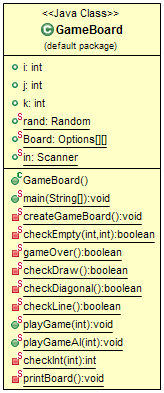
Christopher Kovaleski

Assignment 4

Well, I forgot to do a write-up on this assignment. Serves me right for not reading!

1. The expected input is an x-axis and y-axis coordinate that is unique on a 3 \* 3 grid known as the Tic Tac Toe board. When the input is given, the output is an enumeration value being changed from “Empty” to “X” or “O” depending on whose turn it is. If the selected input is not mapped to an empty enum value on the board, then an error requesting another coordinate location is given.
2. Originally I was going to have a class GameBoard and another called TicTacToe that would simply call the playGame class in GameBoard, but that just showed me how trivial this program was and took away the necessity for a second class. So first I’ll show the UML:   
   This UML pretty much shows my thought process. A constructor called GameBoard(same name as the class) creates the game board as an array of the enum type Options, based on the createGameBoard method. The main method then calls playGame for each player, which uses a series of helper methods to determine the various scenarios that could happen after a given input. The first to consider is if the input is valid using checkInt, followed by checking if the valid integer inputs resolves to an empty spot on the board using checkEmpty. Following this, there are various checks for game-ending conditions called in the gameOver method such as checkDiagonal, checkLine, and checkDraw which are named in accordance to the game-ending conditions. The method printBoard is called after each turn and as you might have concluded, it prints the game board out.  
   Now AI implementation was simple, I replaced the second call to playGame (used for human player 2) with a playGameAI method that uses random number generation and checks for empty squares to determine moves to make on the game board. It’s not smart AI, but the extra credit specs never stated the computer player had to be intelligent!
3. Testing was done by replacing both playGame functions for both players with playGameAI and putting the program into a million iteration loop with checks for each possible scenario of victory being achieved. Fun fact: my helper application took 63,456 iterations before every scenario was achieved. More often than not, computer players 1 and 2 went to a draw; meaning that they collectively went through 9 turns and hit the max runtime of my program.