SYSC2004 Fall 2018

Assignment 2

**Objectives:**

- Inheritance
- Static methods (i.e. main()) and variables
- More practice on 2-D arrays

Notes:

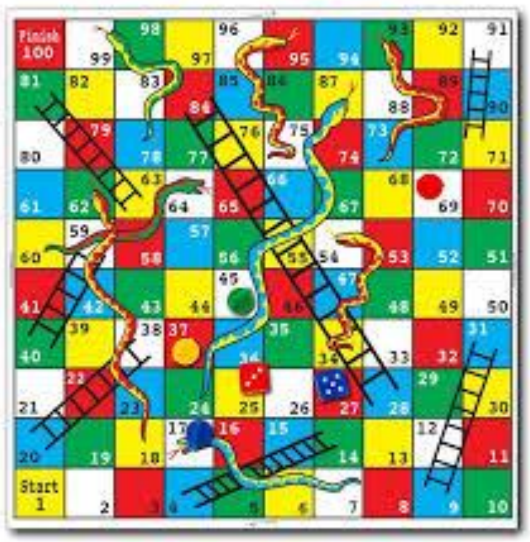
- If you did not successfully complete Assignment 1, a solution will be posted for use, after Assignment 1's deadline.
- This is a small assignment, but vital for your preparation for the second midterm.

Provided Files: SnLSquareTest.java, SorLSquareTest.java, SnakeSquareTest.java, LadderSquareTest.java, SnakesAndLadderTest.java
Re-used Files: Dice.java
Submit Files: SnakesAndLadders.java, SnLSquare.java, SorLSquare.java, LadderSquare.java, SnakeSqaure.java

## Background

We will build the Snakes-and-Ladders game. If you have never played it, do a Web search of the game and familiarize yourself with the game's rules.

The board that you will have to ultimately create is shown below.

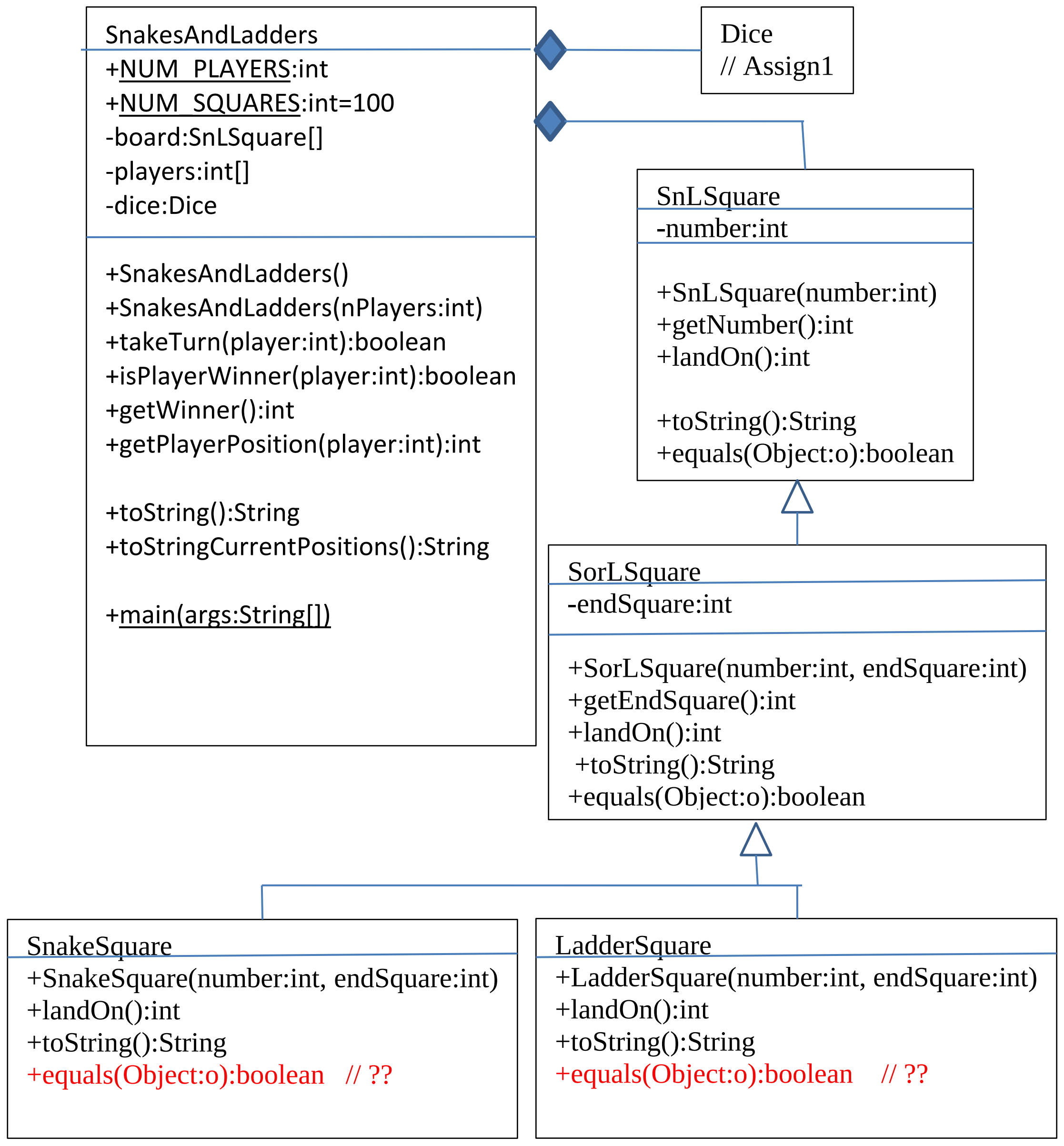


https://goo.gl/images/RoKnqW

## UML Class Diagram

The entire class diagram for the assignment is below. Except where noted below, you must follow the UML exactly – including upper and lower- case letters. You are learning programming conventions by doing so. Explanations for the diagram are given in the remainder of the document. Keep this page as a reference.

- Diamonds are for the has-a composition relationship between classes
- Triangles are for the is-a inheritance relationship between classes.



## Part 1:  The Square Hierarchy

Prepare the implementation of the entire inheritance hierarchy, but work incrementally.

1. Write the `SnLSquare` class.  Test it until all unit tests pass.
    - The format for the `toString`() method is:  <*number*>
        - <> means that it is a variable and you must substitute an actual value; don't include the angle brackets in your string .
        - Example:  1
    - The `landOn`() method should simply return the square's number.
2. Write the `SorLSquare` class. Test it until all tests pass.  Hopefully, the tests for `equals`() fail in order to teach you about the proper implementation of the `equals`() method within an inheritance chain.  If you pass on the first time, you're a star!
    - The constructor must throw an `IllegalArgumentException` if the `endSquare` is the same as the square itself.
        - Note: We have not covered this topic in class yet; however, there are plenty of examples in Assignment 1's provided code.  Learn by doing. Teach yourself.
    - The format for the `toString`() method is: <*number*>**:**<*endSquare*>
    - Example: 12:45              Means that square 12 is connected to square 45
3. Write one-by-one the `LadderSquare` and `SnakeSqaure` classes.  Test one.  Test the other.
    - The constructor of `LadderSquare` must throw an `IllegalArgumentException` if the `endSquare` is lower than itself; the `SnakeSquare` if it is higher than itself.
    - The format for the `Snake::toString`() method is: <*number*>-<*endSquare*>
        - Example: 12+34 Means that square 12 has a ladder going up to square 34
    - The format for the `Ladder`:: toString() method is: <*number*>+<*endSquare*>
        - Example: 34-12 Means that square 32 has a snake going down to square 12
    - Notice that these two subclasses do not have any local instance variables.  For that reason, I have left a question for you in the UML: **Do you have to override the `equals()` method in these classes?**  You have to make the decision: To override or not to override?  You will be marked on your choice.  You should also be prepared to explain your choice on an exam.

## Part 2: The Game Engine

Implement and test the `SnakesAndLadder` class.

- The constructors shall construct a board identical to that shown above, at the top of the document.
    - Squares are to be numbered from **1** to 100 (Max).
    - All players should start on Square 1.
    - The default constructor should build a default game with two players.
- The `takeTurn`() method should mimic the actions of a player taking their turn:
    - Roll the dice and calculate to which square the player should move.
        - To win, a player must land EXACTLY on the final square. If the roll is too large, the player must move backwards, in excess amount
            - Example: If the player is on square 98, and rolls a 5, the player moves 2 forward to square 100, but then moves 3 backwards, finally landing on square 97.
    - Move and land on the new square.
    - Return `true` if a double was rolled (so that another turn can be taken); otherwise return false.
- The `getWinner`() method should return the first-found player that is located on the final square; return -1 if no player is found.
- The format for the `toString`() method is a text representation of the board, with each square separated by the vertical slash '|' as well as padded spaces; ten squares per line.
    - (Partial) Example:

| 1 | 2 | 3 | **4+14** | 5 | 6 | 7 | 8 | **9+31** | 10 ← Notice the two ladders in this row
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 and so on.

- The format for the `toStringCurrentPosition`() method is a text representation of the current position of all players (without the board), separated by a space, all on one line.
    - Example: 1:15 2:34 (Player 1 on Square 15 and Player 2 on Square 34).

A full transcript of a sample game is provided at the end of Part 3 to help you understand the expected format of the toString() methods.

## Part 3: The Game Client

By this time you have implemented all the classes and tested them using the provided test clients. You must complete this assignment by writing a game client that automatically plays an entire game. You are replacing the test clients with your own client. This client will be written as part of the `SnakesAndLadders` class by adding a `main`() method to it.

1. In `SnakesAndLadders` class, add the call signature for the main() method

   ```
   public static void main(String args[]) { }
   ```

2. In the `main`() method:
   a. Instantiate an object of the `SnakesAndLadders` class.
   b. Print out the board
   c. Add code to repeatedly have each player take their turn until one of them reaches the final square. After each turn, print out the locations of all players. Don't forget the players take another turn if they rolled a double.
   d. Print out a final message saying who won.
3. Run the program by right-clicking on the `SnakesAndLadders` class (in the left navigation pane) and selecting "Run".
4. Play your game with various numbers of players, simply by changing the value of the NUM_PLAYERS constant.

**Sample Transcript to show you expected outputs.** **Please remove extra debugging statements.**

- Notice how the printed board below matches the image on the first page.

```
Snakes and Ladders
 | 1 | 2 | 3 | 4+14 | 5 | 6 | 7 | 8 | 9+31 | 10
 | 11 | 12 | 13 | 14 | 15 | 16 | 17-7 | 18 | 19 | 20+38
 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28+84 | 29 | 30
 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40+59
 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50
 | 51 | 52 | 53 | 54-34 | 55 | 56 | 57 | 58 | 59 | 60
 | 61 | 62-18 | 63+81 | 64-60 | 65 | 66 | 67 | 68 | 69 | 70
 | 71+91 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80
 | 81 | 82 | 83 | 84 | 85 | 86 | 87-24 | 88 | 89 | 90
 | 91 | 92 | 93-73 | 94 | 95-75 | 96 | 97 | 98 | 99-78 | 100

Player 0 rolled 12
0:13 1:1
Player 0 rolled 12
0:25 1:1
Player 0 rolled 8
0:33 1:1
Player 0 rolled 9
0:42 1:1
Player 0 rolled 10
0:52 1:1
Player 0 rolled 10
Oh no!
0:18 1:1
…..
.
0:91 1:1
Player 0 rolled 2
Oh no!
0:73 1:1
Player 0 rolled 7
0:80 1:1
Player 0 rolled 11
0:91 1:1
Player 0 rolled 9
Player 0 wins.
```

## Marking Scheme (Total : 64 – 16*4)

Assignments will not be accepted if standard indentation is not used. A mark of zero will be earned.

Instruction to TAs: Make sure your write your initials on each assignment that you mark, so that the student knows who to contact for concerns about their mark.

Instructions to Students: Afterwards, if you have a question about the marking of your assignment -

1. On the course webpage, study the schedule for the TAs and see when your TA is on duty in the lab. If you do not have a class/lab conflict with that time, visit the TA during their lab period.
2. Alternatively, email the TA that marked your assignment. Pose your question if it is simple; if not, request a meeting. If you do not receive a reply from the TA within 2 working days, re-send your email and now 'cc your instructor so that s/he can follow up.

Marking Values: Each criterion is worth 4 marks and will be marked according to the rubric levels of BDAE (0 is not present at all, or consisting only of the UML call signatures)
Beginning for 1 mark: A bit but mostly not.
Developing for 2 marks: Some but missing key points and/or ignoring requirements & conventions

- Maximum mark possible if the code does not work.

Accomplished for 3 marks: Mostly of the vital learning points demonstrate mastery
Exemplary for 4 marks: Professional level code that is even better than asked.

Part 1
/4 SnLSquare passes unit tests and the code itself is correct, especially equals()
/4 SorLSquare passes unit tests and the code itself is correct, especially equals() and toString()
/4 Both LadderSquare and SnakeSquare pass unit tests and the code itself is correct (for all methods except equals(). Next criteria.
/4 * 2 To override or not override? What choice did you make? Did you explain your decision in a comment in your code?

Part 2 – All within the SnakesAndLadders
/4 Constructors both pass unit tests and the board matches the description in the assignment.

- Beginning – if ANY instance variables is initialized as part of the declaration (instead of in the constructor.
- Exemplary – ONLY if constructor chaining is used; otherwise limited to Accomplished.

/4 takeTurn() passes unit test and the code is well constructed to meet all requirements (doubles, exact roll).
/4 Both toString() methods – Meet the required format.

Part 3
/4 main() method is written as described, within SnakesAndLadders and does all required printing
/4 The use of loops is well-structured and minimally written to handle both winning and doubles.

Overall Code Inspection:
DOES the code adhere to the Java Style Guidlines?
(https://learn.zybooks.com/zybook/CARLETONSYSC2004SchrammWinter2018/chapter/2/section/8 )
. /4 - For Whitespace – Appropriate, consistent whitespace between lines of methods as well as spaces between operators and arguments (Example: x = 5; instead of x=5; ). Readability.

/4 - For Braces – consistent placement of { } for readability
/4 - Naming of any working variables using conventions
/4 – Efficient use of variables – avoidance of repetitive allocation of variables through judicious use of variable scope.
/4 * 2 – Well-constructed JavaDoc for all methods in all classes.