Background Information:
You are being asked to work in a team of 3 students to engineer a software product that represents a Scrabble computer game. Scrabble is a word game for 2-4 players, played on a 15x15 2-dimensional board of squares. Full rules for Scrabble can be found at web sites such as the following:

http://www.thepixiepit.co.uk/scrabble/rules.html
http://spikej.brinkster.net/scrabble/instructsc.asp
http://www.hasbro.com/scrabble/pl/page.rules/dn/home.cfm

The "customer" paying you to write the product is wealthy business tycoon, Marty Stepp, CEO of the large video game company, Tacoma Arts ("TA"). He intends to compensate your team for its services by awarding grade points.

Requirements:
The customer does not know exactly what he wants, but he does have the following requests:

- The product should display the company's logo prominently on the UI
- The product's income is ad-based, so provide space on the UI for an advertisement image
- The product should be as usable as possible, even for people who are not expert computer users
- The product's main functionality should let the user do the following things:
  - play a 2-player game with 2 human players
  - finish a game, then start a new game (in other words, it should be possible to play more than one game without restarting the program)
  - type in each player's name
  - view each player's name, tiles, and score
  - view a list of high scores, which saves and loads when the program starts and quits
  - make plays by placing tiles on the board
  - verify that moves are legal, and prevent a player from making an illegal move

- The product should have one "extra" feature. Your group can decide what you would like to use for your extra feature. Some examples include:
  - being able to play a game with more than 2 human players
  - being able to play against the computer
  - having more than one view of the game in the GUI (model-view-controller with multiple views)
  - having sound support in your game
  - having animated graphics in your game view
  - allowing two players to play a game over a network connection
  - allowing your game to run as an applet

Beyond these requirements, you are free to make UI decisions of your own. You should talk to your customer as you plan this project in order to make sure your product meets his needs. At least two meetings are expected: one to cover your requirements and design, and another to discuss your implementation and testing.
1. Software Requirements Specification (SRS)

The first milestone artifact you will submit for this project is a set of documents about your project requirements. Refer to Faulk's paper for basics about requirements and SRS documents. This project description can be considered a partial requirement specification, but much information is intentionally left out. This is to encourage your group to come up with questions to ask "the customer," the instructor. You may ask these questions in lecture, by email, or on the course message board. A submitted SRS that does not reflect questions and answers with the "customer" will not receive full credit.

Specifically, your SRS must contain the following documents:

1. **Use cases**
   - Submit the following use case documents:
     a. A UML use case diagram that depicts all the major use cases and actors in the system.
     b. Two formal use cases for scenarios you think are two of the most important to your product. They should be similar to Use Cases 1, 2, 3, and 5 from Cockburn's paper, and should include: primary actor, scope, level, stakeholders and interests, preconditions and guarantees, a list of steps to the success scenario, a list of properly numbered extensions, and a failure-handling remedy for each extension as appropriate.
     c. One casual (paragraph form) use case for one other scenario that you think is important to your product. This use case should be similar to Use Case 4 from Cockburn's paper, describing the main success scenario first in paragraph form, then listing each extension and its remedy in a second paragraph.

2. Software Design Specification (SDS)

The second milestone you will submit for this project is a set of documents about your project's design. Your design should specify information about how to implement an object-oriented software product that meets the requirements laid out in your SRS. Your SDS should answer the following three questions: what are your classes, what are the responsibilities of each class, and how do these classes collaborate?

Specifically, your SDS must contain the following documents:

1. **UML class diagram**
   - Submit a UML class diagram in the proper format as explained in Fowler, Chapter 3. Your class diagram should display all major classes in the system, each class's attributes (fields), methods (do not list get/set methods), inheritance and interface relationships between classes, and named directed associational relationships with multiplicity adornments between classes that collaborate. Your design will be evaluated on completeness as well as level of thought, and attention to principles discussed in class such as cohesion, coupling, and object-oriented design heuristics.

2. **UML sequence diagram**
   - Submit a UML sequence diagram that depicts your product executing one of the use cases you wrote in your SRS. The sequence diagram should follow the format of the examples from Fowler, Chapter 4. Your diagram should show all participants (objects) in the sequence, all important directed messages between them and their return values (if any), as well as interaction frames with proper operator adornments as appropriate.

   Accompanying the sequence diagram should be a pseudo-code description of the same algorithm, similar to Fowler's Figure 4.4.
3. Implementation - Milestone:
The third milestone you will submit for this project is the actual code and supporting files for your project's implementation, based on the design laid out in your SDS. Your project may be an Eclipse project or loose Java files for a standard editor such as TextPad. Your project must have its main method in a class named `Main.java`, so that the instructor can run it. You should assume that the grader and instructor are running your program on a computer in one of our school's computer labs. Do not use any Java features or software programs that are not available in this environment. Assume the presence of Java J2SE SDK v1.4.2.

The Eclipse and TextPad editors and Java 2 SDK can be downloaded at the following web sites, respectively:

http://www.eclipse.org/  
http://www.textpad.com/  
http://java.sun.com/j2se/1.4.2/download.html

The first version of your code implementation will be called the "milestone" version. This version must be a working, running program with a graphical user interface that allows the user to play a minimal 2-player game of Scrabble. The following relaxations of the requirements are allowed for the milestone version:

- It does not need to be possible to play more than one game.
- The players' names can be hard-wired into the game, without any UI to set them.
- The game does not need to display players' names, scores, or a list of high scores.
- The game does not need to verify whether a move is legal or not. (Any tiles put on the board can be considered a legal move.)
- You do not need any "extra" feature in your milestone.
- You may have some bugs in your program, as long as the normal functionality is generally testable.

4. Implementation - Final Version:
Some time after submitting your "milestone" implementation, you will submit your final version. The final version will be graded more strictly than the milestone, and will be expected to meet all requirements for the project. The correctness of your code will be graded by running your product and manipulating its graphical user interface. Exceptions should not occur under normal usage. Your code should not produce any console output, such as `System.out.println` statements in your code that were not specified.

The style of your code will be graded on how well you perform tasks such as the following:

- implement this project specification and your design
- use appropriate collections and data structures, and reasonable code and algorithms
- have model-view separation, as seen in the Model-View-Controller (MVC) and Observer patterns
- comment your code:
  - Put a short header with your name and course on each class.
  - Put Javadoc comments on methods and class headers. This includes /** and */ headers, as well as @param, @return, and @throws tags.
  - Put normal // comments on complex sections of code to help explain them.
- use meaningful identifier names
- appropriately use modifiers such as public, private, protected, and static
- avoid redundancy
- avoid unnecessary fields in your objects, and avoid static global variables whenever possible
- effectively use spacing and indentation in your code

You will submit your project electronically through the course web site's Catalyst e-submit system. Submit all .java source files, any Eclipse project files and settings, any input data or text files needed by your code, images, sound files, and other required code resources that the instructor will need to compile and run your program.
Weekly Reports:
During the implementation phase of this project, each group must submit a "weekly report" file to demonstrate the group's coding progress for that week. The weekly report consists of a .ZIP file named weekly_report_x.zip (where x is the week number, starting from 1 for the first weekly report), containing the following:

- all .java source code files for your project
- a file README.txt that briefly (one or two paragraphs) explains what code was written this week

Weekly reports will be graded on the following criteria:

- Was the report submitted correctly, on time, with the proper file name and contents?
- Does the README.txt accurately describe the progress made this week?
- Was a non-trivial amount of coding progress made this week?

5. Test Cases:
The fifth milestone you will submit for this project is a group of JUnit test cases that perform testing and validation on your project's implementation. Submit test cases that extend junit.framework.TestCase and cover two nontrivial, non-GUI classes in your project. Also submit a test suite file using junit.framework.TestSuite that runs your two tests. The test cases should verify that the major preconditions and postconditions hold on the methods of their respective classes. JUnit can be downloaded from http://www.junit.org/ and can also be run directly from within Eclipse.

Your unit tests will be graded on whether your group performs tasks such as the following:

- submit 2 .java test case files and 1 .java test suite file
- each unit test examines a distinct part of the system
- each unit test has a significant number of testing methods
- each testing method performs a non-trivial test on its class
- each test case file uses useful testing methodologies such as boundary testing, equivalence testing
- each test case's code has proper style (comments, coding conventions, etc)

To understand more about testing and unit tests, consult Breugge and Dutoit Ch. 9. To understand more about preconditions and postconditions, consult Horstmann Ch. 3 and the following URL: http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html

Project Demonstrations:
All projects will be demonstrated in class during one of the final lectures. During this demonstration, your group will show off the major features of your product to your classmates and briefly answer questions from the audience. It may help you to show part of your design during this demonstration. You will be expected to bring a digital UML class diagram to your presentation that shows your final set of Java classes. Demonstrations will be graded on attendance and effort. All group members must be present for the demonstration unless prior arrangements have been made with the instructor at least 48 hours in advance.
Grading:
The points for each part of this project are weighted as follows:

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Due Date (Section A, B)</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weekly reports</td>
<td>each week of implementation</td>
<td>5%</td>
</tr>
<tr>
<td>Meetings with customer</td>
<td>once per phase (2 total)</td>
<td>5%</td>
</tr>
<tr>
<td>Project demos</td>
<td>Wed 6/1/2005, in class Thu 6/2/2005, in class</td>
<td>5%</td>
</tr>
</tbody>
</table>

Participation:
Each group member is expected to contribute a significant amount to the analysis, design, implementation, and testing of your project. This includes willingness to do all of the following:

- meet at least once weekly with your group at a scheduled time
- meet with the instructor together with your group at least once during each milestone
- communicate regularly with your group partners as needed by email, in person, by phone, or otherwise
- hold your group partners accountable for their work, and talk to them and/or report to the instructor if they fail to do it

Your group's project will receive a percentage grade as specified previously. By default, this grade will be shared among all group members. However, if particular group members contribute especially more or especially less than their partners, they may be subject to an individual multiplier between 0 and 2. Missing meetings, failing to communicate, not doing one's share of work, or other negligence may result in a multiplier less than 1.0. For example, if the group's overall grade is 80% but a particular member is deemed to have only done half the amount of work they should have done, they might receive 80% * 0.5 = 40%. This will only be done in extreme circumstances where the group members and instructor agree clearly that there was a lack of work performed.

If one of your partners is not doing his/her share of the work, it is your responsibility to make a reasonable attempt to convince the group member to contribute. If this is not successful, please notify the instructor promptly. The instructor can help ailing groups to agree on a work schedule, and if some group members do not do their share, the instructor can set up different multipliers for project scores. If the instructor is not notified of a group problem until the last minute, there is not a good chance that a group problem can be resolved, nor different grades assigned.