CS 245 Homework 1
Duopoly

1 Overview

Summer visits to see Grandma always meant games. Dominoes, Chinese checkers, and seventeen different card games. She'd make her move, then sit back in her chair and trashtalk while I planned mine. Grandma was beatable before her morning coffee. But afterward, you'd better watch out! In this homework, you will write a framework for building two-player turn-based games like the ones Grandma loved.

Two-player turn-based games all follow a similar structure. Player 1 moves, player 2 moves, player 1 moves, player 2 moves, etc., until one player wins or the game comes to a draw. The first classes you write will codify this abstract notion of all such games.

You will also write two concrete games, one of your choosing and a two-player text-based rendition of the Tiger Toys game Lights Out. In Lights Out you have an $n \times n$ board of lights:

```
 01234
0_____
1__x__
2_xxx_
3_____
4_xxx_
```

Here, the board is 5×5 and 'x' means a light is on. If player 1 presses on column 2, row 4, that light and its left, right, up, and down (which it doesn't have, being in the bottom row) neighbors toggle their state:

```
 01234
0_____
1__x__
2_xxx_
3__x__
4_____
```

If player 2 presses on column 2, row 2, that light and its left, right, up, and down neighbors toggle their state:

```
 01234
0_____
1_____
2_____
3_____
4_____
```

Player 1 doesn't get another turn, because player 2 just won the game by turning all the lights off. You are encouraged to check out one of the various online implementations to get a feel for the rules and strategy.

As you complete this homework, you will learn about the following topics: inheritance, abstract classes, and polymorphism.

2 Requirements

Specifications do not tell you how to solve a problem—just what pieces may be used. The classes and methods described below will need to be thought about and pieced together using your own good mind. You will likely need to read this section many times.

Your solution is to meet the following specification:

1. Write all code in your fork of the class Bitbucket project, in package `hw1`.
2. Write an `enum` `GameState` that represents the current status of a game. It has the following specification:
   (a) Has values `WON`, `DRAW`, and `IN_PROGRESS`.
3. Write an `abstract` class `Duopoly` that encapsulates the notion of a two-player turned-based game. It has the following specification:
   (a) Has a constructor taking two `DuopolyPlayer` parameters. The first parameter is player 1 and the second parameter is player 2.
   (b) Has a method `play` that administers a game between the two players. The game is initially considered to be in progress. Player 1 takes a move. If the game isn't over, player 2 takes a move. The process repeats until the game is won or over in a draw. The textual user interface must communicate the game board before and after each move, as demonstrated through the following pseudocode:

   ```
   show game
   player move
   show game
   ```

   If, when the game is over, a player has won, announce the winner to `System.out`. Otherwise, declare a draw.
4. Write an `abstract` class `DuopolyPlayer` that encapsulates the notion of a player in a two-player turned-based game. It has the following specification:
   (a) Has a constructor taking a `String` parameter for the name of the player.
   (b) Has a getter for the player's name.
   (c) Has an `abstract` method `move` that accepts a `Duopoly` game parameter for the game in which the player is to take a turn. This method returns the state of the game after the player's move as a `GameState` reference. This method is `abstract` because `DuopolyPlayer`s aren't associated with any particular game and the concept of making a move in an undefined game is illogical. Instead, subclasses will override this method according to the rules of a real game.
5. Write a class `LightsOut` that encapsulates the board of an $n \times n$ Lights Out game. No more than one instance variable is needed to accomplish this task. It has the following specification:
   (a) Is a subclass of `Duopoly`.

(b) Has a constructor that accepts two `LightsOutPlayer` parameters and an `int` for the size of the game board. This constructor creates a $n \times n$ board on which all lights are initially off. You may assume $n < 10$.

(c) Has a method `randomize` that randomly enables the board's lights. It accepts a `long` parameter, which you are to use as the seed for a `Random` generator. Visit each light on the board, starting with row 0, and generate a random boolean determining the light's status. The seed is necessary to make "repeatable" and therefore "gradeable" randomness.

(d) Has a method `press` that accepts two `int` parameters, one for the column and one for the row of a player's move. It toggles the state of the light at the given location and its four neighbors. Not all neighbors may be exist. Only toggle those that do. If the row and column are not on the board, throw an `IllegalArgumentException`.

(e) Has a method `isDark` that returns `true` if all lights are off and `false` otherwise.

(f) Has a method `toString` that returns a visual display of the board of the following form:

```
 012345
0_____
1x__x_
2____x
3___x_
4___x_
```

The numbers specify the row and column addresses of each light. An underscore means the light is off. An 'x' means the light is on. Each row is followed by a newline. (You can use string concatenation to assemble the board. A more industrial grade solution might use `StringWriter` and `PrintWriter`.)

6. Write a class `LightsOutPlayer`. No instance variables are needed to accomplish this task. It has the following specification:

(a) Is a subclass of `DuopolyPlayer`.

(b) Has a constructor that accepts as a `String` parameter the name of the player.

(c) Overrides `move` such that the player presses one light of the game. Prompt the user and read from `System.in` the column and row that is to be pressed. (The grader hijacks `System.in` to accomplish its task of checking your work. A consequence of this hijacking is that you will need to construct a `Scanner` or whatever class you use to read `System.in` each time this method is called.) If, after the press, all the lights of the game are off, return `WON`. Otherwise, return `IN_PROGRESS`. This method must take a parameter of type `Duopoly` game in order to override its superclass. You may assume the runtime type is LightsOut.

7. Write a class `Main` with the following specification:

(a) Has a `main` method. What it does is not specified, but you are suggested to use it to test your code. Relying exclusively on the SpecChecker to test things will rob your brain of some neurons that are in your best interest to grow.

8. Write a second two-player turned-based game of your choosing, which may be as simple or as complicated as you like. Share your game and player classes on Piazza, tagged `duopoly`.

3 Submission

This homework is a regular assignment and is graded by hand and with help from the SpecChecker. To submit your work for grading:

1. Put the SpecChecker for this homework in your Build Path.
2. Run the SpecChecker as a Java Application (not a JUnit Test) and fix problems until all tests pass.
3. Commit and push your work to your repository. If you are resubmitting an earlier assignment, email me. The time of your email will determine the submission week.

The SpecChecker cannot check everything. Your assignment is also expected to fully meet the requirements above and the following:

- Variable names should be meaningful and accurate.
- Non-obvious parts of your code should be commented.
- Code should be cleanly formatted and indented.
- You must not plagiarize. Write your own code. Talk about code with your classmates. Ask questions of your instructor or TA. Do not look at others' code. Do not ask questions specific to your homework anywhere online but Piazza. (If you find violators of this rule, please let me know.)
- Work must be submitted according to course policies on deadlines. To be eligible for later-week submission, you must have at minimum the skeletons for all specified classes and methods in your repository by the homework deadline.