CM1210: JAVA EXERCISES

Teaching Week: 4 **Laboratory Exercise:** 3

Learning Outcomes

After completing these exercises, you should be able to ...

- write overloaded methods
- use the **Vector** class to store and retrieve objects
- read/write text files

Source Code

The source code for this worksheet is available on Learning Central at:

CM1210 → Learning Materials → Exercises

Method overloading

1. Given a class containing a method:

```
public static void foobar( float a, Date b )
```

which of the following overloaded methods are invalid for the class:

(a)
```
public static void foobar( Date a, float b )
```

(b)
```
public static void foobar( float a, String b )
```

(c)
```
public static int foobar( float a, Date b )
```

(d)
```
public static int foobar( float a, Date b, int c )
```

(e)
```
public static void foobar( float x, Date y )
```

(f)
```
public void foobar( float x, Date y )
```

Verify your answers by implementing each method in the same class.

2. Download the file **Student.java**.

   Add another overloaded constructor to the **Student** class that sets the average mark as well as setting the forename/surname and initialising the student number.

   Since some code is now duplicated in both constructors, create a method **init** to perform the task common to both methods (you will need to decide on appropriate arguments). Both constructors should then call this method. You should test that both constructors work as they are supposed to.

   What access specifier should be used for the method **init**?

   HINT: does client code using the **Student** class need to call the method directly?

   Separating the common code into a single method is good practice.

Vectors

3. Write an application **ReverseInputs** that first allows the user to enter multiple lines of text, and then prints out each line reversed. The user should indicate that they have finished entering text by entering an empty line. An example would be:

```
PLEASE ENTER TEXT. ENTER A BLANK LINE WHEN YOU ARE FINISHED:
The quick brown fox
jumps over the lazy dog

YOUR INPUT REVERSED:
xof nworb kciuq ehT
god yzal eht revo spmuj
```

   You should use the **Vector** class in **java.util** to store each line of the user's input. A **Vector** is a growable array; it stores multiple elements of the same type and can grow or shrink as elements are added or removed. For example, to create a **Vector of Strings**, add a few elements to it, print out its size, and print out its first element:

```
Vector<String> v = new Vector<String>();
v.add( "pear" );
v.add( "apple" );
v.add( "orange" );
System.out.println( v.size() );
System.out.println( v.get(0) );
```

   HINT: To reverse a **String**, you may wish to look at the **StringBuffer** class.

Reading text files

4. Write an application **Find** that searches a file specified on the command line and prints out all lines in that file containing a particular keyword. Text file **bestpicturefilms.txt** has been provided as an example. In this example, running:

```
java Find Godfather bestpicturefilms.txt
```

should print:

1972	(45th)	The Godfather
1974	(47th)	The Godfather Part II

The keyword to be found is always given as the first command line argument.

5. [ADVANCED] Modify **Find** to allow the user to include a **command line option** *-i* that will cause the matching to be case insensitive. For example, running:

```
java Find -i return bestpicturefilms.txt
```

should print:

2003	(76th)	The Lord of The Rings: The Return of the King

Including *-i* is optional; if it is included, it should precede the keyword.

Reading from text files with java.io

6. The **java.io** class **FileReader** provides low-level methods to allow reading from a text file one character at a time. One of these methods is **read()**, which returns an **int** representing a character. This **int** can be **cast** to a **char** with the expression **(char)fileReaderObject.read()**. For example, to read and print out one character from a file using **FileReader**:

```
FileReader reader = new FileReader( "textfile.txt" );
char c = (char)reader.read();
System.out.println( "First character in file: " + c );
reader.close();
```

Using only the **FileReader** class, write a command line application **CharReader** which will read all of the characters in the file **alphabet.txt**.

HINT: **read()** returns **-1** to indicate that it has reached the end of the file and cannot read any more characters.

7. The **ReaderTest.java** example from the lectures demonstrates the use of a **BufferedReader** with **FileReader** to read a text file line by line. Adapt this application so that it outputs the following to the command line:

- the smallest number of characters on a line
- the largest number of characters on a line
- the average number of characters on a line

Writing to text files with java.io

8. The class **FileWriter** is the counterpart to **FileReader** and is used for writing individual characters to a text file. The **PrintWriter** class provides additional functionality for conveniently writing **String** representations of objects and primitives to a text file. **PrintWriter** can be used with **FileWriter** as follows:

```
FileWriter writer = new FileWriter( "textwriteexample.txt" );
PrintWriter out = new PrintWriter( writer );
out.println( "One" ); // writes "One"
out.println( 2 ); // writes "2"
out.println( 3.0 ); // writes "3.0"
out.close()
```

   Write a command-line application **FivesWriter** that will write the first 12 multiples of 5 to a text file named **times5.txt**.

9. A **FileWriter** object can operate in either **write mode** or **append mode**. By default a **FileWriter** operates in write mode, meaning that the existing file is erased before new characters are written to it. To enable append mode, the **FileWriter** sould be constructed as follows:

```
FileWriter writer = new FileWriter( "textwriteexample.txt", true );
```

   Modify your solution to Q8 so that the **FileWriter** operates in append mode. Execute the application twice. What do you notice about the contents of **textwriteexample.txt**?

10. Adapt your solution to Q8 to write a command-line application **MultiplesWriter** that will write the multiplication table for the first twelves multiples of a given integer to a given text file. The integer and output file should be provided as command-line arguments. For example, executing:

```
java MultiplesWriter 12 times12.txt
```

    should produce a text filed called **times12.txt** containing:

1 * 12 = 12
2 * 12 = 24
3 * 12 = 36
4 * 12 = 48
...

11. [ADVANCED] Write an application **Filter** that searches a text file specified on the command-line and writes out all lines in that file containing a particular keyword to another file. Using the text file **bestpicturefilms.txt** provided, running:

```
java Filter Godfather bestpicturefilms.txt filteredlist.txt
```

    should create a filed called **filteredlist.txt** containing:

1972	(45th)	The Godfather
1974	(47th)	The Godfather Part II

    The keyword to be found is always given as the first command-line argument. The second argument is the input file and the final argument is the output file.