6.170 Laboratory in Software Engineering
Fall 2005
Problem Set 6: Boggle
Due: **Thursday**, November 3rd, 2005, at 1:00pm

Purpose

The purpose of this assignment is to introduce you to the Java Swing windowing toolkit by having you develop a graphical user interface (GUI, pronounced *gooey*) for the board game Boggle.

Background

Boggle is a game where you have a 4x4 grid of letters from which you try to form English words by connecting adjacent letters. For example, suppose the grid were as follows:

N	**S**	**P**	**U**
E	**O**	**T**	**R**
W	**F**	**I**	**E**
S	**P**	**A**	**P**

You could form the word **NEWS** by connecting the letters in the left-hand-column of the grid. You could also form **PUT** by using the letters in the upper-right-hand corner (diagonal squares are considered adjacent in Boggle). However, you cannot legally form **TOT** because you can only use the same square once in a word. This does not prohibit you from using a letter twice if the letter actually appears more than once in the grid – for example, the word **PAPER** is legal in this configuration because **P** appears multiple times. Basically, a word is legal if it is a sequence of letters that can be traced as a path in the grid, where edges in the path connect adjacent squares, and no node in the path is used twice.

In the physical Boggle game, these letters appear on cubes that are scrambled by shaking them inside of a plastic box. When the round begins, the cover is lifted from the box, revealing the grid. For three minutes, players furiously try to find as many words as possible in the grid. At the end of three minutes, players stop writing and list the words that they found out loud. If a player has the same word on her list as another player, then both players cross it off their list and receive no points for it. Once each player has determined which words that he found that no other player has found, he gives himself points for each of these unique words according to the following table (note that words must be at least three letters long to receive any points for them):

3 letter word	1 point
4 letter word	1 point
5 letter word	2 points
6 letter word	3 points
7 letter word	5 points
8+ letter word	11 points

Whichever player has the most points at the end of the round is declared the winner.

Exercises

1. This problem set comes with a considerable amount of sample code. Start out by looking at the interface `ps6.model.Board` as well as the class `ps6.model.BoggleBoard` that implements it. Once you believe that you have an understanding of what this code does, do the following:
    a. Run `ps6.model.BoggleBoardTest` to verify that `BoggleBoard` passes its unit test.
    b. Fill in the comment at the top of `BoggleBoard` where it says **`RI:`** with an appropriate rep invariant for the class.
    c. Implement your rep invariant inside the **`checkRep()`** method.
    d. Now run `ps6.model.BoggleBoardTest` again to verify that `BoggleBoard` still works – does `BoggleBoard` still pass its unit test? If so, say so. If not, investigate what change you made to `checkRep()` that is causing `BoggleBoard` to fail its test. Is it a bug in your `checkRep()` method or has `checkRep()` revealed an actual bug in `BoggleBoard`? Explain.
2. Give an example of how each of the following design patterns is used in this problem set, either in your code or in the provided code. For each example, explain the rationale behind the use of the pattern and the advantages that the use of the design pattern provides.
    a. Model-View-Controller (Structural)
    b. Observer (Behavioral)

Design Problem

Your job is to build a GUI that allows a single person to practice Boggle. (It is difficult to play a multiplayer version of Boggle on a computer without playing over a network because it would be awkward for two players to share a keyboard at the same computer.)

When a user starts your application, he should be able to:

1. Create a new, randomized Boggle board.
2. Start a timer that gives the user 3 minutes to enter words.
3. Input words using TWO different methods:

   First, using the mouse:

   1. The user should press and hold on the first letter of the word, and then drag the mouse over every subsequent letter of the word while keeping the mouse button pressed.
   2. When the user releases the mouse button, the word should be checked to ensure that it forms a valid letter sequence on the Boggle board (e.g. doesn't skip squares), represents a word in the English dictionary we provide, and has length of at least 3.
   3. If the checks pass, the word should be added to the list of found words and the score updated.
   4. When the mouse is first pressed on the first letter of a word, or while it is dragged over new letters, those letters should become highlighted in some way.
   5. For usability purposes, the area of each boggle letter panel which registers a click or drag should be smaller than the letter's entire panel. This will allow users to drag diagonally without inadvertently selecting non-diagonal letters.
   6. Note that the requirements do not specify any obvious way to allow the user to cancel while in the middle of mouse input. You may choose to design and implement such a feature if desired.

   Second, the user should be able to enter the words into a text box using a keyboard:

   7. As each letter is entered, the adjacent boggle board panel should highlight the letter which was typed. (If there are duplicates of the same letter in the boggle board, it is fine to just highlight all instances of that letter. It may be interesting to come up with a mechanism that tries to highlight exactly what the user intended; however, this is not required.)
   8. When the user presses the ENTER keyboard button, the word should be checked to ensure that it forms a valid letter sequence on the Boggle board, represents a word in the English dictionary we provide, and has length of at least 3. If the checks pass, the word should be added to the list of found words and the score updated.
4. Get the total score for the user's words once 3 minutes have expired.

At a minimum, your GUI must display:

1. The grid of letters.

2. How much time the user has left.
3. The words the user has entered thus far.
4. An error message if the user tries to enter a word that does not appear in the grid, is less than three letters long, or is not a legitimate English word. (We will provide a plaintext dictionary of words that can be compared against.)
5. As described above, each letter of the user's word must also be highlighted either as it is selected with the mouse, or after it is entered with the keyboard.

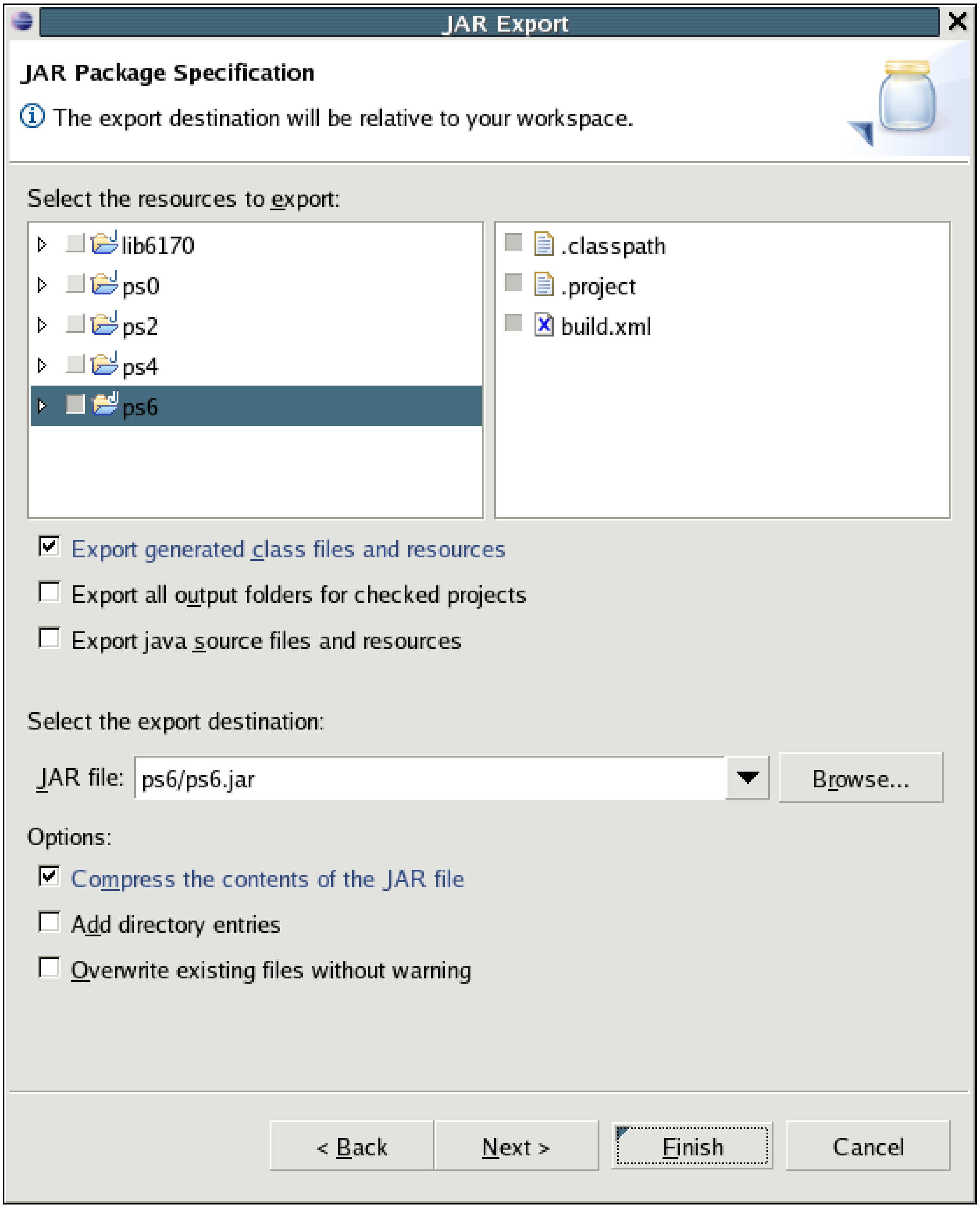
Your GUI should be *easy enough to use* that anyone who already knows how to play Boggle will not need an instruction manual to use it.

Finally, you must include a write-up about your application. Talk about any problems you encountered, any design patterns that you used, and any interesting features that your GUI provides. Include this as `doc/write-up.txt`.

This outlines the basic requirements for the assignment. Feel free to add any more functionality or features as you see fit.

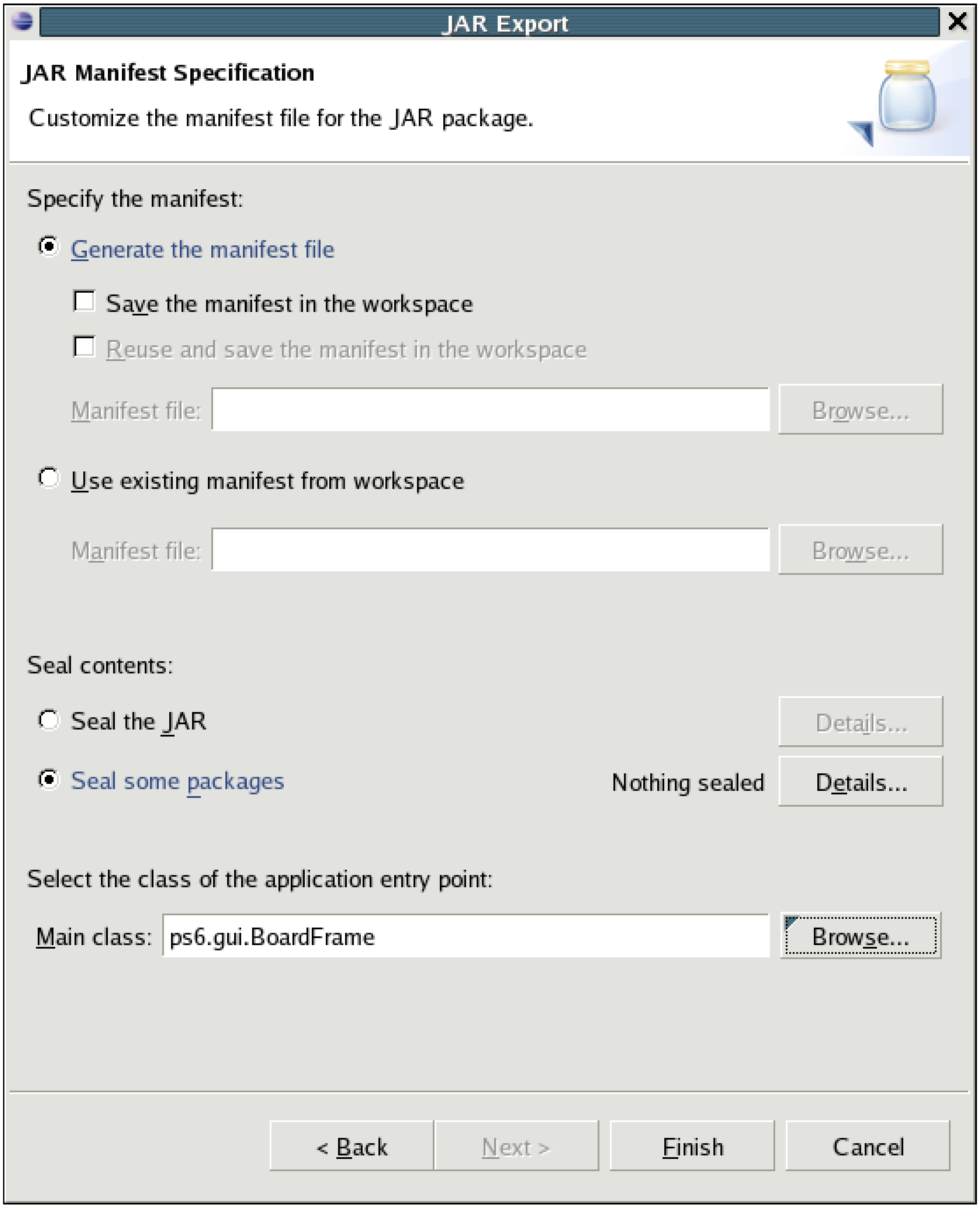
When you are completely finished, please package the class files of your entire problem set into a **.JAR** file. A JAR file allows a developer to bundle compiled code into a single file which can be imported into other projects or even, as in the case for this problem set, be executed directly from the command line. Eclipse makes this easy by providing a tool for exporting your project to a JAR.

1. Click **File->Export** from the menu and select **JAR file**.
2. Select **ps6** as the resource to export, and enter the destination of the JAR. Note that this is relative to your workspace directory in Eclipse (which is most likely the parent directory of the ps6 project directory).



Courtesy of The Eclipse Foundation.

3. The next screen has further options that you can customize. The defaults suffice for our purposes.
4. The manifest specification screen specifies the "manifest file," which indicates the class whose `main()` method should be run should the JAR be executed. Eclipse can automatically generate this manifest file, provided that you specify the location of your **Main class**. If you use the GUI framework we provide, this will likely be `ps6.gui.BoardFrame`.



Courtesy of The Eclipse Foundation.

5. Once you're done, click **Finish**, and check that the JAR file is successfully created.
6. JAR files can be run from the command line. To do so, simply call `java -jar ps6.jar` from the directory containing the jar. Be sure to add your JAR to CVS.
7. (Side note: JAR archives are actually packaged in the same way as the popular .zip format. If you're curious, you can find out what's actually in your JAR by calling `unzip ps6.jar` from the command line.)

Resources

Swing

If you have never done any GUI programming and are not familiar with Java graphics tools, don't panic! A quick survey of Swing is available on the MIT server. The Swing Lab tutorial, written by the 6.170 staff, explains how to create graphical windows using various layout managers, how to paint into a window, and how to handle keyboard and mouse input. --!> If you would like to learn more about Swing before getting started, Sun also provides its own Swing Tutorial.

Also, if you are interested in creating a layout that is more detailed than traditional layout managers such as `BorderLayout` or `FlowLayout`, and you find yourself perplexed by the specification `GridBagLayout`, fear not, for you are not alone! Fortunately, there is another layout manager called TableLayout that facilitates complex layouts. We have already included the TableLayout jar in your `lib/` directory, but you will still have to visit its web site if you want to look at its API (unfortunately, TableLayout is not part of Sun's SDK).

For this problem set, you **MUST** use Java Swing to build your GUI. You will be able to use other GUI libraries (such as SWT) in your final project.

Threads

Because we require you to update a clock timer in this problem set, you may think that you need to subclass `java.lang.Thread` to create a thread to update the clock periodically – this is not the case. Instead, look at javax.swing.Timer, as it provides a convenient abstraction for GUI threads and simplifies thread management.

Handling the Dictionary file

To assure that all the words entered by the user are English words, we have included the file `input/dictionary.txt` that has one word on each line. Your Boggle game should load this file upon startup and parse each word into your own internal data structure. You do not need to distinguish between uppercase and lower case letters.

There are many efficient ways to store these words so that they can easily be accessed for comparison, including Sets and Maps. One interesting data structure is a Trie. A Trie is a just like a tree, except it stores data at each node within the tree itself, not only the leafs. Values can be retrieved from the tree by starting at the root node, and walking down the tree, recording the values at each node until you reach a leaf. It is not necessary to use a Trie for this assignment, but more information about this and other interesting string-matching data structures can be found in the lecture notes for Advanced Data Structures from Spring 2005 (6.897).

Provided Code

As you will notice, we have already built a Boggle ADT for you that uses the polymorphic multi-graph. To use this ADT in your code, all you need to write is:

```
Board board = new BoggleBoard();
```

and *voila*, you have a board that:

- can be shuffled by invoking `shuffle()`
- can be read by invoking `getLettersAt()`
- can be queried by invoking `containsWord()`

Thus, most of the functionality for Boggle is already written – you just need to wrap a GUI around it!

Mechanics

Create a class in the `ps6` package called `Boggle` with a `main` method that a client can invoke to start your GUI. If your GUI is contained in a `javax.swing.JFrame` called `BoggleFrame`, then your `main` method may look something like this:

```
public static void main(String[] args) {
  JFrame frame = new BoggleFrame();
  frame.setVisible(true);
}
```

The other packages in your `src/` directory are:

- `ps6.graph` - this is an implementation of the proposed design for a multi-graph
- `ps6.model` - this builds on the previous package to create an ADT for Boggle. Though you should be able to use `ps6.model.Board` without looking at is implementation (the beauty of abstraction!), the source is provided for those of you who are curious about how it all fits together.

Expectations

Deliverables

- Source code for your GUI
- Writeup in `doc/write-up.txt`
- Executable JAR `ps6.jar` in your ps6 project's root directory (i.e. the parent of the `src` and `input` directories). When adding to CVS, be sure to set this as a binary file.

What you need to know before you start:

You should be familiar with the Swing GUI library.

What you should expect to learn:

You will learn how to create a GUI in Java.

How you will be evaluated:

- 25 points for the exercises
- 65 points for your implementation
- 10 points for your write-up in `doc/write-up.txt`

Note that this problem set awards more points for implementation than any previous problem set. This is because the Swing library is large, and takes awhile to learn if you are using it for the first time. We want you to focus on learning Swing in this problem set, which is why we provide you with a complete backend for Boggle and require little in the way of a write-up. You will notice that it is awkward to write abstraction functions, rep invariants, and test cases for GUI classes (though it is certainly possible: see java.awt.Robot); thus, we do not require them for this problem set. To reiterate, we are awarding more than half of the points for this problem set for your implementation, both because we want you to focus on learning Swing and because we expect you to spend a considerable amount of time doing so – please budget your time accordingly!

Errata

Point Distribution Change (Oct 30, 2005):

The distribution of points has been changed slightly. Previously, the distribution was 30/60/10 for Exercises/Implementation/Write-up. This has been changed to 25/65/10 for Exercises/Implementation/Write-up.