COMP132 – Computer Science II
Fall 2015

Homework #7
File I/O
Solutions

1. Imagine that the values in the following variables are to be written to a file:

```
String s = “This is a Test”
int y = 1739;
long x = (long)191256;
double z = 17.25;
float q = (float)1.75;
```

How many bytes will be in the file if the file is:

a. A text file where each value is written using `println`?

(14 + 1) + (4 + 1) + (6 + 1) + (5 + 1) + (4 + 1) = 38 bytes.

The first (…) includes 14 bytes for the characters of the string and one byte for the newline character generated by the `println` method. Each of these characters requires 1 byte because they are all standard ASCII characters. The others are similar with one byte for each digit in the number and one byte for the newline character.

b. A text file where each value is written using `print`?

14 + 4 + 6 + 5 + 4 = 33 bytes.

This is similar to part a, except that no newline characters are included because the `print` method is used instead of the `println` method. In this case each value would be smushed up against the previous one, all on one line. Not ideal formatting, but that wasn’t the point of the question!

C. A binary file where each value is written using the appropriate `DataOutputStream` method (e.g. `writeUTF(s)`, `writeInt(y)`, `writeLong(x)` etc.)?

(2 + 14) + 4 + 8 + 8 + 4 = 40 bytes.

The (2+14) represents the string in UTF format using the initial 2 bytes to indicate the number of bytes in the string and the remaining bytes to represent the characters in the string. Because each of the characters in the string is an ASCII character, each requires only 1 byte (thus 14). Each of the numeric data types uses the same number of bytes as its Java equivalent (4 for `int`, 8 for `long`, 8 for `double` and 4 for `float`).

2. Give the decimal value of each byte that would appear in a text file if it contained the string "`Test`" on the first line and the string "`1 2 3!`" on the second line.

```
84 101 115 116 10 49 32 50 32 51 33
```

Looking the characters up in the ASCII table shows:

```
T = 84
e = 101
s = 115
t = 116
```

The value `10` is the ASCII code for the newline character and causes the following characters (`1 2 3!`) to be displayed on the next line. Looking up the characters 1, 2, 3, space and `!` in the ASCII table gives the remaining values 49, 32, 50, 32, 51 and 33.

3. Give a line or two of code that creates an object that can be used for each of the tasks listed below. You may assume that the files named appear in the working directory. Also, you do not need to include the try/catch statements that would be necessary to handle any checked exceptions that might be generated.

a. Read binary data from a file named "`two.bin`".

```
DataInputStream dis = new DataInputStream(
  new FileInputStream("two.bin"));
```

b. Write binary data to a file named "`three.bin`", overwriting the file if it already exists.

```
DataOutputStream dos = new DataOutputStream(
  new FileOutputStream("three.bin", false));
```

c. Read text from a file named "`four.txt`".

```
Scanner scr = new Scanner(new FileInputStream("four.txt"));
```

d. Write to a text file named "`five.txt`". The file should be appended to if it already exists.

```
PrintWriter pw2 = new PrintWriter(
  new FileOutputStream("five.txt", true));
```

4. Give a snippet of code that accomplishes each of the tasks below. If the code that you write for the given task might throw a checked exception enclose it in a try/catch block that catches the exception. You do not need to catch unchecked exceptions. Hint: Use the JavaDoc in the `Scanner`, `PrintWriter`, `DataOuputStream` and `DataInputStream` to learn what methods are available in each class and what types of exceptions are thrown by each method. Assume the that the following variables have been initialized and are available for use in your code:

- `scr` : a reference to a `Scanner` for the file "`aye.txt`".
- pw: a reference to a `PrintWriter` for the file "`bee.txt`"
- `dos`: a reference to a `DataOutputStream` for the file "`sea.bin`"
- `dis`: a reference to a `DataInputStream` for the file "`dee.bin`"
- p: a `float`.
- q: a `double`.
- `x` : a `short`.
- y: an `integer`.
- `s`: a `String`.

a. Write the value of `s` to a single line of the file `bee.txt`.

```
pw.println(s);
```

b. Read a value from the file `aye.txt` into q.

```
String doubStr = scr.nextLine();
q = Double.parseDouble(doubStr);
```

c. Read consecutive values from `dee.bin` into `p` and `y`.

```
try {
  p = dis.readFloat();
  y = dis.readInt();
}
catch (IOException e) {
  System.out.println("error reading file");
}
```

d. Write the values of `s`, `p`, `q`, `x` and `y` into the file `sea.bin`.

```
try {
  dos.writeUTF(s);
  dos.writeFloat(p);
  dos.writeDouble(q);
  dos.writeShort(x);
  dos.writeInt(y);
}
catch (IOException e) {
  System.out.println("error writing data");
}
```

e. Write the values of `q` and `y` separated by a single space to a line in `bee.txt` and `p` and `x` separated by a single space to the next line.

```
pw.println(q + " " + y);
pw.println(p + " " + x);
```

5. Using the lab macs give the following file paths:

a. An absolute path for a file named *`hw7.docx`* that is in your home directory.

```
/Users/braught/hw7.docx
```

Absolute paths begin with a '/' and list each directory, separated by '/', that must be opened to reach the file. Of course your solution will include your username and not mine!

b. A relative path from your home directory to a file named `hw7-soln.pdf` that is in a folder named *`HW`* that is in your *`Documents`* folder.

```
Documents/HW/hw7-soln.pdf
```

Relative paths do not begin with a '/' and list each directory, beginning from the current directory, that must be opened to reach the file. This path assumes that the folder `Documents` appears in the home directory and that that folder contains a folder named `HW`, which then contains the file `hw7-soln.pdf`.

c. A relative path from the folder containing your eclipse workspace to the `BinaryFileExamples.java` source code.

```
132SampleCode/src/comp132/examples/files/BinaryFileExamples.java
```

d. An absolute path to the file from part c.

```
/Users/braught/Documents/workspace/132SampleCode/src/comp132/examples
/files/BinaryFileExamples.java
```

This path puts the absolute path to the eclipse workspace before the relative path from part c to produce the absolute path.

6. Using "`..`" in a path indicates that the path goes to the parent directory. So for example, if the current directory is `/Users/braught/workspace` then the relative path `../myfile.txt` refers to a file named `myfile.txt` in my home directory. Making use of the "`..`" notation give:

a. three different relative paths to a file named `myfile.txt` in my home directory, assuming that the current directory my home directory

```
myfile.txt
../braught/myfile.txt
../../Users/braught/myfile.txt
Documents/../myfile.txt
Documents/../Documents/../myfile.txt
etc…
```

There are infinitely many of these!

b. two different absolute paths to a file named `myfile.txt` in the `files` folder in my home directory.

```
/Users/braught/files/myfile.txt
/Users/braught/../braught/files/myfile.txt
/Users/../Users/braught/../../Users/braught/files/../files/myfile.txt
etc…
```

There are also infinitely many of these!

6. As we defined in class all files are just a sequence of bytes. This definition has some interesting implications that are explored in this exercise. If a file is really just a sequence of bytes then the meaning of the bytes in a file depends upon how they are interpreted. For example, if the bytes in a file are interpreted as a text file, each byte represents one character using the UTF UNICODE encoding. If instead the bytes in a file are interpreted as a binary file, the bytes can represent different types of values. For example, a value written with `writeInt` takes 4 bytes and when read with `readInt` those 4 bytes will be interpreted as a binary integer value.

We also saw in class that when we opened a text file in a binary file viewer it was happy to interpret the bytes in any way we ask it to (e.g. any sequence of 4 bytes could be interpreted as a binary integer value). Similarly, when we opened a binary file in a text editor, it was happy to interpret each byte as an ASCII character, resulting in some characters that did not display and others that displayed as punctuation or other odd characters.

Given this equivalence we can have some fun!

a. Write a small program that uses a `DataOutputStream` to write the integer value `1330464545` to a file. Then open the file in a text editor.

i. Give the code you used to write the value to the file.

```java
public static void sixA() throws IOException {
  DataOutputStream dos = null;
  try {
    FileOutputStream fos = new FileOutputStream("sixA.bin");
    dos = new DataOutputStream(fos);
    dos.writeInt(1330464545);
  }
  finally {
    dos.close();
  }
}
```

ii. What is displayed in the text editor? Why?

```
OMG!
```

The binary representation of the integer `1330464545` is:

```
01001111 01001101 01000111 00100001
```

i.e. (… $1*2^8 + 1*2^5 + 1*2^0$) = `13304064545`.

The `writeInt` method of `DataOutputStream` writes these 4 bytes into the file. When this file is then opened in the text editor it interprets each of the bytes as an ASCII character code. The 4 bytes that were written to the file along with their decimal values and corresponding ASCII character are:

```
01001111 01001101 01000111 00100001
  (79)     (77)     (71)     (33)
   O        M        G        !
```

b. Open a text editor, type in the characters "`DSON`" and save the file. Write a small program that uses a `DataInputStream` to read an integer (via `readInt`) from the file that you saved.

i. Give the code you used to read the value from the file.

```java
public static void sixB() throws IOException {
  DataInputStream dis = null;
  try {
    FileInputStream fis = new FileInputStream("sixB.txt");
    dis = new DataInputStream(fis);
    int val = dis.readInt();
    System.out.println(val);
  }
  finally {
    dis.close();
  }
}
```

ii. What integer value was read? Why?

```
1146310478
```

When you typed the characters DSON into the text editor it placed the byte corresponding to the ASCII code for each of those characters into the file. Thus, the following 4 bytes were written into the text file:

```
  01000100 01010011 01001111 01001110
    (68)     (83)     (79)     (78)
    'D'      'S'      'O'      'N'
```

Then when the readInt method of the DataInputStream was used to read those four bytes they were interpreted not as ASCII characters but as a base 2 (binary) number. When this value is converted from binary to decimal, its value is `1146310478`.