CS180 Project 3: Simple Checkers Applet

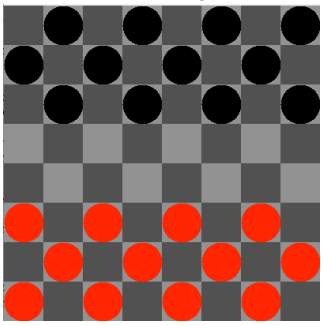
Chapters from the textbook relevant for this project: Chapter 7, Chapter 12
Project created by: John Franklin Jr.. Edited by: Aditya Mathur
Project created on: 8/8/2011
Project assigned on: 10/14/2011
Project due date: 10/28/2011. 11:59pm.
TEAM PROJECT: You may work on teams of up to 3 members. ONLY ONE PROGRAM IS TO BE SUBMITTED BY EACH TEAM.

1. Learning objectives:

1. Manipulating multi-dimensional arrays.
2. Creating graphical user interfaces.

2. Requirements:

Create a simple two-player checkers applet. The applet will have simple functionality. Not all standard rules of the game need to be implemented. The following rules are the only rules required of the applet.



- A new game starts with a standard checker setup, like seen above where each player has 12 chips.
- Chips can only move diagonally.
- Spaces are highlighted of possible moves after clicking on a chip.
- Player's chips can only be moved during their turn.
- Display, below the board, the count of each player's chips.
- Display, above the board, the name of the player whose turn it is.
- Chips can only jump over one opponent per turn.
- Must remove taken pieces from the board.
- Place a button, below the board, that starts a new game.

3. Files:

`Board.java`
`CheckersApplet.html`

There is no need for you to modify `Board.java`. It contains the `Board` class. This class has the following methods to help you program this project. [Of course, you are welcome to view the `Board.java` file.]

- Constructor named `Board` that does not take any parameter and creates an 8x8 board.

- `public void` setState(int x, int y, int s): This method updates the state of the checker board.

  where `x` and `y` are the coordinates of the board to be updated, and `s` denotes the color of the chip to be placed at these coordinates. The black chip is referred to as Board.BLACK and the red chip as Board.RED. Place Board.HIGHLIGHT at a position to highlight it. An "empty" or "blank" chip is referred to as Board.EMPTY.

  Example of use:

  Suppose that `b` denotes an object of type `Board`, and p and q denote the coordinates of a position on the board.

  b.setState(p, q, Board.RED); // Place a RED chip at position (p, q).

  b.setState(p, q, Board.EMPTY); // Place an empty chip at position (p, q); this clears the position.

  b.setState(p, q, Board.HIGHLIGHT); // Highlight position (p, q).

- **`public int`** getState(int x, int y): This method returns the current state of the checker board at position x, y.

  Example:

  `int` chip =b.getState(p, q); // Returns the current state of the board b at position (p,q).

  Note that each chip is coded as an integer declared in the `Board` class. For example, Board.EMPTY is 0 and Board.RED is 1. You may look into the `Board.java` class to find the different codes.

4. Steps:

(i) Setup:

Create the `project3` folder in the `cs180` folder to hold all the source code files. Go inside the `project3` folder and open DrJava (or your favorite Text Editor) to begin developing the program.

Note:

1. A few commands to help you setup:

```
% cd cs180
% mkdir project3
% cd project3
% pwd
/u/u91/yourlogin/cs180/project3
% drjava Checkers.java
```

2. Import classes. For this project, you will need to import at least `javax.swing.*`, `java.awt.*`, and `java.awt.event.*`. You can import other classes if need be.
3. Open Java console for debugging.

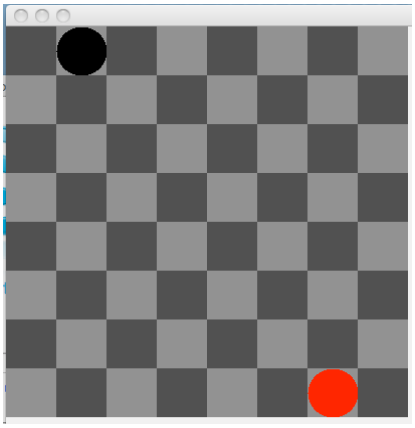
(ii) Learn to use the Board class

Prior to starting to work on coding the project, it would be helpful if you play around with the Board class. Doing so will help you understand how to use the methods in `Board`. The following examples should help you understand how to use `Board`.

Example 1: Write a program to create a Board object named `b`. Place a black chip in row 0 column 1. Place a red chip in row 7 and column 6. Add `b` to the frame and display the frame.

```
import javax.swing.*;
public class CheckersExample1{
   public static void main(String [] args){
      Board b=new Board();// Create Board object.
      JFrame f=new JFrame(); // Create a JFrame object
      f.setSize(500,500); // Set frame size
      b.setState(1,0, Board.BLACK); // Add black chip to b
      b.setState(6,7, Board.RED); // Add red chip to b
      f.add(b); // Add b to the frame
      f.setVisible(true); // Display frame
   }// End of main()
} // End of CheckersExample1
```

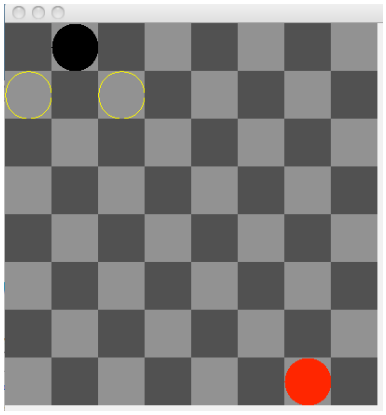
Output from CheckersExample1:



Example 2: Write a program to create a Board object named `b`. Place a black chip in row 0 column 1. Place a red chip in row 7 and column 6. Highlight the diagonals around the black chip. Add b to the `JFrame` and display the frame.

```
import javax.swing.*;
public class CheckersExample2{
public static void main(String [] args){
        Board b=new Board();// Create Board object.
        JFrame f=new JFrame(); // Create a JFrame object
        f.setSize(500,500); // Set frame size
        b.setState(1,0, Board.BLACK); // Add black chip to b at column 1 row 0
        b.setState(6,7, Board.RED); // Add red chip to b at column 6 row 7
        b.setState(0,1,Board.HIGHLIGHT); // Highlight column 0, row 1.
        b.setState(2,1,Board.HIGHLIGHT); // Highlight column 2 row 1.
        f.add(b); // Add b to the frame
        f.setVisible(true); // Display frame
    }// End of main()
} // End of CheckersExample2
```

Output from CheckersExample2:

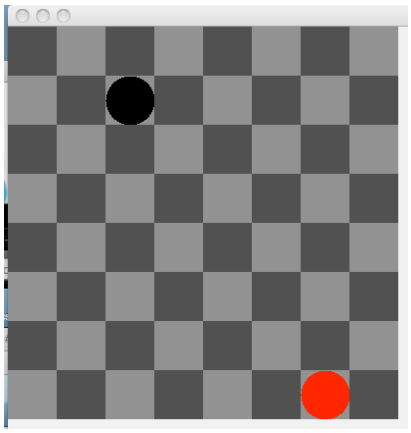


Example 3: Write a program to create a Board object named `b`. Add `b` to the frame and display the frame. Place a black chip in row 0 column 1. Place a red chip in row 7 and column 6. Highlight the diagonals around the black chip. Remove the highlights. Move the black chip to one of the previously highlighted positions. The code below performs the desired tasks. *Note that we have moved the code to add b to the frame and make the frame visible prior to placing the chips on the board.*

```
import javax.swing.*;
public class CheckersExample3{
    public static void main(String [] args){
        Board b=new Board(); // Create  Board object
       JFrame f=new JFrame(); // Create a JFrame object
        f.setSize(500,500); // Set frame size
        f.add(b); // Add b to the frame.
        f.setVisible(true); // Make frame visible
```

```
        b.setState(1,0, Board.BLACK); // Add black chip to b at column 1 row 0
        b.setState(6,7, Board.RED); // Add red chip to b at column 6 row 7
        b.setState(0,1, Board.HIGHLIGHT); // Highlight column 0, row 1.
        b.setState(2,1, Board.HIGHLIGHT); // Highlight column 2 row 1.
        b.setState(1,0, Board.EMPTY); // Remove black chip
        b.setState(2,1, Board.BLACK); // Place it at column 2 row 1
        b.setState(0,1, Board.EMPTY); // Un-highlight the other diagonal position.
    } // End of main()
}// End of CheckersExample3
```

Output from CheckersExample3:

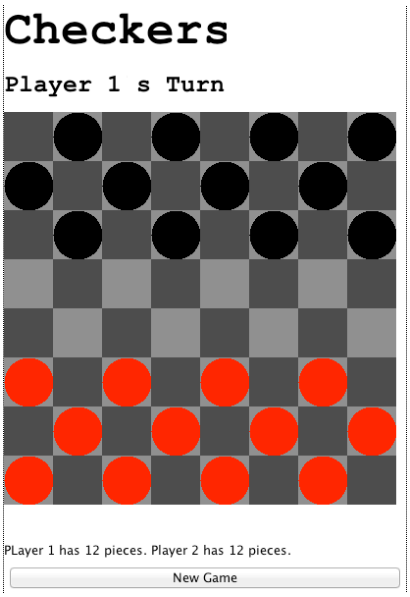


The above examples illustrate how to change the state of a Board object. You may write a few more programs to play with a Board object. Note that the GUI for the checkers game shows more than just the Board object!

IMPORTANT: *Make sure you create the initial Board object with all the chips on it using loops, and* ***not*** *by using one statement each to place a chip on the board.*

(iii) Create GUI:

Create a `JApplet` with a `BorderLayout`. Use the `North` section to display the title of the applet and name of the player whose turn it is, e.g. Player 1's Turn or Player 2's Turn. The center section should hold the board. An implementation of the board is given with `Board.java`. The `Board` class extends `JPanel` and can be treated as a `JPanel`. The `South` section should hold the count of the pieces on the board of both players and a button that starts a new game. Below is an example of what the GUI should look like.



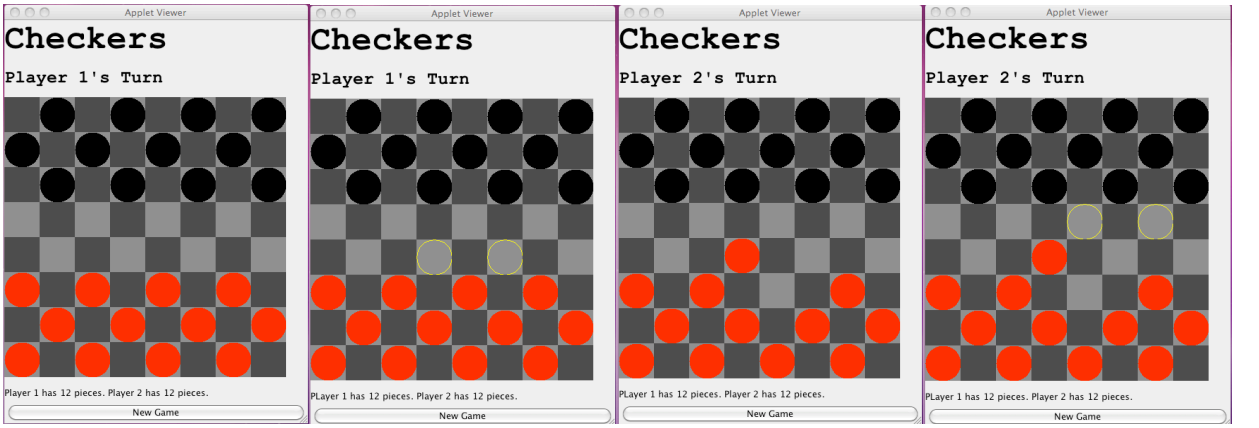
(iv) Board:

Create a `MouseListener` and add it to an instance of `Board`. To get the **`x`** and **`y`** coordinates of where a mouse click event occurred, use the `MouseEvent` object passed as a parameter to the `mousePressed()` method required by the `MouseListener`. Each block on the board has a width and height of 50 pixels. Once the x and y coordinates are retrieved from the `MouseEvent` object, divide those values by 50 to get the x and y values to be used for setting and getting the state of the `Board` object. There is no need to implement the remaining methods in the `MouseListener` interface.

Use the static variables within the **`Board`** class to set the state of each position on the board.

(v) Logic:

Compose your program as a collection of simple methods to split logic into different components. This allows for neater and less error prone code. Example methods would be `movePiece(), jump(), highlightLeft(), hightlightRight(), removeHighlights(), newGame()`. Of course, you may not use these method names and, instead, use method names of your choice. First 8 moves in a sample game are shown below.

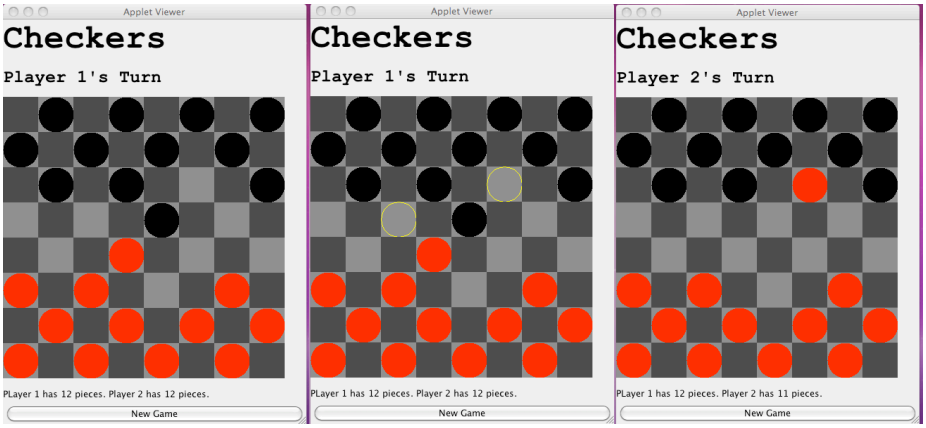


Player 1 clicks on chip at row 5 column 4.

Player 1 clicks highlighted position at row 4 column 3

Player 2 clicks on row 2 column 5

Player 2 clicks on highlighted row 3 column 4.



Player 1 clicks on chip at row 5 column 4.

Player 1 clicks highlighted position at row 4 column 3

5. Test cases:

Your program will be tested by playing the checkers game with your applet. There is no need for your applet to check if the game has ended.

Note that DrJava allows you to run your program as an applet assuming that the class you have defined extends `JApplet`. To run a correctly compiled program as an applet click on the `Tools` menu and select `Run Document as Applet`.

6. Grading:

Correct GUI layout, including the listeners	30%
Update GUI appropriately	10%
Correct highlighting	30%
Moving pieces, including jumping	30%

7. Project turn-in

1. At the terminal, move to parent directory of your Project 3 directory.
2. Type the following command replacing **`${myproject3}`** with your Project 3 directory name and **`${labID}`** with your lab section's code:
   `turnin -v -c cs180=${labID} -p project2 ${myproject3}`
3. Verify your submission by typing the following again replacing **`${labID}`** with your lab section's code:
   `turnin -c cs180=${labID} -p project2 -v`
4. **Each team must turn in only one program. Each team member's name and login ID must be at the top of the file turned in. Any one member of a team may turn in the project. Only the last project file turned in prior to the project deadline will be graded.**

<End of CS 180 Project 3>