# COP 3330
# Homework 6

## Out: 3/30/12 (Friday)
## Due: 4/13/12 (Friday) at 11:55 PM WebCourses time
**Late submissions accepted (with penalty) until 4/15/12 (Sunday) at 11:55 PM**

## Objective

1. To familiarize you with designing your own classes to solve a problem.
2. To get you accustomed to writing documentation using the Javadoc tool.

## Code

You may discuss high-level ideas of how to solve these problems with your classmates, but do not discuss actual source code (except for boiler-plate code such as opening a file for reading, reading user input, etc.). All source code should be yours and yours alone. **Do not:**

- Share your source code with anyone
- Ask anyone to share his/her source code with you
- Ask/pay a programming forum/community to write your homework for you
- Pair program
- Do anything else that you know is deceptive, dishonest, or misleading in any way

## Problem: Checkers

For this homework assignment, you must implement a simplified game of Checkers, with a GUI for playing the game.

Here is a list of the rules of the full game:
http://boardgames.about.com/cs/checkersdraughts/ht/play_checkers.htm

In the simplified version, we will omit rules 4 (we don't care about who plays first), 7 (partially - just the multiple jump part), 9, 10, 11, 12, and 13 from that list.

So to sum up the rules, pieces can only move once (either normally, or with a jump to capture an enemy piece). Multiple jumps in the same turn are not allowed, and a player is not forced to make a capture just because they can. We won't have any kings in the game, and the game ends when one player (the loser) is unable to make any moves - either because they have no pieces left, or their remaining pieces cannot move.

## Requirements

There are relatively few constraints on how you implement this (also see the Extra Credit section later), but at the bare minimum we'd like the following functionality:

1. Display a checkerboard, with game pieces of both colors.
2. Players must alternate turns.
3. On each turn, a player can 'select' a piece by clicking it. Clicking again should deselect it. Only a single piece can be selected at a time. If a new piece is selected, the program should deselect the old piece.
4. There should be some visual indication that a piece is selected, like lighting it up, giving its square a distinctive border, or something similar.
5. When a piece is selected, the squares it may move to should also be indicated visually in some way. (Perhaps by highlighting them.) If a player clicks on one of these destination squares, the selected piece should move to that square, capturing an enemy piece if the move is a jump.
6. Captured pieces should be removed from the board.
7. When the game ends, the program should indicate the winner in some way. A popup dialog is one way to do this.
8. A menu bar with at least one menu containing the items “New Game” and “Exit”.

You should create classes for all relevant aspects of the problem. For example, you might have a Board class, a Piece class, a Game class, a class to handle the GUI, etc. These are just recommendations, though – you are free to design the classes any way you want. Just make sure you follow reasonable principles of object-oriented design – information hiding, each class handles a particular task, classes are built by composition, use of inheritance and interfaces (optional, don't force yourself to use either unless your design demands it), etc.

Caution: It's easy to fall into the trap of trying to code in Java the way you might code C – a single class and a bunch of functions. This is the **Wrong Way** to do things in an Object Oriented language, and will be penalized. The purpose of this assignment is specifically to make you write Java code the way it is meant to be written.

Place each class in its own .java file (unless of course it's an inner class). You must write Javadoc comments in each class, so that the Javadoc tool can generate documentation from them. This will account for a significant part of the grade for this program.

Your main class should be named **SimpleCheckersGUI.java**. This is the one we will run to grade your program, so it is where your *main()* method should be placed. We'll ignore main() methods inside any other class.

## Sample I/O

A video of a working solution to this assignment will be posted soon. Keep an eye on the Webcourses Announcements section for a link to it.

## Extra Credit

As with homework 4, a separate assignment will be created for up to 20 points of extra credit. In this one you are free to build a feature-rich version of the game. Examples of features you can add include:

1. Kings
2. Multiple jumps in one move (like in regular checkers)
3. Actual graphics for the board and the pieces (as opposed to shapes on a Canvas).
4. Undo last move
5. Drag and drop movement

This list is not exhaustive, so feel free to add anything else that you think would be cool. As before, limit your extra features to the extra credit assignment, and make sure you submit the standard one too.

For this, your main class should be called **CheckersGUI.java**. You must also submit a README file listing all the features you've implemented.

## Deliverables

Submit the source files over Webcourses as an attachment. **In particular, *do not* submit any .class files!!! This will result in 0 credit for the assignment.** You must send your source files as an attachment using the "Add Attachments" button. Assignments that are typed into the submission box will **not** be accepted.

## Restrictions

Your program must compile using Java 6.0 or later on the command prompt. It's okay to develop your program using the IDE of your choice, but the TAs will use the command prompt to compile your code and run the programs. Your code should include a header comment with the following information:

Your name
Course number, section number
Description of the code

You **WILL** lose credit if this information is not found in the beginning of each of your programs. You should also include **inline comments** to describe different parts of your program where appropriate.

## Execution and Grading Instructions

1. Download the source *.java* files and place in a folder.
2. Check the source code of each program to make sure it contains header comments, inline comments and reasonable use of variable names.

3. Run the program and test to see if all the required functionality is present.
4. If all required functionality is present and works correctly, give full credit for execution. Otherwise, give partial credit based on the situation.
5. Run the javadoc tool to generate documentation from the comments.
6. If the javadoc is satisfactory, give full credit for documentation. Otherwise, give partial credit based on its quality.

## Points Breakdown

- **Code/Design**: 30 points
- **Execution**: 50 points
- **Javadoc comments**: 20 points