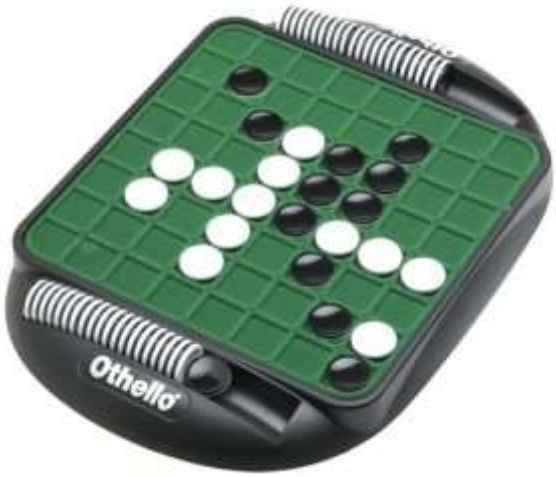
CSCI-140 Computer Science AP
CSCI-242 Computer Science Transfers
Lab 6 - Reversi



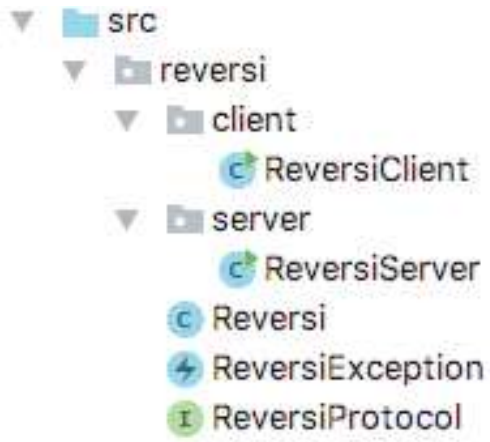
Introduction

You will be implementing a client-server program for the game of Reversi. Assume the following rules:

- The game can be played on an N by N board, where N is even and greater than or equal to 4.
- To begin with, the center of the board is populated with two white pieces and two black pieces that are diagonal to each other.
- White goes first, followed by black.
- When it is the players turn to move, they can place their piece on any empty cell directly adjacent to an occupied cell. This means the player will always be able to make a move (there is no passing).
- When the game ends and the board is full, the player with the most pieces is the winner. It is possible for there to be a tie.

Program Design

If you did not already get the starter code from the in-lab activity, you can retrieve it here. Minimally your program should be structured as follows:



Please pay careful attention to the following or else you will not be able to submit to try:

- The only classes you should create should be in the `reversi.client` and `reversi.server` packages.
- The main program for the clients must be `reversi.client.ReversiClient`.
- The main program for the server must be `reversi.server.ReversiServer`.
- You are allowed/encouraged to add more classes to either of these packages, but do not change or add to any of the classes provided above in the `reversi` package

Documentation

The documentation for the provided classes is here:

- Reversi: Provides a complete representation of the current game that updates each time a new valid move is made. The server and clients will hold instances of this class in order to handle the gameplay and display the board to each client.
- ReversiProtocol: Provides the definition of all the strings that are used for requests between the server and client, and vice versa.
- ReversiException: A custom exception class that is used by `Reversi` to communicate issues about invalid moves.

Input and Output Requirements

Server

The server requires the board dimensions and the port on the command line. If the arguments are present they are guaranteed to be valid, otherwise display a usage message and exit, e.g.:

```
Usage: java ReversiServer DIM port
```

If there are any IO exceptions, display an appropriate message and exit the program.

When the server is run it should display a message that it is waiting for the first player to connect, e.g.:

```
Waiting for player one...
```

When the first client connects, they are the white player, `O`. Next, the server should display that the first player has connected is waiting for the second, e.g.

```
Player 1 connected!
Waiting for player two...
```

Once both players have connected, it should display that the game has started, e.g.:

```
Starting game!
```

No more messages are needed from the server. But you may find it helpful to add more in when you are debugging.

Client

The client is run on the command line with the hostname and port of the server to connect. If both arguments are present you can assume their types are correct. If not, display a usage error and exit, e.g.:

```
Usage: java ReversiClient hostname port
```

If there are any IO exceptions, display an appropriate message and exit the program.

When a client successfully connects, the default board should be displayed, e.g. a 4x4 board:

```
  0  1  2  3
0[.][.][.][.]
1[.][O][X][.]
2[.][X][O][.]
3[.][.][.][.]
```

When it is the clients turn to move, it should display the last move (only after first move), the current board, and then request the move from standard input, e.g. second player's first move request after player 1 moved to (2, 0)

```
A move has made in row 2 column 0
  0  1  2  3
0[.][.][.][.]
1[.][O][X][.]
2[O][O][O][.]
3[.][.][.][.]

Your turn ! Enter row column:
```

The row and column should be entered as integers with a space between. If the input or move is invalid, the client and server should terminate gracefully.

If there are no errors and the game goes to completion, there are one of three messages that the client will display based on the outcome:

- Player won: `"You won! Yay!"`
- Player lost: `"You lost! Boo!"`
- Player tied: `"You tied! Meh!"`

Implementation

Your program must use TCP/IP sockets for communication, e.g. java.net.ServerSocket and java.net.Socket. It should use the protocol defined above so that a client written by someone else, or even in another language that is using sockets, would work with your server.

Keep in mind the following things:

- The server is not multi-threaded. It can only handle a single game and then should gracefully terminate.
- Your program should never crash with an exception. An exception should result in an error message and the graceful termination of server and both clients.
- It is guaranteed the client moves are valid moves. If the client specifies an invalid coordinate you should generate an error messae and terminate the server and both clients.

Sample Run

Here is a sample run for the server and both clients of a full 4x4 game.

Design Hints

20% of your grade is based on how well you design your system. Keep in mind the following things:

- Have a very small main method that reads the command line, constructs and object, and then passes control to the instance via a method.
- Use constructors whenever possible to allocate resources.
- Have "main loops" in a method other than the main, e.g. run(). Still, the run() should be minimized with work passed on to other methods.
- Use separate methods for the details of sending/receiving each protocol request.
- We strongly suggest using try-with-resources because they will automatically close all resources under all conditions. If you don't use them, make sure you use the finally clause of try-catch to manually close them.

Submission

Try is setup to handle code that you submit with packages. However, all files need to be sent "flat" without sub-directories. You are allowed to create source files in both the reversi/client.

```
try csapx-grd lab6-1 log.txt reversi/client/*.java reversi/server/*.java
```

Grading

The grade breakdown for this lab is as follows:

- Problem Solving: 15%
- In-Lab Activity: 10%
- Design: 20%
- Functionality: 45%
- Code Style & Version Control: 10%