
        }
    } catch (IndexOutOfBoundsException e){
        throw new GraphException("Too much graph data input"); }
    catch (EOFException e){
        if (i != noOfDataPointPairs)
            throw new GraphException("Insufficient graph data input"); }
    catch (IOException e){
        throw new GraphException("Problem reading graph data input"); }
}
```



## More powerful writing methods

- The `ObjectOutputStream` class can also be used to write arrays of primitive data



## More powerful writing methods

- The `ObjectOutputStream` class can also be used to write arrays of primitive data



## More powerful writing methods

- The `ObjectOutputStream` class can also be used to write arrays of primitive data
- In fact, it can be used for any object that is serializable - but we do not go there today



## More powerful writing methods

- The `ObjectOutputStream` class can also be used to write arrays of primitive data
- In fact, it can be used for any object that is serializable - but we do not go there today





## More powerful writing methods

- The `ObjectOutputStream` class can also be used to write arrays of primitive data
- In fact, it can be used for any object that is serializable - but we do not go there today
- We show the writing (and reading) of array data - it's so simple!

## Saving the Graph

```
public void saveGraph(ObjectOutputStream output)
    throws GraphException
{
    try {
        output.writeInt(normalXStep);

    }

    catch (IOException e) {
        throw new GraphException("Problem saving graph data");
    }
}
```

## Saving the Graph

```
public void saveGraph(ObjectOutputStream output)
    throws GraphException
{
    try {
        output.writeInt(normalXStep);
        output.writeObject(xValues);
    }

    catch (IOException e) {
        throw new GraphException("Problem saving graph data");
    }
}
```

## Saving the Graph

```
public void saveGraph(ObjectOutputStream output)
    throws GraphException
{
    try {
        output.writeInt(normalXStep);
        output.writeObject(xValues);
        output.writeObject(yValues); }
    catch (IOException e) {
        throw new GraphException("Problem saving graph data");
    }
}
```

## Reading a saved Graph

```
public void readGraph(ObjectInputStream input)
    throws GraphException
{

    normalXStep = input.readInt();

}
```

## Reading a saved Graph

```
public void readGraph(ObjectInputStream input)
    throws GraphException
{

    normalXStep = input.readInt();
    int [] tempXValues = (int [])input.readObject();

}
```

## Reading a saved Graph

```
public void readGraph(ObjectInputStream input)
    throws GraphException
{
    try {
        normalXStep = input.readInt();
        int [] tempXValues = (int [])input.readObject();

    }

    catch (IOException e) {
        throw new GraphException("Problem reading saved graph"); }
    catch (ClassNotFoundException e) {
        throw new GraphException("Problem reading saved graph"
                                   + " --- corrupt data?"); }
}
```

## Reading a saved Graph

```
public void readGraph(ObjectInputStream input)
    throws GraphException
{
    try {
        normalXStep = input.readInt();
        int [] tempXValues = (int [])input.readObject();
        double [] tempYValues = (double [])input.readObject();

    }

    catch (IOException e) {
        throw new GraphException("Problem reading saved graph"); }
    catch (ClassNotFoundException e) {
        throw new GraphException("Problem reading saved graph"
                                   + " --- corrupt data?"); }
}
```



## Reading a saved Graph

```
public void readGraph(ObjectInputStream input)
    throws GraphException
{
    try {
        normalXStep = input.readInt();
        int [] tempXValues = (int [])input.readObject();
        double [] tempYValues = (double [])input.readObject();
        xValues = tempXValues;
        yValues = tempYValues; }
    catch (IOException e) {
        throw new GraphException("Problem reading saved graph"); }
    catch (ClassNotFoundException e) {
        throw new GraphException("Problem reading saved graph"
                                   + " --- corrupt data?"); }
}
```



# A Simple Interpolation

No time now!

## A Simple Interpolation

No time now! — for completeness see forthcoming notes

# Outline

## Binary Input and Output

ObjectOutputStream & ObjectInputStream

A Graphing Task

Summary



## Summary

- Use `ObjectOutputStream` class for binary output of data



## Summary

- Use `ObjectOutputStream` class for binary output of data
- And `ObjectInputStream` for reading data (written via an `ObjectOutputStream`)



## Summary

- Use `ObjectOutputStream` class for binary output of data
- And `ObjectInputStream` for reading data (written via an `ObjectOutputStream`)
- Used writing methods: `writeInt`, `writeDouble` for primitive data



## Summary

- Use `ObjectOutputStream` class for binary output of data
- And `ObjectInputStream` for reading data (written via an `ObjectOutputStream`)
- Used writing methods: `writeInt`, `writeDouble` for primitive data
- With reading methods: `readInt()` returning an `int`, `readDouble()` returning a `double`





## Summary

- Use `ObjectOutputStream` class for binary output of data
- And `ObjectInputStream` for reading data (written via an `ObjectOutputStream`)
- Used writing methods: `writeInt`, `writeDouble` for primitive data
- With reading methods: `readInt()` returning an `int`, `readDouble()` returning a `double`
- Attempt to read past end of file raises an `EOFException`



## Summary

- Use `ObjectOutputStream` class for binary output of data
- And `ObjectInputStream` for reading data (written via an `ObjectOutputStream`)
- Used writing methods: `writeInt`, `writeDouble` for primitive data
- With reading methods: `readInt()` returning an `int`, `readDouble()` returning a `double`
- Attempt to read past end of file raises an `EOFException`
- Can write an array of primitive data with `writeObject` method



## Summary

- Use `ObjectOutputStream` class for binary output of data
- And `ObjectInputStream` for reading data (written via an `ObjectOutputStream`)
- Used writing methods: `writeInt`, `writeDouble` for primitive data
- With reading methods: `readInt()` returning an `int`, `readDouble()` returning a `double`
- Attempt to read past end of file raises an `EOFException`
- Can write an array of primitive data with `writeObject` method
- And read an array of primitive data with `readObject()` — but must cast to the appropriate type



## Summary

- Use `ObjectOutputStream` class for binary output of data
- And `ObjectInputStream` for reading data (written via an `ObjectOutputStream`)
- Used writing methods: `writeInt`, `writeDouble` for primitive data
- With reading methods: `readInt()` returning an `int`, `readDouble()` returning a `double`
- Attempt to read past end of file raises an `EOFException`
- Can write an array of primitive data with `writeObject` method
- And read an array of primitive data with `readObject()` — but must cast to the appropriate type
- An `ClassNotFoundException` exception (as well as `IOException`) may be raised with `readObject`