ASSIGNMENT 3
Object Oriented Programming

COMP-202B, Winter 2011, All Sections

Due: Wednesday, February 23rd, 2011 (13:00)

You **MUST** do this assignment individually and, unless otherwise specified, you **MUST** follow all the general instructions and regulations for assignments. Graders have the discretion to deduct up to 10% of the value of this assignment for deviations from the general instructions and regulations.

Please read this entire document before starting. Important information may be contained in the warm-up questions.

Part 1:	0 points
Part 2, Question 1:	30 points
Part 2, Question 2:	70 points
	100 points total

Part 0: Information

We will talk extensively about the following concepts in class. However, we present here a brief summary which, combined with reading documentation, should be enough to get you started.

Defining an `enum`

An `enum` is a user-defined type. The values that the `enum` can take come from a pre-defined set of constants. An example of an application for an `enum` could be to store a day of the week.

To define an `enum` type, first decide on the name of your new type, and create a `.java` file with that name. (Much as you do with a class name). Then inside that file, you will just have one statement:

`public enum ENUMNAME { list of values }`

For example, to define an enum for the days of the week, you would create a file `Day.java` and inside that file have the following statement.

```
public enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY }
```

To use the new type you have created, you can now declare a variable of this new type, or access its values by the following

`ENUMNAME.ENUMVALUE`

For example, after defining the prior enum `Day` you could have in another class

```
Day happyDay = Day.FRIDAY; Day[] danLectures = { Day.MONDAY, Day.WEDNESDAY, Day.FRIDAY} Day[]
majaLectures = { Day.TUESDAY, Day.THURDAY }
```

You will then need to compile the file `Day.java` at the same time (i.e. by writing javac Day.java YourClass.java) or beforehand (i.e javac Day.java and then subsequently javac YourClass.java) as your class that will use it.

Note that enums are like primitive types as they are constants. They are not references.

Creating objects

Remember that an `object` in Java is a class that normally has both *attributes* and *bahviours*. To define an object, you will create a class in its own file as you normally do.

To define attributes of an object, you add variables inside the class that are not part of any method. To define behaviours of an object, you add methods inside the class. For example if I create a file `Chef.java` with the following contents:

```
public class Chef
   private String[] recipes;

   public int numberKnownRecipes;

   public Chef() {

   }

   public void cookRecipe(String dishName) {
             //......some actions
   }

   private void heatPan() {
             // ....some actions
   }
}
```

I would be defining the type `Chef` as an object that has 2 properties–a list of recipes (stored as an array of `String`) and an integer which stores the number of recipes the chef knows. An object of type `Chef` knows how to do two actions—cookRecipe and heatPan.

Once I have defined a type of object, I can create instances of that type elsewhere in my program or class. In other words, once I define what a Chef is, I can create as many actual Chefs as I want elsewhere in my program. Objects are always reference types. To declare a variable that can store the address of a `Chef` we could write `Chef ratatouille`

To actually, *create* a `Chef` (allocate it in computer memory), or any object, we must use the `new` keyword. This is very similar to how we allocate an array.

```
Chef someChef = new Chef()
```

Once we have created a `Chef` we have access to some of its properties and methods. We can do this by writing the name of our variable, followed by a `.` followed by the name of the attribute or behaviour we wish to use. For example:

```
int x = someChef.numberKnownRecipes;
```

will assign to `x` the value of `numberKnownRecipes` inside of `someChef`.

Any property or method that we write `public` before, is accessible outside the method this way. Any property or method that we do not write `public` before, or write `private` before, is not accessible. For example, we can not write:

```
someChef.heatPan();
```

because the method `heatPan` is defined as `private`

Additionally, classes can have a *constructor*. A constructor is a method that must have exactly the same name as the name of the class. It does not have any return type and is called only one time per instance of an object—at the moment the object is created. In other words, when you write

```
new ObjectName()
```

you are calling the constructor of `ObjectName` with no arguments. A constructor is normally used to initialize some values inside of a class.

If one wanted, they could define a constructor which took arguments. This is done the same way as with normal methods. For example, we could in the `Chef` class

```
public Chef(String[] recipeList) {
  recipes = recipeList;

}
```

Then when we create an instance, we could write:

```
String[] recipes = { "Pizza" , "Pasta" , "Cappucino" };

Chef italianChef = new Chef(recipes);
```

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions.*

Warm-up Question 1 (0 points)

Practice with enums, arrays, and classes

In this program you will write a program to shuffle cards.

To do this, first you should define 2 different enums: `Suit` and `Value`

`Suit` can be spades, hearts, clubs, or diamonds.

`Value` can be ace,2,3,4,5,6,7,8,9,10, jack, queen, king

You should then define a class `Card` which has 2 instance variables: a variable representing a `Suit` and one representing a `Value`. Your class should then have 5 methods, 2 each to get the values of the variables, and 2 to set them and an additional method that prints the values. Note that to print an enum you can put it into a System.out.println statement as normal.

You should write a main method which should do the following:

- Create a new deck (i.e. an array) of 52 cards. It should fill these in with the appropriate values that are sorted.
- Print the entire deck using the print method defined in the card class.

- Call a method called shuffle() that you will write and will take as input an array and returns void. It should, however, modify the values inside the array. Your shuffle method should pick 2 random integers between 0 and 51 (inclusive). It should then swap the cards that are at those locations in the array. This should be in a for loop so that the method repeats NUMTIMES (a final declared at the top)
- Print the entire deck after shuffling

To choose a random integer, you will need to use a random number generator. In Java, to do this, you first have to import the class `java.util.Random`

Then make a variable of type `Random`. You should make this a **class** variable. What this means, is that you should write the word `static` before the type and name of the variable.

Recall that to do this, you declare the variable outside of any other method in the class.

In your main method, create a new instance of the `Random` class and call a method `shuffle`

Now inside your shuffle method, you can write

`random.nextInt(52)` to get a random integer between 0 (inclusive) and 52 (not inclusive). (Assuming of course that you have named your variable random.)

Note that to compile your program, you will have to include all files on the same line. i.e.

```
javac Value.java Suit.java Card.java CardShuffler.java
```

Warm-up Question 2 (0 points)
Practice using loops and enums

Inside the `ConnectFour.java` class, write a method `printBoard` that prints a 2d array of type `Piece[][]` that it takes as input.

Every RED piece should be displayed as an 'O'

Every BLACK piece should be displayed as an 'X'

Every EMPTY square should be displayed as an '_'

Because it will be hard to read otherwise, you should add a space between every column and a new line between every row.

Warm-up Question 3 (0 points)
Practice with classes In this exercise, we will design a class to keep track of the moves of a connect-4 game and will give you practice with writing a class.

The task is to design a class `GameLogger` which can keep track of a sequence of moves in a Connect Four game.

Your class will have 1 attribute, an array of ints representing "move numbers".

Your class should have the following methods:

- `public GameLogger()` This constructor method should initialize an array instance variable `moveList` of size 2. It should also set a instance variable `size` to be equal to 2.
- `public void addMove(int newMove)`

  This method should add the value of `newMove` to the end of the array stored as a instance variable. If the array is full, it should a)create a new temporary array of size `size` b)copy all values from `moveList` to this temporary variable, c)create space to store `size*2` values using the `new` keyword and store this array in `moveList` d)copy all values from the temporary array to `moveList` e)set `size` to be equal to `size * 2`. Then finally add the value of `newMove` to the array.

- `public void printGameFromList()`

  This method should print reconstruct the game from the list of moves and print the list by writing the player and then the move and then a new line. You method may assume that the first player to go is "X" and the 2nd player to go is "Y"

  For example, if the move list is `1,2,3,4` your program should print

```
X 1

Y 2

X 3

Y 4
```

Part 2

The questions in this part of the assignment will be graded.

Question 1: Vector Class (30 points)

This question will give you practice with classes. It will also provide practice with loops. The methods you write will be inside of a class.

Create a class called Vector. The class Vector must contain 3 instance variables. Each instance variable **must** be declared as private.

```
private double x;
private double y;
private double z;
```

You class should have the following public methods:

1. `public Vector(double xcoord, double ycoord, double zcoord)`

   This is the constructor of your vector class. It should initialize 3 private instance variables, called `x`, `y`, `z` with the values of the input variables, `xcoord`, `ycoord`, and `zcoord` respectively.

   Note that because the constructor has 3 input variables, you will only be able to create a `Vector` by providing 3 numbers.

   For example:

```
Vector origin = new Vector(0,0,0);
Vector equal = new Vector(1,1,1);
```

2. `public double getX()` This method should return the value of the instance variable `x`
3. `public double getY()` This method should return the value of the instance variable `y`
4. `public double getZ()` This method should return the value of the instance variable `z`
5. `public Vector add(Vector v)`

   This method should return a new Vector which is calculated by adding `this` vector–i.e. the vector which is doing the behaviour–to the vector passed to it as a parameter.

6. `public double dotProduct(Vector v)`

   This method should compute the dot product of `this` vector with the vector given as parameter. Note that to compute the dot product of 2 vectors, you multiply each component and then add the sums.

   For example:

$$(x_1, y_1, z_1) \cdot (x_2, y_2, z_2) = x_1x_2 + y_1y_2 + z_1z_2$$

```
Vector v1 = new Vector(1,1,1);
Vector v2 = new Vector(2,3,1);
double dotProduct = v1.dotProduct(v2);  //double is now 2+3+1=6
```

7. `public double calculateMagnitude()` This method should compute the magnitude of the vector. One can compute the magnitude of a vector by summing the squares of all coordinates and taking the square root.

   That is:

$$magnitude = \sqrt{x^2 + y^2 + z^2}$$

8. `public void normalize()`

   This method should normalize the `this` vector. Note that this is the only method which will change the value of the current vector instead of returning a new value.

   If the vector has magnitude of 0, then your normalize() method should return without making any changes. This is to avoid dividing by 0

   One can normalize a vector by dividing each of a vector's components by the magnitude of the vector.

9. `public Vector scaleVector(double factor)` This method should scale the vector by the factor listed as parameter.

10. `public Vector projectOnto(Vector v)`

    This method should project the vector on which the behaviour is acting onto the vector `v` passed in as a parameter and return the corresponding vector. Note that the projection of vector $\vec{u}$ onto vector $\vec{v}$ is defined as :

$$\frac{\vec{u} \cdot \vec{v}}{|v||v|}\vec{v}$$

    If the vector $\vec{v}$ has magnitude zero, you should simply return v to avoid dividing by 0.

Inside the file `Vector.java` is a very basic start to the `Vector` class to help get you started.

As in the previous assignment, we have provided a set of test cases for you to run your program on. These are in the file `VectorTest.java`

You are strongly encouraged to do additional testing on your own as you develop the class. To do this, you can declare a main method inside the `Vector` class. Inside this, you can create new Vectors. For example:

```
 public static void main(String[] args){
Vector v1 = new Vector(2,3,4);
Vector v2 = new Vector(3,4,5);
System.out.println("The value of v1 dot v2 is " + v1.dotProduct(v2);
}
```

Note that this main method will not be graded.

Question 2: Connect Four (70 points)

In this exercise, you will work with a skeleton program to produce a large part of the game Connect Four. See `http://en.wikipedia.org/wiki/Connect_Four` In this, you will write much of the board logic to determine things like who has won the game. You will also design part of the Artificial Intelligence for the computer player.

This will provide practice in using 2-dimensional arrays, classes, and nested loops.

Inside the code directory there are several files:

- `Player.java`
- `HumanPlayer.java`
- `Board.java`
- `ComputerPlayer.java`
- `ConnectFour.java`
- `Piece.java`

You will add methods to the files `HumanPlayer.java` , `Board.java` , and `ComputerPlayer.java`

Any changes you make to the other files will not be graded. You may change the file `ConnectFour.java` to do the warm-up exercises, but you must not change any other files.

Throughout this program, we will represent the connect four board as a 2-dimensional array of type `Piece` which is an enum defined to be either EMPTY, RED, or BLACK. We will use the first index of the 2d array to represent the row number and the second number to represent the column number. We will call row 0 the bottom row, and column 0 the left most row.

So for example, if the board is stored in a 2d array called `theboard`, then `theboard[0][2]` refers to the square located in the bottom most row and 3rd column from the left. We will use a connect four board which has 6 rows and 8 columns.

(a) (5 points) HumanPlayer class You must fill in the following method to `HumanPlayer.java`

- `public int chooseMove(Board b)`

  This method should read an `int` from the user. This is a number from 0-7 that represents the column, going from left to right, of the move to be made. It should then return the number that the user entered. Note that you do not need to do any sort of checking here to make sure the number is between 0 and 7, but you must return an int.

(b) (40 points) Board class

The Board class must contain the following private instance variables:

- A 2-d array of type `Piece` called `theBoard` representing the piece on each square. The type `Piece` is defined using an `enum` in the file `Piece.java` and can take 1 of 3 values: *RED*, *BLACK*, *EMPTY*

  The first index of this array will be used to store the row and the second index will store the column.

  For example

  `theBoard[2][3]` will store whether the 3rd row and 4th column (since we count starting from 0) contains a red piece, black piece, or is empty.

  Note that row 0 is on the bottom and column 0 is on the left. For this example, we will use 6 total rows and 8 total columns.

- A boolean `isRedTurn` which will allow you to know whose turn it is by storing whether it is redsTurn or not

In addition, the board class should contain two *public* identifiers.

- A final int NUMROWS which will be set to 6
- A final int NUMCOLS which will be set to 8

Throughout the board class, you should never refer to the value 6 or 8 directly. You must always refer to the final described above.

In addition to these variables, you must write the following methods:

- `public Board(Piece[][] oldBoard)`
  This constructor will create a board from an array of Pieces. It **must** copy values one at a time from the original array `oldBoard` to the new one. There must be two distinct 2-d arrays after this is created.

  That is, it is not sufficient to write

  ```
  theBoard = oldBoard;
  ```

  The constuctor should also set the boolean `isRedTurn` to be true.

- `public Piece getPiece(int row, int col)`
  This method should return the piece that is located at the specified row and column. Note that the row and column indices both start from zero. So to access the element in the bottom-left most spot, one could write

  ```
  Piece lowerleft = getPiece(0,0);
  ```

- `public Piece[][] getPieceArray()`
  This method should return the entire 2dimensional array of pieces. Note that you should just return the 2d array. You do not and must not make a copy of the array.

- `public int findFirstFreeRow(int column)`
  This method should return the first empty space in the specified column. For example, if the first three rows of a column are filled, it should return 3. Remember that the first row is labeled row 0.

  If there are no free spaces in the column, you function should return -1

- `public boolean isLegal(int nextMove)`
  This method should return a boolean representing whether the current move is a legal move or not. A move is legal provided that

  - nextMove is between 0 and NUMCOLS (inclusive of 0 but not of NUMCOLS)
  - There is an open space in the row

- `public boolean existLegalMoves()`
  This method should check if there exist any legal moves on the board. Do not worry whether one side has already connected 4 in a row

- `public Piece getWinner()`
  This method should check if either player has already connected four. It should return Piece.Black if black has won, Piece.Red if red has won, or Piece.Empty if there is no winner.

  This method must work any time that there are four *consecutive* pieces in a row/column/diagonal of the same colour, regardless of whether it is a "legal" sequence of connect four moves. You can not rely on whose turn it is, for example.

- `void updateBoard(int moveChoice, Piece move)`
  This method should take as input a specific column and update the board by putting the piece whose turn it is in the first free row of the specified column. You may assume that the move is legal. Remember to update the boolean `isRedTurn` to correspond with the new board state.
- `public static Board GenerateEmptyBoard()`
  This static method should create a new Board and fill it in with empty pieces. It should then return the Board. The number of rows and columns should be based off of the constants NUMROWS and NUMCOLS

(c) (25 points) Artificial Intelligence Part
In this part of the question, you will modify the program `ComputerPlayer.java` . Inside this file you will find a part of the artificial intelligence already written.

A commonly used algorithm for artificial intelligence is called the **minimax** algorithm. What this algorithm does is at any given step, it will search through every possible move. It will then choose the move which leads to the best conclusion in the *worst* case, which is that the opponent makes his or her best move.

For example, if the computer has 5 choices of moves, he will pick the one that leads to the best position when its opponent makes his best move.

Due to the limited resources of computers, it is not normally possible to search through every possible move. Computers will thus search to a fixed depth (maybe 5 or 10 moves for example), and then at that point make a "guess" or *heuristic* as to who is winning. Thus a heuristic is simply a way for the computer to evaluate the current position and make some sort of guess as to how well it is doing.

In this question, we have provided you with the code to search through moves and select the best ones. You will write the heuristic function for a Connect Four artificial intelligence.

For this part of the question, you should only modify the method `heuristic()` Note that you can (and should) add other additional methods if you want to the ones that already exist.

The heuristic method will take as input a `Board` and a `Piece` representing the turn. It will then output the evaluation of how it views the position.

```
public double heuristic(Board b, Piece p)
```

Your heuristic function should return a number based on the following.

- If there is an "imminent win" the method should return 1000. An "imminent win" means that the player whose turn it is has already connected 3 in a row and it is possible for him/her/it to make a fourth immediately. This requires that the player have a)three in a row connected, b)A fourth square that is open and c)The column of the 4th square is already filled up enough that that square can be immediately played. (See Table 1)
- Otherwise, if the player whose turn it is is already "forked," the method should return **-900**. A fork will happen if the opponent has 2 groups of three in a row that can not both be blocked at the same time. (See Table 2 and Table 3)
- Otherwise, if the player whose turn it is has an opportunity to "fork" the opponent the method should return 800, unless the move to "fork" the opponent causes the opponent to have an imminent win (See Table 4)
- If none of these cases apply, the method should return the number of squares for which the player has already connected three in a row and is one square away from a connect four minus the number of squares for which his opponent satisfies this. It does not matter if the square is blocked by an opposing piece or otherwise unplayable, but if there are 3 in a row and the 4th square is off the board, then it should not count.

Note that you should count any "non-edge" horizontal or diagonal group of three as TWO pairs of three. This is because there are 2 possible "connect fours" that can be made from these triplets. (See Table 5)

Finally note that all these cases are not the same as being 1 square away from connect four. It is only checking if you have 3 in a row with a possible 4. So the scenario where you have 2 pieces, a gap, and then a 4th piece does not count in any of the above cases.

Hint: It will be useful to write helper methods such as `boolean imminentWin(Board b, Piece p)`. Remember that you may also use the method `updateBoard` from the Board class, but you should make sure to create a new Board before doing so.

Hint 2:A very useful helper method to write is

`int getTotalTriples(Board b, Piece p, boolean requiredOpening)`

which returns the total number of times that player p has connected 3 in a row (not counting cases where the 4th in a row would be off the board). If *requiredOpening* is true, then it will verify that the 4th space is open. Otherwise it will not check.

What we have provided to you

We have provided to you several files.

- `Piece.java` This file defines the enum Piece which can take values EMPTY, BLACK, or RED
- `ConnectFour.java` This file controls the logic for a simple game of connect four.
- `ComputerPlayer.java` This file defines what a ComputerPlayer is. We have provided some methods for you, but you will modify the `heuristic()` method
- `Player.java` This file defines what a Player is. You should make no changes to this file
- `HumanPlayer.java` This file defines what a HumanPlayer is. You will write the method `chooseMove`

In addition, we have provided test cases for your classes. It is important to note that the test cases are not necessarily exhaustive and you will probably find it useful to write you own additional test cases in a main method (or if you feel ambitious by adding them to the test case files themselves). These additional test cases that you write will not be looked at by TAs.

- `BoardTest.java` This file can be used to test your `Board` class
- `ComputerPlayerTest.java` This file can be used to test your `ComputerPlayer` class
- `VectorTest.java` This file can be used to test your `Vector` class

To run these test cases, you can do something identical to what you did in assignment 2.

- First put all files related to the question in the same directory.
- Then compile **YOUR** code. You can do this by writing

  ```
  javac Board.java ComputerPlayer.java ConnectFour.java HumanPlayer.java Piece.java Player.java
  ```

  or alternatively, if you separate each question into a separate folder:

  ```
  javac *.java
  ```

  (You would do something identical for the vector class except with different file names)

Table 1: An example of an imminent win. If it is red's turn, then he has an imminent win because he can connect four immediately. There could also be vertical and diagonal imminent wins

-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
B	B	B	-	-	-	-	-
R	R	R	-	-	-	-	-

Table 2: An example of a fork. Even though it is red's turn, there is nothing he can do to avoid losing because black has 2 possible ways to win.

-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	B	-	-	-	-
R	R	-	B	-	-	-	R
R	R	-	B	B	B	-	R

Table 3: An example of a fork that is also an imminent win. In this case, the heuristic should output the return of an imminent win since it does not matter that red is forked (since he can immediately win anyway)

-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
R	-	-	B	-	-	-	-
R	R	-	B	-	-	-	-
R	R	-	B	B	B	-	R

Table 4: An example of a fork possibility. If it is BLACK's turn, he can put a piece into the 3rd column from the leftto set up a fork

-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	R	R
-	-	-	B	B	-	R	R

Table 5: Here BLACK has 2 triplets of three in a row. (Without any red pieces on the board, he would have 2 different squares on the board that could be filled at some point to connect 4. Red has 2 triplets: the horizontal one on the bottom row, and the vertical one on the furthest right rows. The diagonal does not count because there is no square to get a connect 4 from. (Since there are only 8 columns)

-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	R
-	-	-	-	-	-	R	R
-	-	B	B	B	R	R	R

- You then compile the test case via the following command:
  Windows `javac -cp junit.jar;junit.jar;.  BoardTest.java`
  Mac OsX `javac -cp junit.jar:.  BoardTest.java`
- To run the program, you run the connect four class, i.e. `java ConnectFour`
- To run the test cases, you write
  Windows `javac -cp junit.jar;.  BoardTest`
  Mac OsX `javac -cp junit.jar:.  BoardTest`

Useful suggestions for getting started

The following are useful suggestions for getting started on the assignment

- First get everything to compile, including the test cases. You can do this by writing "skeleton" methods that always return meaningless values (like 0 or false)
- If you can get everything to compile with the test cases, then it means you have the correct required methods written.
- Do the warm up exercises which include a printBoard() method.
- Write the changes to HumanPlayer.java that allow you to play as a human player.
- The 2 above steps will allow you to run the connectFour program via 2 human players, so that you can test your `Board` class as you write it. (Note that until you write some of the methods in the `Board` class you will still not be able to play again since the logic to update moves won't be present yet)
- You can then work on tackling 1 method at a time. If you try to tackle all the methods at once, it will be overwhelming, so take each method one at a time.

What To Submit

```
Vector.java
HumanPlayer.java
Board.java
ComputerPlayer.java
```

Spiritual Growth completely optional not for any credit section

We will post online instructions for how you can do fun things like play each other's ComputerPlayer. This is completely optional and will not gain you any extra credit on the assignment, but you may find it fun.