The University of Melbourne
Department of Computing and Information Systems
COMP90041 Programming and Software Development
Semester 1, 2015

Project A
Due: 4pm, Thursday 02 April, 2015

1 Background

This project is the first in a series of three, with the ultimate objective of designing and implementing (in Java) a simple variant of the game of Nim. It is a two player game, and the rules of the version used here are as follows:

- The game begins with a number of objects (e.g., stones placed on a table).
- Each player takes turns removing stones from the table.
- On each turn, a player must remove at least one stone. In addition, there is an upper bound on the number of stones that can be removed in a single turn. For example, if this upper bound is 3, a player has the choice of removing 1, 2 or 3 stones on each turn.
- The game ends when there are no more stones remaining. The player who removes the last stone, loses. The other player is, of course, the winner.
- Both the initial number of stones, and the upper bound on the number that can be removed, can be varied from game to game, and must be chosen before a game commences.

Here is an example play-through of the game, using 12 initial stones, and an upper bound of 3 stones removed per turn.

- There are 12 stones on the table.
- Player 1 removes 3 stones. 9 stones remain.
- Player 2 removes 1 stone. 8 stones remain.
- Player 1 removes 1 stone. 7 stones remain.
- Player 2 removes 2 stones. 5 stones remain.
- Player 1 removes 3 stones. 2 stones remain.
- Player 2 removes 1 stone. 1 stone remains.
- Player 1 removes 1 stone. 0 stones remain.
- Player 2 wins. Player 1 loses.

2 Your Task

For this first project, the focus will be on creating two players and playing for one game:

- Your program will begin by displaying a welcome message.
- The program will then prompt for a string (no space in the string) to be entered via standard input (the keyboard) - this will be the name of Player 1. You may assume that all inputs to the program will be valid.
- The program will then prompt for another string (no space in the string) to be entered - this will be the name of Player 2.
- The program will then prompt for an integer to be entered - this will be the upper bound on the number of stones that can be removed in a single turn.
- The program will then prompt for another integer to be entered - this will be the initial number of stones.
- The program will then print the number of stones, and will also display the stones, which will be represented by asterisks '*'.
- The program will then prompt for another integer to be entered - this time, a number of stones to be removed. Again, you may assume this input will be valid and will not exceed the number of stones remaining or the upper bound on the number of stones that can be removed.
- The program should then show an updated display of stones.
- The previous two steps should then be repeated until there are no stones remaining. When this occurs, the program should display '`Game Over`', the name of the winner, and then terminate.

Here is an example execution:

```
Welcome to Nim

Please enter Player 1's name:
Luke

Please enter Player 2's name:
Han

Please enter upper bound of stone removal:
3

Please enter initial number of stones:
12

12 stones left: * * * * * * * * * * * *
Luke's turn - remove how many?
3

9 stones left: * * * * * * * * *
Han's turn - remove how many?
1

8 stones left: * * * * * * * *
Luke's turn - remove how many?
1
```

```
7 stones left: * * * * * * *
Han’s turn - remove how many?
2

5 stones left: * * * * *
Luke’s turn - remove how many?
3

2 stones left: * *
Han’s turn - remove how many?
1

1 stones left: *
Luke’s turn - remove how many?
1

Game Over
Han wins!
```

Please note that:

- You need to create a class called `Nimsys` with a `main()` method to manage the above game playing process. Particularly, when a player’s name is entered, a new object of the `NimPlayer` class should be created.
- You need to create the class `NimPlayer`. Player 1 and Player 2 are two instances of this class. This class should have a `String` typed instance variable representing the player name. This class should also have a `removeStone()` method that returns the number of stones the player wants to remove in his/her turn.
- Add other variables and methods where appropriate such that the concept of information hiding and encapsulation is reflected by the code written.
- There is **NO** blank line before the first line, i.e., no `println()` before ‘`Welcome to Nim`’.
- The last line should be **a full line**, i.e., ‘`Han wins!`’ is printed out using `println()`.
- And there are **NO** blanks after the last stone in lines displaying asterisks.

You do not need to worry about changing your output for singular/plural entities, i.e., you should output ‘1 stones’, etc.

Important Notes

Computer automatic test will be conducted on your program by automatically compiling, running, and comparing your outputs for several test cases with generated expected outputs. The automatic test will deem your output wrong if your output does not match the expected output, even if the difference is just having an **extra space or missing a colon**. Therefore it is crucial that **your output follows exactly the same format shown in the examples above.**

The syntax `import` is available for you to use standard java packages. However, please **DO NOT** use the `package` syntax to customize your source files. The automatic test system cannot deal with customized packages. If you are using Netbeans as the IDE, please be aware that the project name may automatically be used as the package name. Please remove the line like

```
package ProjA;
```

at the beginning of the source files before you submit them to the system.

Please use **ONLY ONE** Scanner object throughout your program. Otherwise the automatic test will cause your program to generate exceptions and terminate. The reason is that in the automatic test, multiple lines of test inputs are sent all together to the program. As the program receives the inputs, it will pass them all to the currently active Scanner object, leaving the rest Scanner objects nothing to read and hence cause run-time exception. Therefore it is crucial that **your program has only one Scanner object.** Arguments such as "It runs correctly when I do manual test, but fails under automatic test" will not be accepted.

3 Assessment

This project is worth 5% of the total marks for the subject.

Your Java program will be assessed based on correctness of the output as well as quality of code implementation. See LMS for a detailed marking scheme.

4 Submission

Your submission should have two Java source code files. You must name them `Nimsys.java` and `NimPlayer.java`, and store them in a directory under your home directory on the CSSE network. Then, you can submit your work using the following command:

```
submit 90041 projA *.java
```

You should then verify your submission using the following command. This will store the verification information in the file '`feedback.txt`', which you can then view:

```
verify 90041 projA > feedback.txt
```

You should issue the above commands from within the same directory as where the file is stored (to get there you may need to use the `cd` 'Change Directory' command). Note that you can submit as many times as you like before the deadline.

How you edit, compile and run your Java program is up to you. You are free to use any editor or development environment. However, **you need to ensure that your program compiles and runs correctly on the CSSE student servers**, using **build 1.6.0** of Oracle's (as Sun Microsystems has been acquired by Oracle in 2010) Java Compiler and Runtime Environment, i.e. `javac` and `java` programs. You should also remove any Ctrl-M characters which may be inserted if you edit your source file under a Windows environment - this can be done using the `dos2unix` command, available on the CSSE student servers. Submit your program to the CSSE student servers a couple of days before the deadline to ensure that they work (you can still improve your program). **"I can't get my code to work on**

the student server but it worked on my Windows machine" is not an acceptable excuse for late submissions.

The deadline for the project is **4pm, Thursday 02 April, 2015**. The allowed time is more than enough for completing the project. Late submissions will NOT be accepted.

5 Individual Work

Note well that this project is part of your final assessment, so cheating is not acceptable. Any form of material exchange, whether written, electronic or any other medium is considered cheating, and so is the soliciting of help from electronic newsgroups. Providing undue assistance is considered as serious as receiving it, and in the case of similarities that indicate exchange of more than basic ideas, formal disciplinary action will be taken for all involved parties.