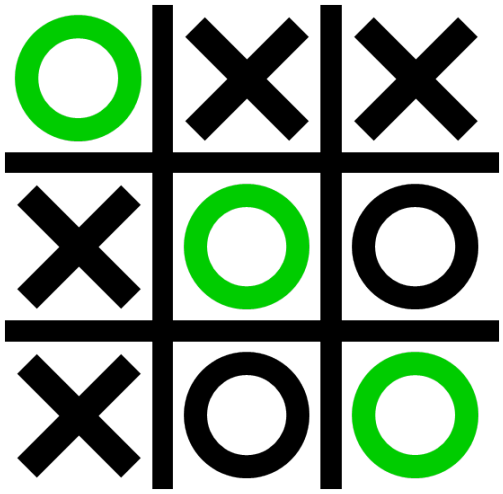
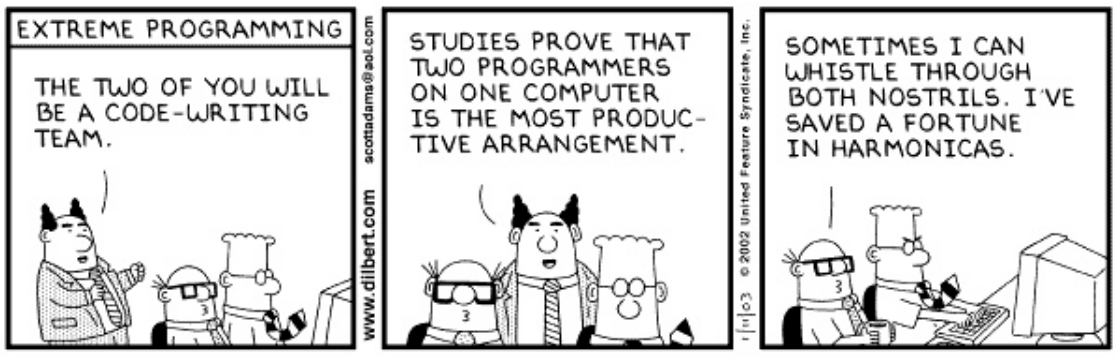
CS201 - Assignment 3, Part 2
Due: Wednesday March 5, at the beginning of class



For this assignment we will be developing a text-based Tic Tac Toe game[1]. The key to this assignment is that we're going to build up the parts of this game incrementally using an object-oriented approach!

For this assignment you may (and are encouraged to) work with a partner if you'd like. If you do, you must do **all** of the work together. That means if either of you is working on it, the other person should be there. Only turn in one copy of the assignment, but make sure both of your names are in the comments at the top of the file.



Before you start, read through all of this handout!

[1]For those who aren't familiar with the rules, see the Wikipedia page `http://en.wikipedia.org/wiki/Tic-tac-toe`

1 Game Overview

We will be implementing a text-based version of tic tac toe. Here is the output of an example game:

```
0 1 2
3 4 5
6 7 8

Pick a move X: 0
X 1 2
3 4 5
6 7 8

Pick a move O: 4
X 1 2
3 O 5
6 7 8

Pick a move X: 9
9 is not a valid move!
X 1 2
3 O 5
6 7 8

Pick a move X: 0
0 isn't empty!
X 1 2
3 O 5
6 7 8

Pick a move X: 8
X 1 2
3 O 5
6 7 X

Pick a move O: 6
X 1 2
3 O 5
O 7 X

Pick a move X: 2
X 1 X
3 O 5
O 7 X
```

```
Pick a move O: 1
X O X
3 O 5
O 7 X

Pick a move X: 5
Game over.
X O X
3 O X
O 7 X

X wins!
```

If O wins, then the game should indicate this instead. If it is a tie, i.e. all of the spaces get filled up, the game should end and print out that it's a tie.

2 Class Overview

To tackle this program, we're going to break our program into *four* classes. Although this may seem like it's making life more complicated, by breaking it down into separate classes, you'll see that most of the methods we will be writing are very simple. In addition, we can test these methods incrementally using our new found tool, `JUnit`.

The basic classes you will implement are:

- `Mark`: This class represents an empty space on the tic tac toe board. I will provide this class to you and you may not change it!
- `XOMark`: Once the game starts being played, we will start putting down X's and O's. This class will represent these and will be a subclass of the `Mark` class. By doing this, we will be able to keep a board that can store *either* a `Mark` or a `XOMark`.
- `Board`: This class will be used to keep track of the state of the board and most of the dirty work will be done in this class. In addition to keep track of the state of the board, this class will also have support for asking questions about the board such as if the game is over and whether or not someone has won.
- `TicTacToe`: By breaking the game up into smaller pieces, the actually tic tac toe class doesn't have much work to do. This class will handle the user input and the general game playing.

3 Class Details

For this assignment, I'm going to give you strict specifications for each of the classes, in particular, the `public` methods and, in some cases, the required instance variables. You must include all

of these methods in your implementation. In addition, these may be the **only** `public` methods these classes can have. However, you may add additional `private` instance variables and `private` methods to help you.

- `Mark`

  At: `http://www.cs.middlebury.edu/~dkauchak/classes/cs201/assignments/assign3/Mark.java`

  I have provided the `Mark` class. You may emphnot change this class. Look at it and make sure you understand how it works.

- `XOMark`

  The `XOMark` class must `extend` the `Mark` class and have the following methods:

  - `public XOMark(int, boolean)`, that is, a constructor that takes two parameters, the first and `int` and the second a `boolean`. The `int` specifies which board space (1-9) and the `boolean` whether or not this is an 'X' (`true` indicating an 'X').
  - `public boolean isEmpty()`. Is this square empty. *Hint:* XO squares will never be empty.
  - `public String toString()`. "X" if this is an 'X' and "O" if this is an 'O'.

  You will likely need some instance variables to help you out as well :)

- `Board`

  You must use a `private` instance variable for this class:

  ```
  private Mark[] board
  ```

  this will be used to store all of the board states (1-9). Note that since the `XOMark` class `extends` the `Mark` class, we can store *either* a `Mark` object in the `board` array or a `XOMark` object. Make sure that you understand this!

  The class must have the following `public` methods:

  - `public Board()`: Create a new tic tac toe board. This does *not* mean to display it.
  - `public void addXO(int number, boolean isX)`: Add a new entry on the board at number.
  - `public Mark getMark(int number)`: Get the `Mark` stored at number. Notice that this could be either a `Mark`, if the space is empty, or an `XOMark`.
  - `public boolean hasWin()`: Does the board contain a win for either side.
  - `public boolean gameOver()`: Is the game over? The game could be over because someone has won *or* if the board is full.
  - `public String toString()`: Should return a `String` version of the board, as we would like it displayed to the user. If a space is empty, then it should display the number and if it has a mark, then it should display the mark. *Hint:* both the `Mark` and `XOMark` already do all the dirty work for you for each spot. All you really need to do is call their `toString` method and put them in the right places.

- `TicTacToe`

  You much have a `private` instance variable for this class:

  `private Board board`

  In addition, you must have the following `public` methods:

  - `public TicTacToe`: Creates a new tic tac toe object, but doesn't start playing yet.
  - `public playGame()`: Play the tic tac toe game.

  To play a new tic tac toe game you would then write:

```
TicTacToe game = new TicTacToe();
game.playGame();
```

4 JUnit tests

At: `http://www.cs.middlebury.edu/~dkauchak/classes/cs201/assignments/assign3/Tests.java`

I have included a set of `JUnit` tests to help you:

1. Understand how each of the methods works. If you look at the individual tests you can see how each of the methods are called and what they're expected behavior is.
2. Help you test your code.

I *strongly* suggest that you use these tests for both purposes.

5 Requirements

- You must follow the output format shown in the game examples above.
- You can assume that the user enters a number, but you must check to make sure it's a valid number (0-8) and that the space is not occupied. If either of these occurs, a message should be displayed, the board should be shown again and that player should be asked to enter another choice.
- Your must follow *exactly* the class specifications above.
- When your program is done, all of the `JUnit` tests from `Tests` must pass.

6 How to make it all happen!

There are many ways to make this happen. However, here's how I suggest doing it:

1. Read through this whole document and make sure you understand everything.
2. Create your project. Download and add the `Mark` class. Create the `JUnit Tests` class (remember to do it in Eclipse by creating a new `JUnit` task.
3. Copy only those test related to the `Mark` class from the `Tests` class online. Run these and make sure they all pass.
4. Write the `XOMark` class.
5. Copy the tests related to the `XOMark` class from the `Tests` class online. Run them and make sure they all work.
6. Write the `Board` class. I suggest alternating between implementing some of the methods and copying the associated tests and checking to make sure they work. Most of these methods should be very simple! The only one that should be at all complicated is the `hasWin` method since you have to check many different win conditions.
7. Put it all together and write the `TicTacToe` game. Like we did with the `FlippyCard` game, start with the very basics (just having the board print) and then add other features slowly. Most of the work should be done already for you by the `Board` class, so make sure you're using those methods!
8. Play some tic tac toe.

7 When You're Done

Make sure you have comments at the top of any file/class that you wrote with with your name and the assignment number.

To group these files into a single file, you need to export your project. To do this:

1. Right-click on the project (on Mac, ctrl+click) and select `Export`.
2. You'll see a number of options. Open up the `Java` folder and select `JAR file` and click `Next`.
3. You should just see your project selected. Below, make sure **only** the following two options are checked:
   - "Export Java source files and resources"
   - "Compress the contents of the JAR File"

4. Click on the `Browse...` button and pick a location to save the output file. Give the file a name like "kauchak1.jar", where "kauchak" is your last name and "1" is the assignment number.

5. Click `Finish`.

If all went well this will generate a single `.jar` file wherever you picked to save it.

Submit this JAR file as assignment number "2.2" via the online submission mechanism on the course web page.