La Trobe University
Department of Computer Science and Computer Engineering

Object-Oriented Programming Fundamentals

Semester 2, 2012

Assignment

Due Date: Thursday 9.30a.m. October 18th 2012: *Delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty. Students must plan their work to allow for both scheduled and unscheduled downtime.*

This is an individual assignment. You are not permitted to work as a group when writing this assignment.

Copying, Plagiarism: Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. The Department of Computer Science and Computer Engineering treats academic misconduct seriously. When it is detected, penalties are strictly imposed. Refer to the unit guide for further information and strategies you can use to avoid a charge of academic misconduct. ***All submissions will be electronically checked for plagiarism.***

Assignment Objectives

◊ to design programs that conform to given specifications
◊ to practise combining multiple classes and methods into a whole program
◊ to implement programs in Java.
◊ to practice file input and output.

Marking Procedure and Marking Scheme
You are required to demonstrate your program during your allocated lab class in Week 13. If you are unable to attend this execution test, please email Hongen Lu (h.lu@latrobe.edu.au) before your lab class for alternative arrangements.

- *Any student who does not demonstrate their program, without prior arrangement, will receive 0 for this Assignment.*

- *Your understanding (verbal explanation of your solution) may be used as a weighting factor on the final assignment mark.*

Implementation (Execution test) 100% (Do all parts of the programs execute correctly? Note your programs must compile and run to carry out this marking.)

This assignment constitutes 10% of your overall mark in CSE1OOF. Penalties are applied to late assignments (accepted up to 4 days after the due date only). See the departmental *Student Handbook* for details.

Please Note: While you are free to develop the Java code on any operating system, your solution *must* compile and run on the latcs6 system. Code that does not run on the latcs6 system will receive zero marks for implementation.

Submission Details
You must submit electronic copies of your source code files using the submit command on latcs6. Make sure you are in the same directory as the files you are submitting. Submit each file separately. For example (note OOF is in uppercase):

```
submit OOF Country.java
submit OOF MedalTally.java
submit OOF LondonGames.java
submit OOF London2012.txt
```

Put your name and student number at the top of all files you submit

- *If you wrote extra classes, use the submit command to submit them as well.*
- *Do not submit any .class files or text files used for input.*

After submitting the files, the following command will list your submitted files:

```
verify
```

You can submit the same filename as many times as you like before the assignment deadline; the previously submitted copy will be replaced by the latest one.

Do not use LMS to submit your assignment files only use the `submit` command on latcs6

Return of Assignments
Department policy requires that assignments be returned within 3 weeks of the submission date. Check your email for when assignments may be collected from my office.

Problem Background

London 2012 Summer Olympic Games is a huge success. More than 10,000 athletes from 204 nations worldwide competed in 26 sports. Around 4,700 medals are awarded in London Olympic and Paralympic Games. To record and manage the daily medal tally is a challenging task.

Program Requirements

You have been asked to write an interactive program, in Java, to aid in the recording and managing the London Olympic Games medal tally.

Task 1

`Country.java` is a class to represent a country in medal tally. It has an attribute of country name, and attributes to record the numbers of gold, silver, bronze, and total medals. In this class, you should also define constructors, and assessor, mutator methods.

Task 2
`MedalTally.java` is a class to model a medal tally, containing all the countries winning any medal in an Olympic Games. It uses an array, `Country[]`, to store medal records of countries. Define constructors and relevant methods to manage the medal tally.

Task 3
The driver program, `LondonGames.java` starts by calling the `load( )` method. This method prompts (asks) the user for the name of an input text file. The `load( )` method must check that the requested file exists and is not empty. This is to be done using the Java `File` class, not `try/catch` blocks.

If the text file exists and is not empty, then the contents of the text file are read into the `Country[]` array.

The format of each record of this file is country name, numbers of gold, silver, bronze and total medals separated by white space. For example, the first line of the provided input file, `London2012.txt`, which stores the top 20 countries on the medal tally at August 11, 2012:

```
United States of America 41 26 27 94
```

means USA had 41 gold medals, 26 silver, 27 bronze, and total 94 medals.

After the file has been read into the array, the file is closed. All further uses of this program are with the array, not the file, except, of course, for saving the array back to a file. After the file has been read, the user is presented with the main menu, as follows:

```
            Welcome to the London Olympic Games Medal Tally
                               Main Menu
        1. Add Country
        2. Delete Country
        3. Add Medal(s)
        4. Display Medal Tally
        5. Save to file
        6. Exit

        Enter choice >>
```

Implement the functionality of each method as follows (assume user input is correct):

```
1. Add Country
        Add Menu
        1. Country Name
        2. Exit
        Enter choice >>
```

The **`Add`** menu presents the user with a sub-menu as shown above

Picking choice 1 will attempt to add a new **`Country`** to the medal tally. This means that a new **`Country`** object will created and inserted to the `Country[]` array that represents the medal tally. The user will be asked to input the name of a new country. At this stage the new country has no medals; you can add medal(s) from the main menu after the country is added.

Selecting choice 2 returns the program to the main menu

`2. Delete Country`

Selecting this main menu choice prompts (asks) the user for the country name at which to attempt to remove from the medal tally. Only one `Country` can be removed per call to this menu choice. After deletion, the program returns to the main menu.

`3. Add Medal(s)`

This menu choice prompts (asks) the country name, and numbers of gold, silver and bronze medals to add to that country, which is already on the tally.

`4. Display Medal Tally`

This choice displays all the countries and numbers of medals. After display, the programs returns to the main menu.

`5. Save to file`

When this option is selected, the user is asked for the name of a text file. If the file can be opened, then the contents of the entire array is written into the file.

The format of the output must be the same as the format of the input file. The output file must be able to be used as input file the next time the program is run.

The program does **NOT** exit when this menu choice is chosen. After writing to the output file, the program returns to the main menu

`6. Exit`

Selecting this option exits the program. The user is NOT asked if they want to save changes.

The PrintWriter class

We have covered reading from files using the `Scanner` class in lectures and laboratories. Writing to files is just as easy.

`System.out` is an object of the `PrintStream` class which is connected to the standard output (screen). The `PrintWriter` class is similar.

We can create other instances of the `PrintWriter` class and connect them to files. Using these other instances, programs can send their output to files rather than the screen.

To create a `PrintWriter` object connected to an output file we first create a `File` object (as we did with input files):

```
File myFile = new File("output.txt");
```

Then we create a `PrintWriter` object using this `File` object:

```
PrintWriter fout = new PrintWriter(myFile);
```

Alternatively, we can do this in one statement:

```
PrintWriter fout
          = new PrintWriter(new File("output.txt"));
```

We can use the same `print`, `println` and `printf` methods we used with `System.out` with the `fout` object:

```
fout.println("Hello");
```

After you have finished with your output file, you must use the close method to close the file. If you do not close the file sometimes the file is incomplete.

```
    fout.close();
```

Transferring files from Windows to Unix

Depending on how you copy your files across from Windows to Unix
you may get an extra newline character in your input files on Unix. This will
add one to the length of your strings, BUT WILL NOT SHOW UP WHEN YOU LOOK AT THE INPUT FILE.

The problem is that when you try and compare a **`String`** from user input with a **`String`** from the input file, they will NOT be equal, even though they look the same when you print them out.

How to detect the problem:

1.
    print out the length of the **`String`** in your program,
    so let's say that **`fin`** is the **`Scanner`** variable connected to your input file.

    ```
    String str = fin.nextLine( ); // reads "Hello" from the file
    System.out.println( "length of str is "  + str.length( ) );
    ```

    if the answer is 6 instead of 5, then you have the problem

2.
    open your input file, on latcs6, by typing

    ```
    vi -b input01.txt
    ```

    if you see **`^M`** at the end of each line (usually in blue)
    then you have the problem.

How to get rid of the problem:

open up the input file by typing the command

```
vi -b input01.txt
```

now type these commands
(press Esc to make sure you are in command mode)

```
:%s/ctrl-v ctrl-m//g
```

then press enter

`ctrl-v` means press the control key and the v key together

now you should no longer see any **`^M`** anywhere in the file.