**COMP 2150 - Winter 2020 - Assignment 3**

**Due March 18, 2020 at 11:59 PM**

In this assignment, you will write Java code to enable a user to play a game of “Connect”, where players alternate dropping coloured playing pieces into a grid. (The game is sold under the copyrighted name “Connect 4”.) The goal of the game is to get four of your pieces in a line (horizontal, diagonal or vertical) before your opponent does. Here’s a video that describes the game in 19 seconds: https://youtu.be/ylZBRUJi3UQ. In your game, you will use a board that can be set to a custom square size.

In your game, a human player will play against the computer. At the start of the game, the first player (human or computer) should be randomly decided by the system.The game will play once and then end. The size of the square board will be a randomly chosen integer.

You will be responsible for implementing the game logic (controlling turns, determining if someone has one the game, etc.) as well as a game AI (automated player) to play against the human.  You will be provided with the UI code.

**Game Design**

You are responsible for designing the game logic and the game AI for Connect. The game logic is responsible for maintaining the current status of the board (where each players’ pieces are) as well as determining if a player has won. On the other hand, displaying the board and prompting the human player for input is the responsibility of the UI. You are not responsible for developing the UI: it will be given to you.

You are also responsible for implementing a game AI (automated player). How the AI works is up to you, but it must pass a set of minimum standards specified below. You don’t need to program an AI that wins all the time. The reward for developing a very sophisticated game AI is the possible admiration of your classmates, and veritable minutes of entertaining game play.

The logic of your game should proceed roughly as follows:

- The game starts with an empty board. The game logic system determines who plays first (randomly).
- Turns alternate between the human player and the game AI.
- If one player wins, the match stops and the winner is announced. The match may also end in a draw if the board is completely filled.
- After the match ends, the program ends.

The tools for accomplishing all of these tasks are described below. The three major parts of the project are the game UI, the Game Logic, and the Players. The functioning of these parts is dictated by four Java interfaces. They are described below.

**Status Enum:** The game needs a way to talk about different players to answer questions like "whose turn is it?", "who won?" and "which player occupies a certain space in the board?" This will be accomplished with the Status Enum:

```
public enum Status {
    ONE, TWO, NEITHER
}
```

In the game, the human player is always player one and the computer player is always player two. The Status enum can be used to create a variable of type Status. For instance, the following line creates a variable of type Status and sets it to ONE:

```
Status s = Status.ONE;
```

Statuses can also be compared using == and be used in switch statements.

**Player Interface:** The Player interface represents the methods necessary for players of the game (both AI players and human players).

```
public interface Player {
    void lastMove(int lastCol);
    void gameOver(Status winner);
    void setInfo(int size, GameLogic gl);
}
```

The three methods work as follows:

- **lastMove** is called to tell a Player the previous move. The lastCol parameter is the column of the **last move made by the opposite player** (it is -1 if this is the first move of the game and no previous moves have been made). Note that the lastMove does not pass the entire board to the player. It only passes the last move made.

  The lastMove method is **also** an indicator that the game is ready for the player to make its next move. That is, **any time after lastMove() is called, the Player can and should send its next move to the Game Logic class** (see below for details on the Game Logic class). The player sends its move to the Game Logic class by calling the setAnswer() method (in Game Logic, see below).

  The lastMove method will **only** be called when a play is possible (i.e., there is at least one valid move the Player can make) and the Player must make a valid play on its next move. The Player does not need to ensure that the column lastCol is valid.

- **gameOver**: this method is called when the game ends. It sends a parameter that gives the result of the game. The status is the player number of the winner (if it is Status.ONE or Status.TWO) or Status.NEITHER if the game is a draw.

- **setInfo**: this method must be called for each Player before the game starts (by the Game Logic class, see below). It sends the Players basic info about the game. The method tells the Player the dimension of the board with the size parameter. It also sends a pointer to the Game Logic class (see below) so that the Player can communicate with this class.

**Human Interface:** The human interface specifies the method that is necessary for Human players. This method is called by the UI.

```
public interface Human {
     void setAnswer(int col);
}
```

The method **setAnswer** will be called by a UI when the UI has received input from the (real life) human that represents their move. The setAnswer method is the way of sending this information to the class that implements the Human interface, and the information is sent with the col parameter. The setAnswer is only called after a lastMove call is made on the UI (that is, each setAnswer call is preceded by a call to lastMove).

**UI Interface:** This is the interface that is satisfied by the user interface class. **You are not responsible for implementing this interface.** In the supplied code, you will be given a class (TextUI) that implements the UI interface. You **are** responsible for calling these methods in the Human class (see the diagram below in the Class Relationships section).

```
public interface UI {
    void lastMove(int lastCol);
    void gameOver(Status winner);
    void setInfo(Human h, int size);
}
```

The three methods are used as follows:

- The **lastMove** method is called by the Human player class when it wants to inform the UI that a move has been made by the opposing (AI) player. If lastCol=-1, it means that the game has started and there is no previous move. Otherwise, lastCol is the column that was most recently played by the opposing (AI) player. Like the lastMove method in the Player interface, this is also **an indicator that the UI can prompt the human (person) for their choice**. No information is returned by this method, however: this is done by the setAnswer method in the Human interface.

- The **gameOver** method is called by the Human class when the Human class knows that the game is over. The status is the player number of the winner (if it is Status.ONE or Status.TWO) or Status.NEITHER if the game is a draw.

- The **setInfo** method must be called (by the Human class) directly after the UI object is constructed to send a pointer to the Human that it is interacting with. This will allow the UI to call the setAnswer() method of the Human class. It also sends the size of the board.

**Game Logic:** You are responsible for implementing the back end that manages the game logic. Your game logic class must satisfy the following interface:

```
public interface GameLogic {
    void setAnswer (int col);
}
```

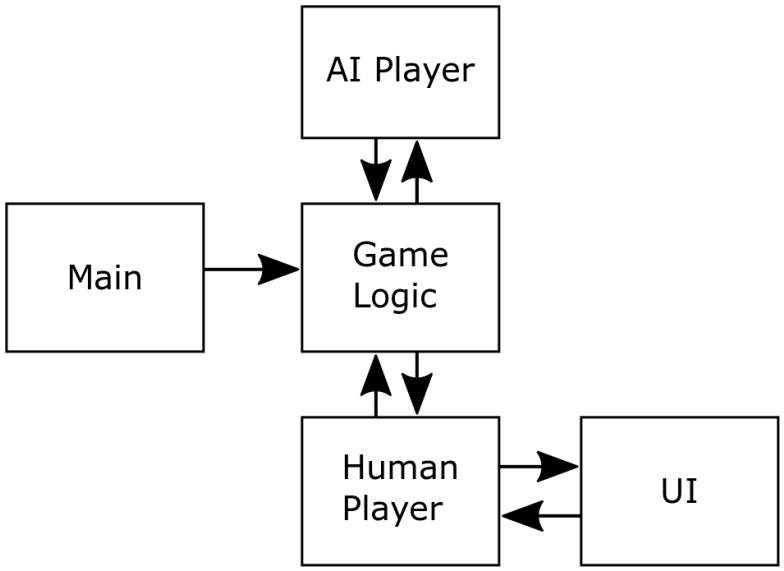
The method **setAnswer** will be called by a Player when the Player (either Human or AI) has determined their move. The setAnswer method is the way of sending this information to the class that implements the GameLogic interface via the col parameter. The setAnswer is only called after a lastMove call is made on the Player (that is, each setAnswer call is preceded by a call to lastMove).

The game logic is as follows:

1. The logic must determine the board size (this should be a board with height and width between 6 and 12 inclusive, which randomly generated). The game logic must inform the players of the board size.
2. The game logic must determine who goes first.
3. The game logic must then alternate between players, asking for their moves (using the method lastMove), and informing the other player of the move. This will use the methods from the Player interface.
4. The game logic must determine if a player wins (or if there is a draw), and then inform the players of this fact.
5. After the game is done, the program is done.

**Class Relationships**

The class relationships are described in the image on the right. This shows which classes interact with other classes (that is, it does not show a hierarchy between classes). An arrow denotes that one class calls methods in another class.




The game will be launched by a small Main class that only contains

a small amount of static code (preferably two lines of code, must be less than five). The Game Logic class will create objects for the Human and AI players, and call methods on those objects to start and continue the game.

As you can see, the UI only interacts with the human player. The UI does not interact with the game logic class or any other classes.

One question you may have is "Why are the lastMove and setAnswer methods separate for the Game Logic and the Players?" (or equivalently, "Why are the lastMove and setAnswer methods separate for the UI and Human Player?"). It may seem that you should simply have lastMove() return a value that represents the move of the current player. This is a suitable approach for a text-based, synchronous UI, but as you will see when you have a GUI, it is not appropriate in an graphical environment with event-driven programming.

The behaviour of your solution can be unpredictable if methods are called out of order (e.g,., if answers are sent before they are requested, or after a game ends).

**Small hint**: you may feel that your Game Logic class needs a loop to continue moving the game. However, you will likely not do this: instead, think of your Game Logic as being driven by method calls, with this observation: when a Player calls sendAnswer (in the GameLogic object), their turn is over, and it's the other Player's turn.

### AI Player Requirements

Due to the complexity of possible AI behaviours, the description of the minimum AI requirements are provided as text only.

1. Your AI must first be defensive: if there is a place where its opponent could play that would result in a loss (for the AI), it must play there. If there are several places that the opponent could play and win, the AI must choose one of them (and then presumably lose on the next turn).
2. Your AI must next be offensive: if case 1 does not occur, but there is a spot where the AI could play that would result in a win (for the AI), it must play there. If there are several places that the AI could play, it must choose one of them.
3. If both case 1 and case 2 do not occur, there are no regulations on your AI. It may play how it likes.

The markers will be running games by hand to verify that these requirements are met. There are no unit tests for this assignment.

### Data Structures

In this assignment you are permitted to use a 2D array**.** No linked structures are likely to be necessary in the game logic. However, any data structures constructed elsewhere in the game, other than 2D arrays to store the board, must be implemented by you.

**Hand-in**

Submit all your source code for all classes you implement. For ease of compilation, you must provide copies of all classes, interfaces and enumerated types that have been provided to you. (However, they cannot be modified. You must submit the same versions that are provided to you.)

You should also submit a readme.txt file that describes exactly how to compile and run your code from the command line. The markers will be using these instructions exactly and if your code does not compile directly, you will lose marks. Each class must be in its own file.

You have been given a preliminary (text-based) interface for programming. You will be given the GUI interface very close to (as little as one day or so before) the deadline. The markers will be running your code with this GUI. The GUI will satisfy the UI interface.

You **MUST** submit all of your files in a zip file.  Submit all files on UM Learn Dropbox.