CS211	Computers and Programming	Spring 2000
	ASSIGNMENT 2: GUI	

POSTED: February 17. **DUE DATE: March 2.**

Make sure you read the whole assignment before attempting to answer the questions.

The Game FIAR: Four In A Row

In this assignment we will implement a game called FIAR (Four In A Row).

FIAR is a game played on a rectangular board. There are two players who take turns. One player has red pieces and the other one has blue pieces. The rectangular board is divided into *slots*. For example, a 6 x 7 board would have 42 slots. Imagine the board to be standing on one of its sides. Pieces are dropped *vertically* in turn into the *columns* of slots. A piece dropped into a column either falls to the bottom of the column if the column was empty or lands on top of the pieces already in this column.

Players take turns in dropping the pieces into columns. Different criteria can be used to declare the winner. The standard version of the game states that a player that first manages to get four of his pieces straight in a row, column or diagonal wins the game. Another variation stipulates that the first player to get his fours pieces *diagonally* in a “straight row” is the winner. More details on different variations are given in the relevant sections. The whole point of the game is to drop pieces strategically and be the first one to meet the criteria for being a winner.

In this assignment, you will write a program that allows two players to play this game.

The Graphical User Interface (GUI)

A graphical user interface (GUI) will be implemented to play the game of FIAR. The GUI consists of two windows, shown in Figure 1 and Figure 2. The *layout* for the two window should include the respective components shown in Figure 1 and Figure 2. Extra panels can be used where appropriate to group components.

The Main Window

The main window is shown in Figure 1. It comes up when the program is started. It allows the user to start a new game at any time by clicking the **New Game** button. This brings up a *modal* dialog window shown in Figure 2. User interaction with the modal dialog window is explained below.

Initially the *graphical game board* is empty as shown in Figure 1a. After a new game has been started, the graphical game board shows the column numbers at the top and the pieces that have been added in subsequent moves by the players (Figure 1b). A graphical game board can be implemented by extending a *panel* which contains empty *labels* corresponding to the slots in the game board. The color of the appropriate label is changed to reflect the *move* made by the user. The main window should call the `repaint()` method to show this change on the screen. Each time a new game is started, the graphical game board should be reset and the main window should call the `validate()` method to update the layout.

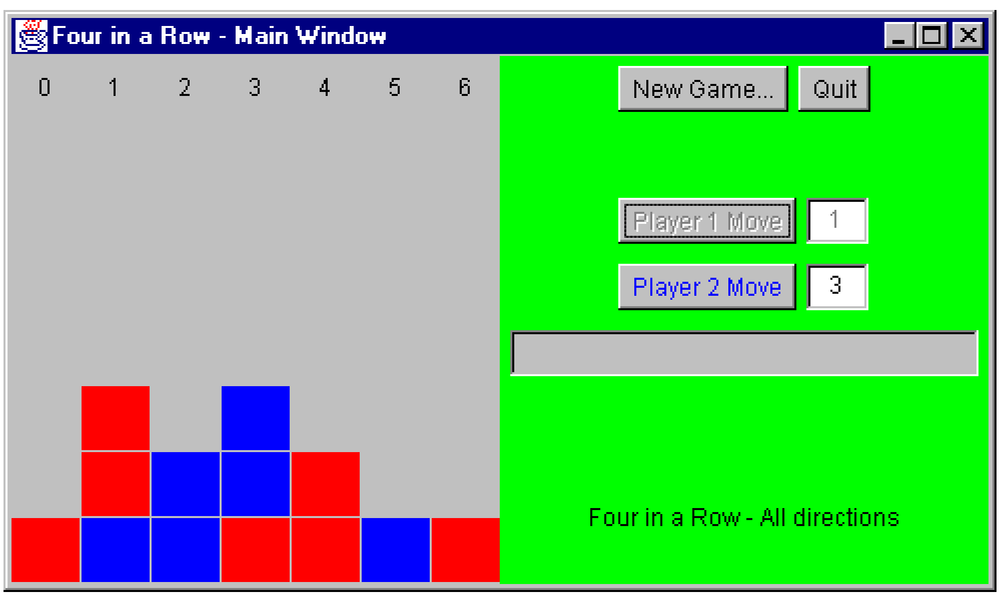
Each player makes a move by specifying the column number and clicking the appropriate player button. Only one player at a time can make a move. The main window should not allow the players to make a move unless a new game has been started (see Figure 1a).

Clicking the **Quit** button obviously terminates the program. The non-editable Message Display text field can be used to give feedback to the user. The main window also shows the variation of the game that is being played.

The New Game Dialog Window

The modal dialog window in Figure 2 comes up when the user clicks the **New Game** button in the main window. The necessary parameters for running the game are specified in this dialog window: the size of the board game, which player makes the first move, and the type of the game.

(a)



(b)

Figure 1

Three types or variations of the game are to be provided:

- Four in a Row - All directions (Variation A)
  This is the standard game of FIAR. A player that first manages to get four of his pieces straight in a row, column or diagonal wins the game. The program checks after each move to see if a winner can be declared.
- Four in a Row - Diagonals only (Variation B)
  In this variation, the first player to get his fours pieces *diagonally* in a "straight row" is the winner. The program checks after each move to see if a winner can be declared.
- No checking (Variation C)
  The two players determine the winner themselves, i.e. the program does not check for a winner.

Clicking the **OK** button in the modal dialog window should *validate* the parameters and pass them to the main window only if they are legal. The board size should be specified as positive integers, choosing between different options can be implemented using *radio buttons* so that the user can only make one choice. Any messages to the user concerning the parameters can be displayed in the Message Display text field in Figure 2. Of course the dialog window can provide default options for the various items.

The modal dialog window is only dismissed if the user clicks the **OK** button and all the information specified is valid or when the user clicks the **Cancel** button, in both cases the main window is informed appropriately. Since this window is modal, the user must dismiss it (either by clicking the **OK** button or the **Cancel** button) in order to proceed with the game in the main window.

Figure 2

Designing the Internals of the Game

The five interfaces given below represent the contracts that can be used to implement the game of FIAR. The internals of the game should be designed so that they know nothing about the GUI.

The interface `IGame` specifies the contract for playing the game. It obviously uses a game board and two players. The game board must implement the `IGameBoard` interface and a player must implement the `IPlayer` interface. The interface `IGame` specifies the method `makeMove()` which can be used to make a move in a particular column for a particular player. The Main Window should know nothing about the internals of the game except what the `IGame` interface provides.

The game board has a two-dimensional array of slots. One strategy for book keeping of the slots is to create and insert them in the two-dimensional array as the players make their moves. Adding a piece to the game board then means inserting the right kind of slot (explained below) and keeping track of which player made the move in this slot.

There are two interfaces for slots: `ISlot` and `ISlotCheck` which inherits from `ISlot`. Slots of the type `ISlot` do not check for a winner but slots of the type `ISlotCheck` do. This means that the type of slot inserted into the game board is dependent on the variation of the game that is being played.

Note that references of superinterface `ISlot` can also be used to denote slots implementing the subinterface `ISlotCheck`. Downcasting can then be used to invoke the specialized behavior of such slots. For the variations of the game in this assignment, at least three classes must be implemented: one type of slot where no winner checking is to be done, second type of slot for the standard version of the game (four in a row, column or diagonal) and third type of slot for the variation where the four pieces should be in a diagonal to declare a winner. See the section below on checking for a winner.

```
/**
  * Interface for implementing the game of FIAR
  */
interface IGame {
    /**
      * Makes the move in the specified column for the specified player
      * @return int The row number in which the move was made if legal move,
      *             otherwise value < 0.
      */
    int makeMove(int col, int playerNum);
    /**
      * @return IPlayer The player who has this particular number.
      */
    IPlayer getPlayer(int playerNum);
    /**
      * @return IPlayer The current player who made the last move.
      */
    IPlayer getCurrentPlayer();
    /**
      * @return IPlayer The player who has won if the game is over,
      *                 otherwise null.
      */
    IPlayer getWinner();
```

```
    /**
      * @return IGameBoard The game board on which the game is being played.
      */
    IGameBoard getGameBoard();
    /**
      * @return True If the game is over.
      */
    boolean isGameOver();
}

/**
  * Interface for implementing a game board
  */
interface IGameBoard {
    /**
      * Standard FIAR - four in a row, column or diagonal.
      * Program checks for winner.
      */
    int FIAR_NORMAL_GAMETYPE = 0;
    /**
      * Four in a diagonal only.
      * Program checks for winner.
      */
    int FIAR_DIAGONALS_ONLY_GAMETYPE = 1;
    /**
      * No checking.
      * Players decide the winner.
      */
    int NO_CHECKING_GAMETYPE = 2;

    /**
      * @return ISlot[][] The slots in the game board
      */
    ISlot[][] get2DimArrayWithSlots();
    /**
      * @return True If specified column number is legal for a move
      * on the game board.
      */
    boolean isLegalColumnNumber(int col);
    /**
      * Inserts a slot in the game board for the specified column and player.
      * The type of slot inserted is dependent on the type of game
      * being played.
      * @return ISlot The slot was inserted.
      */
    ISlot addPiece(int col, IPlayer p);
    /**
      * @return int The number of columns in the game board.
      */
    int getNumColumns();
    /**
      * @return int The number of rows in the game board.
      */
    int getNumRows();
}
```

```java
/**
  * Interface for implementing a slot in the game board.
  * Note this type of slot does not check for a winner.
  */
interface ISlot {
    /**
      * @return int the row of the slot in the game board.
      */
    int getRow();
    /**
      * @return int the column of the slot in the game board.
      */
    int getCol();
    /**
      * @return IPlayer the player who made a move in this slot
      */
    IPlayer getPlayer();
}

/**
  * Interface for implementing a slot which checks for a winner.
  */
interface ISlotCheck extends ISlot {
    /**
      * It checks to see if a winner can be declared depending on
      * a particular criteria for declaring a winner.
      * @return true If the move made into this slot resulted in a winner.
      */
    boolean checkForWinner(IGameBoard gb);
}


/**
  * Interface for implementing a player
  */
interface IPlayer {
    /**
      * @return int The number of the player (1 or 2)
      */
    int getPlayerNumber();
    /**
      * @return Color The color of the player (Color.red or Color.blue)
      */
    Color getPlayerColor();
}
```

Checking for a Winner

A slot which checks for a winner must implement the interface `ISlotCheck` which specifies the method `checkForWinner()`. Note that this method takes the game board (`IGameBoard`) as a parameter. A slot can therefore access the two-dimensional array of slots in the game board, making it possible for the slot to access its neighboring slots.

It is sufficient to check the current slot (in which the current move was made) for the winner criteria in order to decide whether the game is over. Had there been a winner, it would have been discovered in the previous move.

For game variation A, Figure 3a shows which neighboring slots of the current slot ought to be examined. Figure 3b is the analogous for game variation B. Note that the number of neighboring slots will vary depending on the location of the current slot on the game board. Note also that the current slot can be in any position in any sequence of four slots in a specified direction.

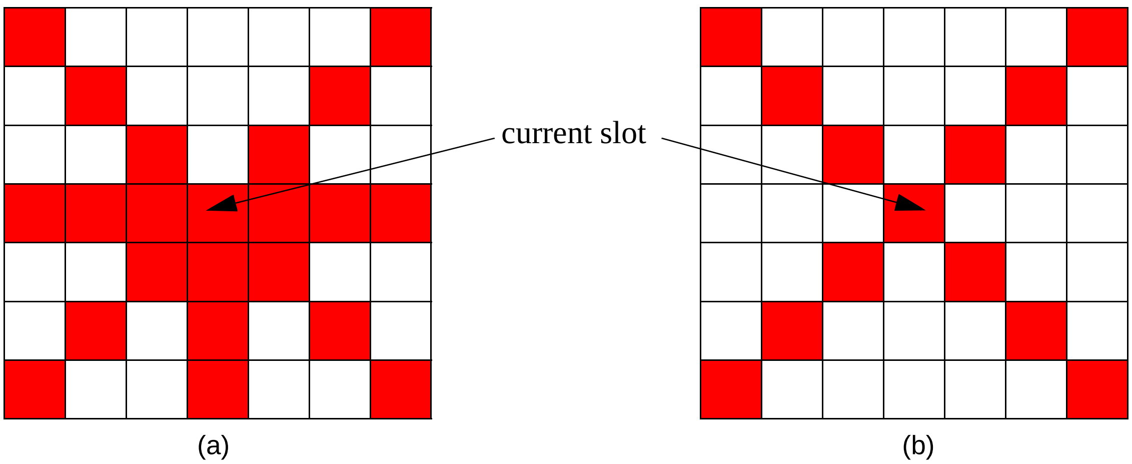


Figure 3

Deliverables: (No hand-written documentation will be accepted.)

- A cover sheet with only the following information. Failure to do so will be penalized by at least 5 points.
    - The name of the course and semester: CS211, Spring 2000
    - The assignment number
    - The assignment due date
    - Your name, identification number, e-mail address and the section you attend.
    - Your partner's name, identification number, e-mail address and the section your partner attends (if you have a partner)
- Class diagrams (at least one for the GUI and another one for the Internals) showing the relationships between the classes and interfaces.
- A sequence diagram that shows what happens when a player clicks to makes a move.
- Containment hierarchy for each top-level window, indicating the layout manager used for each container.
- Code for all the classes. The code should be documented using `javadoc` comments. The generated html documentation files should not be submitted.