Concordia University
Computer Science & Software Engineering
Faculty of Engineering and Computer Science

Concordia University
Comp 249 – Winter 2021
Object-Oriented Programming II
Assignment # 1 – Due 11:59 PM –
Friday, February 5, 2021

Purpose: Practically, this assignment should be considered as *Assignment # **0***! The purpose of this assignment is to help you review some of the main topics covered in previous courses, such as classes, loops, arrays, static attributes and static methods.

General Guidelines When Writing Programs:

- Include the following comments at the top of your source codes
  ```
  // -------------------------------------------------------
  // Assignment (include number)
  // Question: (include question/part number, if applicable)
  // Written by: (include your name and student id)
  // -------------------------------------------------------
  ```
- In a comment, give a general explanation of what your program does. As the programming questions get more complex, the explanations will get lengthier.
- Include comments in your program describing the main steps in your program.
- Display a welcome message which includes your name(s).
- Display clear prompts for users when you are expecting the user to enter data from the keyboard.
- All output should be displayed with clear messages and in an easy to read format.
- End your program with a closing message so that the user knows that the program has terminated.

JavaDoc Documentation:

Documentation for your program must be written in **JavaDoc**.
In addition, the following information must appear at the top of each file:

```
Name(s) and ID(s)   (include full names and IDs)
COMP249
Assignment #        (include the assignment number)
Due Date            (include the due date for this assignment)
```

Part I)

In this part, you need to create a class called LadderAndSnake. A LadderAndSnake object has a board of 10x10 and a number of players attached to it, which is initialized at the creation time of the object. The number of players must be between 2 and 4 inclusively. The number of players must be set through the use of a parameterized constructor of the class. In addition, the board itself is set as shown in Figure 1 below.

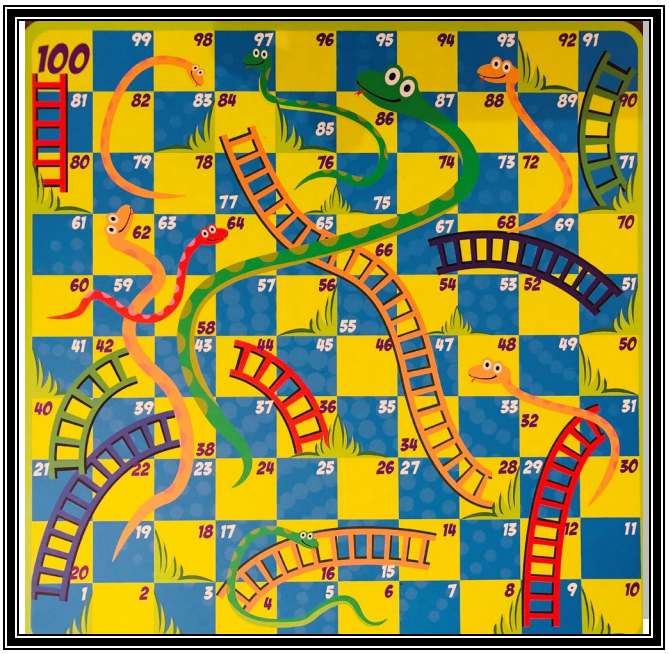

Figure 1: Ladder and Snake Board

Besides the constructors, and all basic methods in the class, the class must include two methods, one called flipDice(), which should return a random value between 1 and 6 inclusively. The other method is called play(), which actually initiate the core engine of the game where the players start to play the game. The rules of the game are as below:

- Before any of the players starts playing, the order of playing turns must be determined. For that, each player must throw the dice to obtain the largest possible number. In case of a tie between any of the players, the process is repeated only between those players. This process is concluded once the order of playing is determined.
- At this point, the players start playing the game by alternating dice flipping.
- Each dice flip will move a player form square 1 with the value of the dice. For example, if a player is at square 1 and the dice value was 4, then the player moves to square 5.
- If the reached square has a bottom of a ladder, then the player moves up to the square that has the top of the ladder. For instance, if a player is at square 33, and the flipped dice value was 3, then the player moves to square 36, which in turn will end moving the player up to square 44.
- If the reached square has a head of a snake, then the player moves down the square that has the tail of the snake. For instance, if a player is at square 77, and the flipped dice value 2, then the player moves to square 79 (which has the tip of the snake's head), which in turn end moving the player down to square 19.

  ⇨ You will have to find a way to record the relation between these particular ladder and snake squares.
- The game is concluded once any of the players EXACTLY, reaches square 100.
- If a player is close to 100, and the dice value exceeds the maximum possible moves, the player moves backward with the excessive amount. For instance, if a player is at square 96 and the dice value is 5, then the player moves to 99 (that is 4 moves to 100, then 1 move backward to 99).

As a general requirement, you must show/display ALL operations of the game. For instance, at start, you should indicate something like:

- Game is Played by *x* players (where *x* is the number of players set for that game);
- Now deciding which player will start playing;
- Player 1 got a dice value of 5
  Player 2 got a dice value of 3
  Player 3 got a dice value of 3
  A tie was achieved between Player 2 and Player 3. Attempting to break the tie
  Player 2 got a dice value of 2
  Player 3 got a dice value of 6
- Reached final decision on order of playing: Player 1, Player 3, Player 2
- Player 1 got dice value of 4; now in square 5
- Player 2 got dice value of 6; now in square 7
- Player 3 got dice value of 3; gone to square 4 then up to square 14
- Game not over; flipping again
- Player 1 got dice value of 4; gone to square 9 then up to square 31
- Player 2 got dice value of 3; now in square 10
- Player 3 got dice value of 2; gone to square 16 then down to square 6
- Game not over; flipping again
- :
- :
-

The above sample of displays is the basic minimum that you should have. A better mark is given for more elaborate/creative displays (i.e. possibly visibly showing the plays/moves, giving each of the players a name, etc. There is no limit of what you can do here. Be impressive!).

Part II)

Create a public driver class called PlayLadderAndSnake. In the main() method, you need to prompt the user to enter the number of players that he/she needs to play the game with. The user must enter a value between 2 and 4. If the user does not enter a correct value, the program must prompt the user again to enter a correct value. However, there is a limit of 4 total attempts. If the user enters an incorrect value 4 times, then the program must display a message to this effect and terminates. For instance, the program behaviour may look something similar to the following:

```
Enter the # of players for your game – Number must be between 2
and 4 inclusively:
        6
Bad Attempt 1 - Invalid # of players. Please enter a # between 2 and
4 inclusively:
        8
Bad Attempt 2 - Invalid # of players. Please enter a # between 2 and
4 inclusively:
    i.  5
Bad Attempt 3 - Invalid # of players. Please enter a # between 2 and
4 inclusively. This is your last attempt:
        10
Bad Attempt 4! You have exhausted all your chances. Program will
terminate!
```

- Of course entering a good value before the number of attempts is exhaused will resut in the game being payed until one player wins.

⇨ A final note: You can see that the description of the assignment, while surely sufficient, is very undetailed! Be creative; this is programming!

Submitting Assignment 1

- For this assignment, you are allowed to work individually, or in a group of a maximum of 2 students (i.e. you and one other student). Groups of more than 2 students = zero mark for all members! Submit only ONE version of an assignment. If more than one version is submitted the first one will be graded and all others will be disregarded.
- Students will have to submit their assignments (one copy per group) using Moodle. Assignments must be submitted in the right DropBox/folder of the assignments. **Assignments uploaded to an incorrect DropBox/folder will not be marked and result in a zero mark. No resubmissions will be allowed.**
- Naming convention for zip file: Create one zip file, containing all source files and produced documentations for your assignment using the following naming convention:

  The zip file should be called *a#_StudentName_StudentID*, where # is the number of the assignment and *StudentName/StudentID* is your name and ID number respectively. Use your "official" name only - no abbreviations or nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. For example, for the first assignment, student 12345678 would submit a zip file named like: *a1_Mike-Simon_123456.zip.* if working in a group, the name should look like: *a1_Mike-Simon_12345678-AND-Linda-Jackson_98765432.zip.*
- If working in a team, only one of the members can upload the assignment. Do **NOT** upload the file for each of the members!

IMPORTANT (Please read very carefully): Additionally, which is very important, a demo will take place with the markers afterwards. Markers will inform you about the details of demo time and how to book a time slot for your demo. If working in a group, both members must be present during demo time. Different marks may be assigned to teammates based on this demo.

- **If you fail to demo, a zero mark is assigned regardless of your submission.**
- **If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a penalty of 50% will be applied.**
- **Failing to demo at the second appointment will result in zero marks and no more chances will be given under any conditions.**

Evaluation Criteria for Assignment 1 (10 points)

Documentations	**1 pts**
JavaDoc documentations	1 pt
Part I (Class LadderAndSnake)	**8 pts**
constructors & Basic methods	1 pt
flipDice() method	1 pt
play() method & all other needed methods for program to correctly function	6 pt
Part II (Driver)	**1 pts**
Handling invalid entries	1 pt