COMP 1230 –Summer 2017
Laboratory No. 3
Due: Saturday, June 3 (by Midnight)

Submission via blearn.tru.ca

This lab will help reinforce the concepts of encapsulation and inheritance that were discussed in the lectures.

Exercise 1 [35]:

Introduction:

This exercise asks you to design your own object-oriented program and build it *"from scratch"*. Your task is to write a program that plays the game of Nim. For now, we will use a text-based interface, though soon in the course we will be able to implement a nice graphical front end to the program.

Tasks:

1. *Decompose the project requirements into smaller stories.*

   Study the rules of the game and the project requirements defined in the specification below. This specification is very simple, listing only a half dozen required behaviors. But some of those behaviors are much too big to implement directly, for example, *"Play a game between the program and the user."*

   From the specification, draft a few scenarios in which two players play the game. Decompose each scenario down into a set of small behaviors that you might be able to implement one at a time.

   Create a list of these small behaviors, in the order you think you might want to implement them.

2. *Design and implement a program to play the game of Nim.*

   Use your list of behaviors to design and write your program. Then write Java classes to implement your design. You also need to write Java Doc comments in your source codes and generate documentation for your program. You may create whatever classes you think best model the problem.

Note: You must identify key objects from the problem description and write classes for them. If you design properly, you should have more than two classes. You may also think of using inheritance.

Make sure to include boundary checks on the user inputs (i.e., handle all invalid user input and provide appropriate message).

3. *Prepare a "read me" file for your program.*

   Create a file named `readme.doc` that describes the design of your Nim program. It should briefly describe each class, the relationships among the classes (use UML diagrams), and how to use your program. Finally, briefly explain (in the readme file) how your design can easily adapt with the following changes:

   i) If we want to introduce a new player, who will select a move following a different strategy.
   ii) If instead of one pile of stones, the game introduces three (or n) piles of stones with different sizes.
   iii) If the game needs to be played between two computers (i.e., both the players are computers).

We will evaluate your program on the quality of the ***design*** and ***implementation***. For example,

i) How well are the responsibilities apportioned to independent objects that collaborate to solve the problem?
ii) How flexible is the design in the face of reasonable changes to the specification?

Your code and your "read me" file are the only ways you have to communicate your design and implementation, so try to use them to communicate well.

A Specification for a Program to Play Nim

The game of Nim consists of two players and a pile of sticks or stones. The players alternate turns, which consist of picking up *one, two, or three* stones. The player who picks up the last stone *loses* the game.

Your program should allow the user to play a number of games of Nim against the computer, displaying results of the play to the user.

Your initial implementation provide only a few simple behaviors:

- Allow the user to specify the number of stones to be used for the game.
- Allow the user to specify which player goes first.
- Play a game between the program and the user.
- When it is the program's turn to play, choose a legal number of stones to pick up.
- After each move by both players, display the state of the game to the user, including the number of stones picked up on the previous move, the number of stones remaining, and the winner of the game, if relevant.
- Allow the user to choose whether to play another game.

Later, we will add more interesting behaviors to the program, such as *different strategies for the computer player*, *computer-versus-computer* and *human-versus-human play*, *a graphical interaction*, and so on. The program you write for this assignment should **NOT** do all of these things, *but if you design your program in a way that takes advantage of interacting objects then you will find it is easier to add them later.*

Your computer player can choose its move in any way you wish, even randomly. (It turns out that, in Nim, one player always has a perfect strategy available, depending on the number of stones that can be picked up on a turn. This means that the player can always win, no matter what her opponent does. If you know or discover this strategy, then you can implement it if you want. But you don't have to!)

The following is an example of the game of Nim played between Marvin and Hal.

- The game starts with 20 sticks on the board.
- Marvin takes 3 sticks, there are 17 sticks remaining.
- Hal takes 2 sticks, there are 15 sticks remaining.
- Marvin takes 1 stick, there are 14 sticks remaining.
- Hal takes 3 sticks, there are 11 sticks remaining.
- Marvin takes 2 sticks, there are 9 sticks remaining.

- Hal takes 2 sticks, there are 7 sticks remaining.
- Marvin takes 3 sticks, there are 4 sticks remaining.
- Hal takes 1 stick, there are 3 sticks remaining.
- Marvin takes 2 sticks, there is 1 stick remaining.
- Hal has to take the final stick and loses.

Following is a snapshot of the Nim game (that you need to write) played between computer and human:

```
Welcome to the Nim Game!

Please enter the number of stones to be used in this
game: 21

Enter your name: Sam

Which player will start the game? Select a number from
the following:



1. Computer

2. Sam

Choice: 1




Computer takes 3 stones, there are 18 stones remaining.

Please enter how many (1, 2, or 3) stones you want to
take from the pile of 18 stones: 3

Sam takes 3 stones, there are 15 stones remaining.

Computer takes 2 stones, there are 13 stones remaining.

Please enter how many (1, 2, or 3) stones you want to
take from the pile of 13 stones: 21
```

```
Invalid choice. Try again...

Please enter how many (1, 2, or 3) stones you want to
take from the pile of 13 stones: -1

Invalid choice. Try again...

Please enter how many (1, 2, or 3) stones you want to
take from the pile of 13 stones: 2

Sam takes 2 stones, there are 11 stones remaining.

Computer takes 1 stones, there are 10 stones remaining.

Please enter how many (1, 2, or 3) stones you want to
take from the pile of 10 stones: 1

Sam takes 1 stones, there are 9 stones remaining.

Computer takes 2 stones, there are 7 stones remaining.

Please enter how many (1, 2, or 3) stones you want to
take from the pile of 7 stones: 8

Invalid choice. Try again...

Please enter how many (1, 2, or 3) stones you want to
take from the pile of 7 stones: 3

Sam takes 3 stones, there are 4 stones remaining.

Computer takes 3 stones, there are 1 stones remaining.

Please enter how many (1, 2, or 3) stones you want to
take from the pile of 1 stones: 1

Sam takes 1 stones, there are 0 stones remaining.


Sam had to take last stone and lost in the game.

Thanks for playing.
```

```
Want to play again? (Y/N): N
```

Submission:

Put all the *source codes*, *java documentation*, and the *readme file* in a zip folder and submit it via blearn.tru.ca. Remember, the readme file should also contain instruction on how to run your program.

Marking Scheme for exercise 1:

1. Design – well described and has UML [10 Points]
2. Correct implementation [10 Points]
3. "readme.doc" – contains all the required information plus answer to the questions. [10 Points]
4. Java Doc – good documentation [5 Points]

Exercise 2 [15 points]:

You should carefully study the Tic-Tac-Toe game (We have developed the code in the class (May 24th). I have uploaded all necessary source codes for the game in belarn). It is a full-functioning program. You can run the game "as it is" by executing the code from the "TicTacToe.java" file. I have provided the source codes for this lab. The reason for providing the source codes are as follows:

- In real-life you will face some situations, when you will be given some existing codes to work with and then you will need to update/modify them to meet the future requirements.
- If you see how other people have applied OOP concept on a large complex problem, then that experience will help you to apply the OOP concepts in your project (coming … next). In other words, these source codes should give you an idea that how Tic-Tac-Toe game app is modularized into small independent classes (modules) and how these classes interact with each other to build a complex game app like Tic-Tac-Toe.

Once you study all the source codes and understand their functions and relations, then, modify the game so that a player can play against the computer. In other words, modify the provided tic-tac-toe game, so that the computer plays the role of the player CROSS ('X'), and automatically plays against the other player ('O').

BONUS (5 points): If you could make the "brain" of the computer smart enough that it never loses in the game.

Submission for Exercise 2:

Submit all the source code files. Highlight all the parts that you have modified from the original code. Comment the source code properly. *Marks will be deducted if the source code is not properly commented. Test and take some snap-shots of your test and submit that as well.*