Simple File Input And Output

You will learn how to write to and read from text and serialized files in Java.

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

110000 → **‘0’**   **110001** → **‘1’**

- 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

110001110010_2 → **3186_{10}**

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

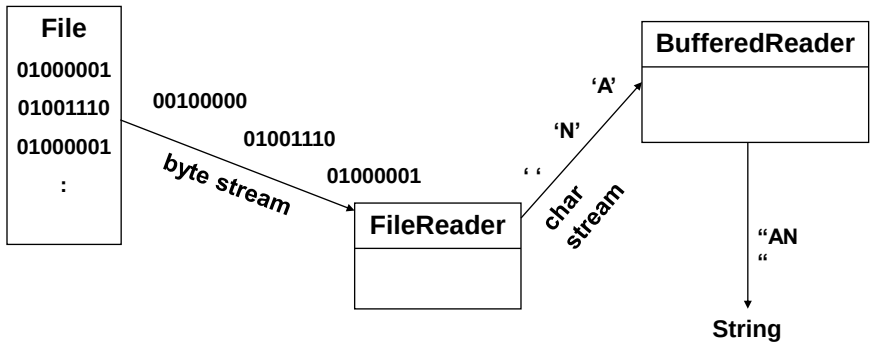
• ÿØÿà JFIF h Z ÿÛ C

• #%$""!&+7/&)4)!"0A149;>>>%.DIC<H7=>;ÿÛ C

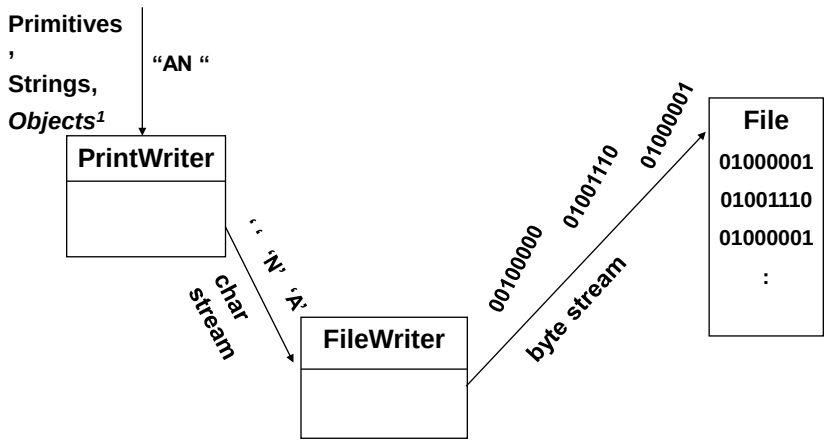
• ;("(;;;ÿÀ ù
" ÿÄ
ÿÄ µ } !1A Qa "q 2‘¡□#B±Á□RÑð$3br,
□□□□□%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyzƒ„…†‡ˆ‰Š’“”•——
˜™š¢£¤¥¦§¨©ª²³´µ¶·¸¹ºÂÃÄÅÆÇÈÉÊÒÓÔÕÖ×ØÙÚáâãäåæçèéêñòóôõö÷øùúÿÄ

ÿÄ µ w !1 AQ aq "2□□B‘¡±Á
#3Rð□brÑ□$4á%ñ□□□□&'()*56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz,ƒ„…†‡ˆ‰Š’“”•–
—˜™š¢£¤¥¦§¨©ª²³´µ¶·¸¹ºÂÃÄÅÆÇÈÉÊÒÓÔÕÖ×ØÙÚâãäåæçèéêòóôõö÷øùúÿÚ
□□ □□□□? îóÅZOÃŠ^õèžxP:{Ðh□† úÐ(¥¦$%(÷¤ö {Ò £¥% ´}(ëF□Z>” v¢ŠJ \ÐM□P□ûÎ“ëAÍ
P□æ,{ÑÚÂ€□4”P□Ñü©)F8 □9žQŸAÍþ´P êii(ö4 ½E Í%.3@ {Q@ hÍ E□ `úQš(¤ :QžzÐ);q@
Et4□ƒÉ¦□Ž´”v¤¤1sëGZNô¢•lÐOhÆE'j ´©M□□½%□¥□ QžzQÞŽ Å □ŒÑE □}E□Š J 3GãG¶(Å□
¢Š _ëGjOÖƒ@(□v Ð□□□œæ—š 3Ç½□bŠ (¢ŽÔ qEè □¥½h£ëÅ □ö¢’–€(ë@é@□□¢Š ;RÒsE -
□žô{R□ hïIÚžb□8£Š)?C□ŽzCÓ/j (Í%-□½hï@úbŽz□□□(□Œþ4QÞ•
J3Eh□QE□□ZJ=úÑ@ÂŒæŠ(□ G½'ô£(□v£<QïF(□ sIGj(□½é(íG|Ð £4□hÅ □v¢Š□□P>´{Ð ?Z(£Z (£¶(â•
□¢Š>´
;u£¥R \Ð ÑIŠ(□´Ru¥üè ÍÑþy¢€
ÑIïGÒ‹Fi;Q@
íIš(üh□ sIõ£ÞŠ □□Qô£µ□Ç4wÉ£¨úQš□ÑÍ□,(ý(?J(aš3é@¤¦□ÑÐÑGãÖƒš□{ÑIŒó@

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

- Location of the online example:
  - /home/233/examples/fileIO/firstExample

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

```
public class SimpleIO
{
    public static void main (String [] argv)
    {
        IntegerWrapper iw1 = new IntegerWrapper ();
        IntegerWrapper iw2 = new IntegerWrapper ();
        String filename = "data.txt";
        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 (3)

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

Reading Until The End-Of-File Is Reached

```
String filename = "data.txt";
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

Location of the full example:
`/home/233/examples/fileIO/secondExample`

Checking For More Specific Error Types (2)

```
public class Driver
{
   public static void main(String [] args) {
      String filename = "data.txt";
      BufferedReader br = null;
      FileReader fr = null;
      String temp = null;
      try {
         fr = new FileReader(filename);
         br = new BufferedReader(fr);
         temp = br.readLine ();
```

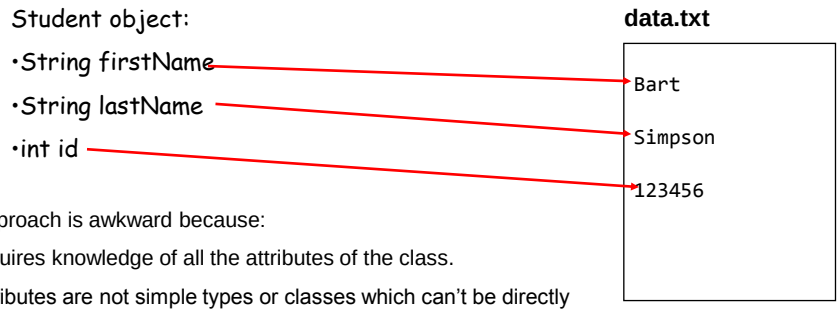
Checking For More Specific Error Types (3)

```
         while (temp != null) {
            System.out.println(temp);
            temp = br.readLine ();
         }
      }
      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"

- Location of the example:
  - /home/233/examples/fileIO/secondExample

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 (2)

```
        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

- Location of the example:
`/home/233/examples/fileIO/thirdExample`

Write all the data for the class all at once

Student object:

- String firstName
- String lastName
- int id

Object is 'serialized' (given a serial number) on the (output) stream

data.txt

```
Bart
Simpson
123456
```

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 whole 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()
    {
        setFirstName("no name");
        setLastName("no name");
        setId(-1);
    }
```

Note: The Data File For Serialized Objects Is Binary

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

Note: Many 'Container' Classes Are Serializable

- Serializable Containers:
  - `ArrayList`
  - `LinkedList`
  - `Vector`
  - **others**

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

Issues With The `Serializable` Interface

1. The contents of the class (data) are confidential. The file is not secure.
2. Some of the contents of the class is meaningful only while the program runs e.g., formatting characteristics like font types, sizes etc. are only used when the program is running and probably shouldn't be serialized and written to file.

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.