Simple File Input And Output

Types of Java Files

Simple File Output in Java

Simple File Input in Java

Writing and reading objects to and from file (by implementing the Serializable interface)

Storing Information On Files

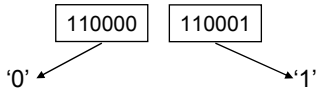
Types of files

- Text files
- Binary files

Text Files

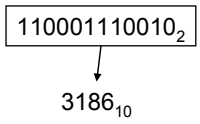
Text files

- Every 8 bits represents a character
- e.g., ‘0’ = 48, ‘1’ = 49



Binary files

- Includes all other types of files (it’s a catch all term for non-binary files)
- e.g., Interpret the bit pattern as a 16 bit unsigned short



James Tam

(You Should Know Why You Can Get Garbage In Text Editors)

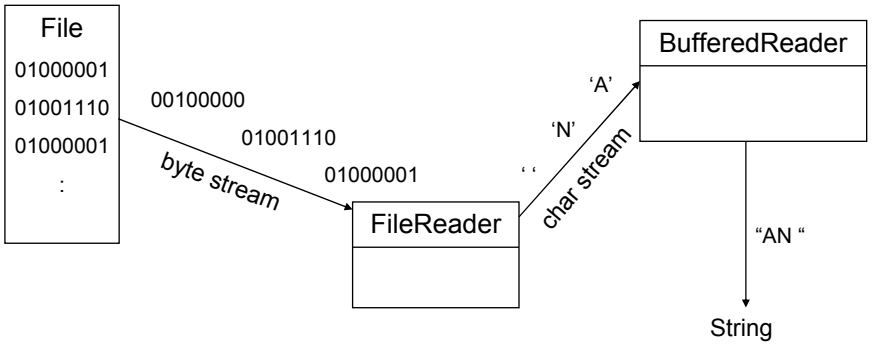
ÿØÿà □JFIF □□ □ □ ÿÛ C □□□□□□□□□□□□ □□□
□

□□□□□□□□□□□□□□□□ $.' ",#□□(7),01444□'9=82<.342ÿÛ C□
□
□□

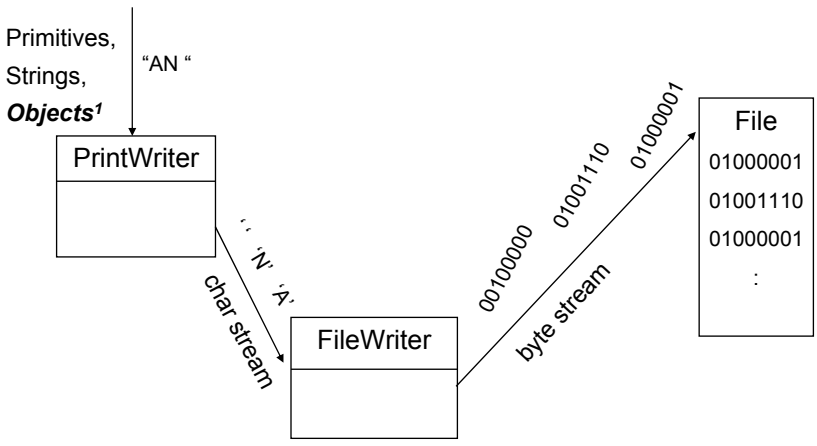
□2!□!2222222222222222222222222222222222222222222222222222ÿÀ □□□ □ □□"
□□□□□□ÿÄ □ □□□□ □□□□□□□ □ÿÄ G□ □□□□□□□□□□□□□□□□□□□□
□□□□!1A□□Qa"2q□□‘¡#3B±Á□RÑáð□□$SbrñCs¸46c’5%&t¢²ÿÄ □□ □□□□
□□ □□□□ÿÄ *□ □□□□□□□□□ □ □ □□□□□□!1A□□"2Q□a#3q□B□ÿÚ □□□
□□□□ ? ò□öwá»?□xïMÐïäž;[¯7{ÀÀ8Û□¸Á Žª;T□□Ñ
ñ□‹it‹›{ê^gó7E`ñ$Í¶6a´ÉB□FNOL¸ãœU□öuâ+O
xóMÖïäžKko7z@ ¹Ý□ À$□¬;ÑŸ□jÞ□ðwŒ´Ív+Ýbõ˜mžA$M§E□ÚÑ:d□=²rËÁÇ□ç
Œ□n¸Œ[èÏi¾ ñ□¯§Ã}c§ù□Ïæþ□LÑ¤—>PlžTlÅåÚ:i□œŽ£□ßîG^¿²?µ?³ÿ áÿ
øÝžt~wá÷mó¼□Þg—ŸâÛŒscœsŽø.4□cÀ¾□Óu¤Ô£:9»□–
□?â|æ□¤»□Ùá†□Àü}x9©ëV°¯‡l¬Ž-â□Ck£¦›ø
K□□´Î¹_2C»â□□+³œ□»□ÒÞx.□□´™Z°<âo□x’
¸j/a□€Ù.¡±.¡y□□ØÄ¾R±}œòvàsœ`Ô□îO^□²?µ?³ÿ áÿ øížt~wá÷mó¼□Þg—
ŸâÛŒscœs^¤¦Ç¤□ý;ÏþÐÿ wþTýç÷¾o—
ß¯µeüW¨øoYÓ4æ»]Qu□3G‹K,□„b&’2vÊÒ□NÝ¬I@€ç□Ã-Hæ„□"¥¥Ë□m□MOÁ>"Ò4
é ´ÿ

James Tam

Reading Text Input From A File



Writing Text Output To A File



1 By objects we of course mean references to objects

An Example Of Simple Input And Output

The full example can be found in Unix in the directory:
/home/233/examples/fileIO/example1

Class IntegerWrapper

```
public class IntegerWrapper
{
   private int num;

   public IntegerWrapper ()
   {
      num = (int) (Math.random() * 100);
   }
   public void setNum (int newValue)
   {
      num = newValue;
   }
   public int getNum ()
   {
      return num;
   }
}
```

Class SimpleIO

```
import java.io.*;

public class SimpleIO
{
   public static void main (String [] argv)
   {
      IntegerWrapper iw1 = new IntegerWrapper ();
      IntegerWrapper iw2 = new IntegerWrapper ();
      String filename = "data";
      PrintWriter pw;
      FileWriter fw;
      BufferedReader br;
      FileReader fr;
```


Class SimpleIO (2)

```
      try
      {
         fw = new FileWriter (filename);
         pw = new PrintWriter (fw);

         System.out.println("Written to file: " + iw1.getNum());
         pw.println(iw1.getNum());
         System.out.println("Written to file: " + iw2.getNum());
         pw.println(iw2.getNum());
         fw.close();
```

Class SimpleIO (3)

```
            fr = new FileReader(filename);
            br = new BufferedReader(fr);
            System.out.println("Read from file: " + br.readLine());
            System.out.println("Read from file: " + br.readLine());
            fr.close();
        }
```


Class SimpleIO (4)

```
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

Reading Until The End-Of-File Is Reached

```
String filename = "data";
BufferedReader br = null;
FileReader fr = null;
String temp = null;

try
{
      fr = new FileReader(filename);
      br = new BufferedReader(fr);
      temp = br.readLine ();
      while (temp != null)
      {
            :     :     :
           temp = br.readLine ();
      }
}
:     :     :
```


Checking For More Specific Error Types

```
String filename = null;
BufferedReader br;
FileReader fr;
boolean fileError = true;

while (fileError == true)
{
   try
   {
      System.out.print("Enter name of input file: ");
      filename = Console.in.readWord();

      fr = new FileReader(filename);
      br = new BufferedReader(fr);
            :          :
      fr.close ();
      fileError = false;
   }
```

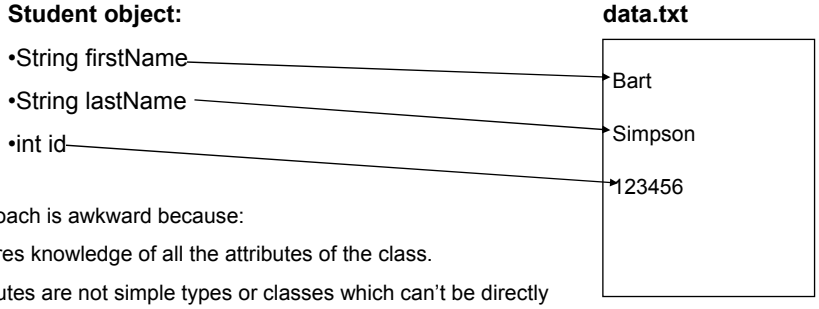
Checking For More Specific Error Types (2)

```
    catch (FileNotFoundException e)
    {
       System.out.println("File called " + filename +
                          " not in the current  directory");
    }
    catch (IOException e)
    {
       System.out.println("General file input error occured.");
       e.printStackTrace();
    }
}
```


Writing Objects Out To File: "The Hard Way"

The full example can be found in Unix in the directory:
/home/233/examples/fileIO/example2

Each field is written out to a file individually



This approach is awkward because:

1. It requires knowledge of all the attributes of the class.
2. If attributes are not simple types or classes which can't be directly written to file the non-writable attribute must be broken down and written to file on a field-by basis.
3. Some attributes may have to be parsed or converted.

The Driver Class

```
public class Driver
{
   public static void main (String [] args)
   {
      final String FILENAME = "data.txt";
      PrintWriter pw;
      FileWriter fw;
      BufferedReader br;
      FileReader fr;
      Student aStudent = new Student("Bart", "Simpson", 123456);
      int tempNum;
      String tempLine;
```


The Driver Class (2)

```
      try
      {
         fw = new FileWriter (FILENAME);
         pw = new PrintWriter (fw);
         pw.println(aStudent.getFirstName());
         pw.println(aStudent.getLastName());
         pw.println(aStudent.getId());
         fw.close();

         fr = new FileReader(FILENAME);
         br = new BufferedReader(fr);
         aStudent.setFirstName(br.readLine());
         aStudent.setLastName(br.readLine());
         tempLine = br.readLine();
         aStudent.setId(Integer.parseInt(tempLine));
         fr.close();

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

The Driver Class (3)

```
        catch (FileNotFoundException e)
        {
           e.printStackTrace();
        }
        catch (IOException e)
        {
           e.printStackTrace();
        }
        catch (NumberFormatException e)
        {
           e.printStackTrace();
        }
    }
}
```


Class Student

```
public class Student
{
   private String firstName;
   private String lastName;
   private int id;

   public Student ()
   {
      firstName = "no name";
      lastName = "no name";
      id = -1;
   }
```

Class Student (2)

```
public Student (String aFirstName,
                String aLastName,
                int anId)
{
   firstName = aFirstName;
   lastName = aLastName;
   id = anId;
}

public String getFirstName () { return firstName; }
public String getLastName () { return lastName; }
public int getId () { return id; }
```

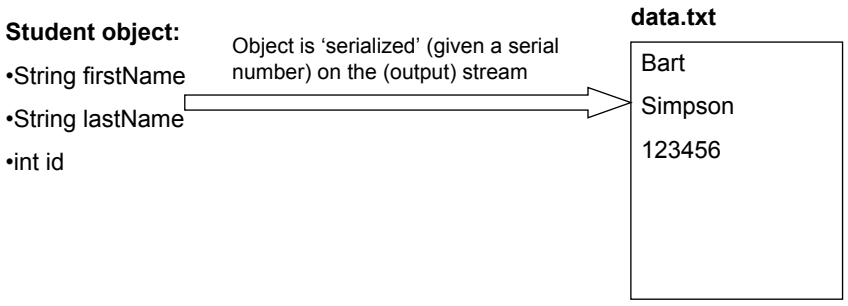

Class Student (3)

```
public void setFirstName (String name) { firstName = name; }
public void setLastName (String name) { lastName = name; }
public void setId (int anId) { id = anId; }
public String toString ()
{
   String s = new String ();
   s = s +
      "First name: " + firstName + "\n" +
      "Last name: " + lastName + "\n" +
      "ID No: " + id + "\n";
   return s;
}
}
```

Writing Objects Out To File: A Better Way

The full example can be found in Unix in the directory:
/home/233/examples/fileIO/example3

Write all the data for the class all at once



Objects of a class can be serialized when the class implements the Serializable interface

The Driver Class

```
public class Driver
{
   public static void main (String [] args)
   {
      final String FILENAME = "data.txt";

      try
      {
         // Write object to file.
         ObjectOutputStream out = new ObjectOutputStream
            (new FileOutputStream(FILENAME));
         Student aStudent = new Student("Bart", "Simpson", 123456);
         out.writeObject(aStudent);
         out.close();
         aStudent = null;
```

The Driver Class (2)

```
            ObjectInputStream in = new ObjectInputStream
                (new FileInputStream(FILENAME));
            aStudent = (Student) in.readObject();
            System.out.println(aStudent);
        }
        catch (FileNotFoundException e)
        {
            e.printStackTrace();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
        }
    }
}
```


The Student Class: Key Difference

```
public class Student implements Serializable
{
    private String firstName;
    private String lastName;
    private int id;

    public Student ()
    {
        firstName = "no name";
        lastName = "no name";
        id = -1;
    }
```

Note: The Data File For Serialized Objects Is In Binary Form

¬í?□sr? Student’±"‹□Ìk□□?□I?□idL?
firstNamet?□Ljava/lang/String;L?□lastNameq?~?□xp?□
â@t?□Bartt? Simpson

Note: Many ‘Container’ Classes Are Serializable

Serializable Containers:

- ArrayList
- LinkedList
- Vector
- :

The effect of having a serializable container class is that the entire container can be serialized

Classes That Don't Implement The Serializable Interface

1. The contents of the class (data) are confidential.
2. The contents of the class is meaningful only while the program runs.

You Should Now Know

How to write to files with Java classes

- FileWriter
- PrintWriter

How to reading text information from files with Java classes

- FileReader
- BufferedReader

How objects can be written to file in a serializable form.