COMP 2150 – Spring 2015
Project 1: Othello
(75 points)

Number of People: Up to 2. If you work with someone else, include both your name and your teammate's name on all source code files. Only one person needs to submit the final product to eCourseware, but include a note with the submission that lists both of your names. If an honors student and a regular student decide to work together, both agree to be graded on the honors scale.

Feel free to ask me for help, or visit the Computer Science Learning Center (www.cs.memphis.edu/cslc).

Due: Monday, Mar. 16 by 12:40 pm

Submission: Zip all of your Java source files (you can zip the entire project folder if using an IDE) into a single file and upload it to the proper folder in the eCourseware dropbox at https://elearn.memphis.edu.

Coding Style: Use consistent indentation. Use standard Java naming conventions for **variableAndMethodNames**, **ClassNames**, **CONSTANT_NAMES**. Include a reasonable amount of comments.

Honors Grading: This assignment is worth up to 80/75 points for regular students, but only 75/75 for honors students.

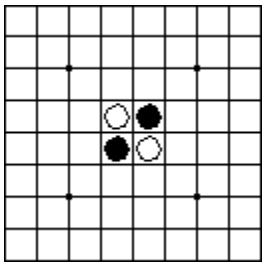
Grader: TA, Hari Cheekati (hchekati@memphis.edu). Questions about grading? Please contact him first!

Background Information

Othello (also known as Reversi) is a two-player game played on an 8x8 board similar to that used in chess or checkers. Each player chooses a color (black or white), and the objective of the game is to accumulate as many pieces of your own color as possible.

How to play

The game starts in the following configuration:



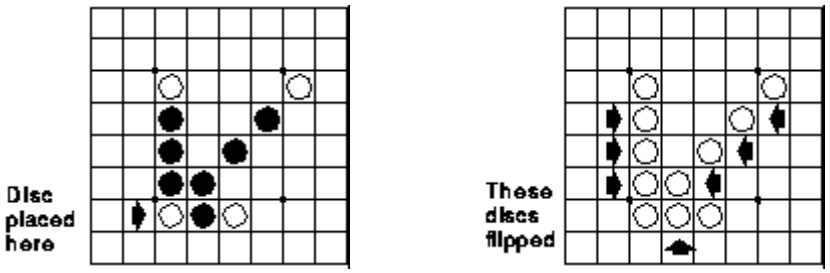
Players alternate turns, with black moving first. A move consists of placing a piece of your color on the board in such a way that it *outflanks* one or more of your opponent's pieces. Outflanking means creating a straight line of your opponent's pieces that are bordered on both ends by your pieces. This may be done horizontally, vertically, or diagonally. Outflanking your opponent's pieces flips them to your color.

If a player cannot outflank any of his/her opponent's pieces during a turn, s/he must forfeit that turn. A player may not "pass" his/her turn if any valid moves exist.

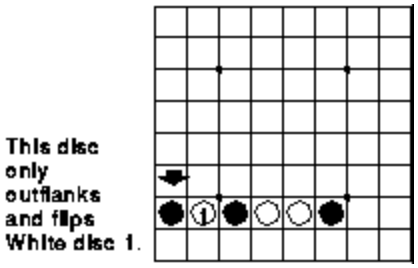
The game ends when neither player is able to make a valid move. (This is guaranteed to happen when the board is full, but it may occur before that. For example, if one player manages to flip all of his/her opponent's pieces before the board is full, the opponent is left with no valid moves.) The winner is the player with the higher number of pieces on the board.

Examples of outflanking

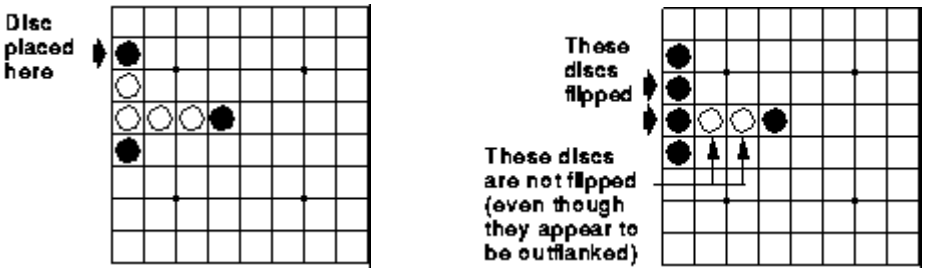
A single move may outflank more than one line of pieces. For example, the following diagrams show what happens when white places a piece that outflanks three lines of black pieces.



Outflanking applies only to the closest pair of bordering pieces. In the diagram below, black's move will flip only the white piece labeled 1, not the other two white pieces.



Outflanking cannot be "chained," as illustrated in the diagrams below. Black's move will flip only the vertical white pieces, not the horizontal ones.



(Diagrams blatantly stolen – ahem, "borrowed" – from http://www.rainfall.com/othello/rules/othellorules.asp, which no longer exists!)

The Assignment

Write a text-based implementation of Othello. Your program should start by displaying the initial configuration of the board as drawn above. Then allow the two players to alternate turns by placing pieces onto the board. If a player cannot make a valid move, the game should automatically forfeit the turn to the other player. The game should continue until neither player can make a valid move. Once this occurs, determine and display the winner of the game.

To make things easier for you (and for the TA!), I've already developed a basic class design for you. Your project must include the classes listed on the next page.

Class Design

- A **Tile** class to represent a single tile on the Othello board. This class should include (but is not limited to) the following parts:
  - Instance variables
    - State of the tile – it can be empty, black, or white
  - Methods
    - Flipping the tile to the opposite color
    - A **toString** method that returns how the tile should be displayed
- A **GameBoard** class to represent the entire set of 64 tiles. GameBoard is used to store the current state of the game. This class should include (but is not limited to) the following parts:
  - Instance variables
    - A 2-D array of **Tile** objects
    - The number of rows and columns on the board (both of these will be 8 for this game, but you should write your code in such a way that it can easily be changed to other dimensions)
  - Methods
    - A method that counts the pieces of a specific color present on the board
    - A method that places a piece of a specific color at a specific location, flipping any appropriate pieces of the opposite color
    - A method that determines whether a valid move exists on the board for a specific color
    - A method that determines the current state of the game (still in progress, victory by black, victory by white, or a tie)
    - A **toString** method that returns how the board should be displayed
- An **Othello** class that serves as the main client program for the game. This class should include (but is not limited to) the following parts:
  - Instance variables
    - A **GameBoard** object to store the state of the game
    - A **Scanner** object to read user input
  - Methods
    - A **main** method that is executed to start the game

Note that all user input should be located only in the **Othello** class. The idea is that the core game objects (**Tile** and **GameBoard**) should be independent of the game's user interface. If you wanted to make a graphical version of Othello, you'd be able to use the existing classes and make changes only to the **Othello** class!

Implement error checking on all user inputs, to the extent that was covered in COMP 1900. For example, when a player makes a move, you should prevent things like:

- Entering coordinates that are outside of the board
- Entering coordinates that already contain a piece
- Entering coordinates that would not result in any outflanking (this is not a valid move)

Need Help?

Although you may choose to work with one partner, do not consult with other students on this assignment. If you find yourself stuck, please feel free to contact me (Top) anytime. The Computer Science Learning Center in Dunn Hall 208 is also open, where you can get help from graduate students. Hours are posted at www.cs.memphis.edu/cslc.

Start working on this project early! It's not a terribly long project, but it requires significantly more code than a lab assignment. If you run into any challenges, it often helps to spend some time mulling it over in your head. You won't have this luxury if you don't start until the night before it's due!