COMP90041: Assignment 1

Lecturers: Prof. Rui Zhang, Dr. Tilman Dingler

Assignment Due: 5pm (AEDT), April 3, 2020.

1 Nim: A Game of Strategy

This project is the first in a series of three, with the ultimate objective of designing and implementing a simple variant of the game of Nim in Java. Nim 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 remaining player wins.
- 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.

1.1 Example Gameplay

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

- *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.*

2 Assignment: Implement Nim's Basic Game Mechanics

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

1. Your program will begin by displaying a welcome message.
2. 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.
3. The program will then prompt for another string (no space in the string) to be entered - this will be the name of Player 2.
4. 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.
5. The program will then prompt for another integer to be entered - this will be the initial number of stones.
6. The program will then print the number of stones, and will also display the stones, which will be represented by asterisks '*'.
7. 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.
8. The program should then show an updated display of stones.
9. The previous two steps should then be repeated until there are no stones remaining. When this occurs, the program should display '`Game Over`', and the name of the winner.
10. The program should then ask user whether the players wanted a play again. The user is prompted to enter 'Y' for yes or 'N' for no. If the user enters 'Y', that is, the user wants to play another game, repeat steps 4-10. Otherwise, the program should terminate. Note, any input apart from 'Y' should terminate the game.

2.1 Example Execution

After executing your program, the flow should be something like this:

```
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!

Do you want to play again (Y/N):
Y
Please enter upper bound of stone removal:
5

Please enter initial number of stones:
15

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

14 stones left: * * * * * * * * * * * * * *
Han's turn - remove how many?
2
```

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

9 stones left: * * * * * * * * *
Han's turn - remove how many?
4

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

Game Over
Han wins!

Do you want to play again (Y/N):
N
```

2.2 Some Hints

Please note that:

- You need to create a class called `Nimsys` with a `main()` method to manage the above game playing process.
- 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. You will lose marks if you fail to create two instances of `NimPlayer`.
- 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`'.
- There is a withespace between `stones left :` sentence and `* * *`, and also between all `* * *`, but **NO** withespace at the end of `* * *`.
- The line that prints the winners name 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.
- Keep a good coding style.

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

3 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.

4 Assessment

This project is worth 10% 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 Canvas for a detailed marking scheme.

5 Test Before Submission

You will find the sample input file and the expected ouput file in the Projects page on LMS for you to use on your own. You should use them to test your code on your own first, and then submit your code. Note that you must submit your code through the submit system in order to make a valid submission of the project, details of submit system can be found in Section 6.

To test your code by yourself:

1. Upload to the server your Java code files named "`Nimsys.java`" and "`NimPlayer.java`", and the test input data files "`test0.txt`".
2. Run command: `javac *.java` (this command will compile all your java file in the current folder)
3. Run command: `java Nimsys < test0.txt > output.txt` (this command will run the Nimsys using contents in '`test0.txt`" as input and write the output in `output.txt`)
4. Run command: `more output.txt` (this command will show the your program's output)

5. Compare your result with the provided output file `test0-output.txt`. Fix your code if they are different.

6. When you are satisfied with your project, submit and verify your project using the instructions given in the Section 6.

NOTE: The test cases used to mark your submissions will be different from the sample tests given. You should test your program extensively to **ensure it is correct for other input values** with the same format as the sample tests.

6 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 Engineering School student server. Then, you can submit your work using the following command:

```
submit COMP90041 projA *.java
For Late submission: submit COMP90041 projA.late *.java
```

Note that *.java includes all Java files in current directory, so you must make sure that you don't include irrelevant Java files in the current directory. 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 COMP90041 projA > feedback.txt
For Late submission: verify COMP90041 projA.late > 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 Engineering School student server**, using **build 1.8.0** of Oracle's (as Sun Microsystems has been acquired by Oracle in 2010) Java Compiler and Runtime Environment, i.e. `javac` and `java` programs.

Submit your program to the Engineering School student servers **a couple of days before the deadline** to ensure that they work (you can still improve your program and submit it again). **"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. Email submissions will not be accepted either.**

The deadline for the project is **5pm, Friday 3 April, 2020**. The allowed time is more than enough for completing the project. **There is a 20% penalty per day for late submissions. Suppose your project gets a mark of 5 but is submitted within 1 day after the deadline, then you get 20% penalty and the mark will be 4 after penalty. There will be no mark for submissions after 4pm 8 April.**

7 Individual Work

This is not a group project! We expect each student to submit a unique solution. Note 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 copying from any online sources in case anyone shares it. 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. A sophisticated program that undertakes deep structural analysis of Java code identifying regions of similarity will be run over all submissions in "compare every pair" mode. In case of verified plagiarism (copying source code from an external source or from each other) will result in scoring 0 on the assignment for each involved party. In severe cases, the involved parties may fail the class. **Plagiarism is not worth it, don't do it and write your own code!**