CS 51 Laboratory # 11
Apples

Objective: To practice using String methods.

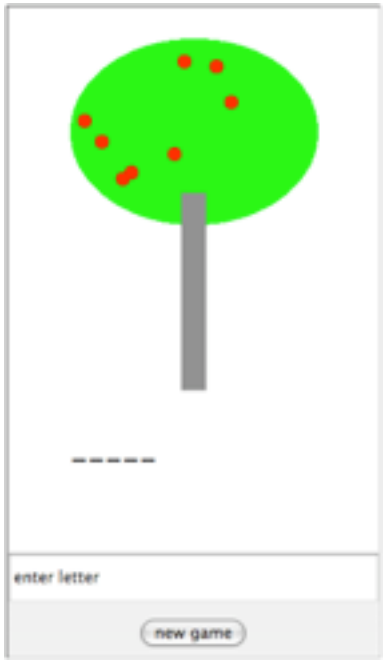
In this lab we'll be implementing a version of the game Hangman. **You are encouraged to work in pairs in this lab.** If you decide to work with a partner, the two of you must do *all* of the design and coding together. A pair should turn in only one copy of the program, but make sure both people's names are on the folder and in the comments.

Hangman is a simple game in which one person, the "*drawer*", thinks of a word and another person, the "*guesser*", tries to guess it. The game starts with the *drawer* writing down a row of dashes, with each dash representing one letter in the word. The *guesser* then repeatedly guesses a letter. If the letter appears in the word, the *drawer* replaces each appropriate dash with that letter. If the letter doesn't appear in the word, the *drawer* makes a note of that. The *guesser* is allowed some number of incorrect guesses before they lose the game. In our case they'll be allowed 8 incorrect guesses.

The name Hangman comes from the fact that instead of simply noting that the *guesser* has lost because they've guessed 8 letters that don't appear in the word, traditionally the *drawer* has drawn a feature of a hanging person for each incorrect letter, with the *guesser* losing when the complete figure of a hanging person has been drawn.

In our case we'll be playing a version of the above game called Apples. In this version, the *drawer* (your program) draws a tree with 8 apples, and each incorrect guess by the *guesser* causes one apple to fall off the tree. When all the apples are gone, the *guesser* has lost.

Below is our version of the game:



The online version of this lab handout includes a demo version of this program. You can play with it to see what we have in mind.

Notice that guesses in either uppercase or lowercase are acceptable. Furthermore, nothing happens if you guess a letter that you've already guessed before.

Overview

Your program will consist of four main classes:

Dictionary will be the class that loads in a file of words and that is responsible for returning a random word.

AppleGameController will be the main class which extends `WindowController` and sets up the game interface.

AppleGame will be the class that maintains the state of the game.

AppleGameDisplay will be the class that maintains the image of the apple tree.

We will provide complete implementations of the `Dictionary` class and the `AppleGameDisplay` classes, which are described in the following section. You'll need to write the other two, following the specifications below.

The `Dictionary` class The only thing you need to know about the `Dictionary` class is that the constructor has the following signature:

```
public Dictionary(String filename)
```

and that it has one important method:

```
public String getRandomWord()
```

The file passed to the constructor should contain one word per line. We've included such a file and an appropriate constant in the `AppleGameController` class. The method gives you a random word from the dictionary.

The `AppleGameDisplay` class This class deals with drawing the tree with the apples and provides methods that let you remove an apple and that let you refill the tree. The signatures for the constructor and public methods are:

```
public AppleGameDisplay(int numApples, DrawingCanvas canvas)
public void fillTree()
public int getNumApplesLeft()
public void removeApple()
```

You can read the comments in the provided code to see more what these methods do, but most should be intiutive from their names.

The `AppleGame` class The `AppleGame` class will manage the state of the game. This involves three basic things:

- You need to keep track of the state of the game. You may **NOT** use arrays to keep track of things. Instead, you should use `String`s and appropriate operatios to keep track of the state. Don't forget that `String`s are immutable!

  There are three basic things you'll need to keep track of: the word the person is trying to guess, the current state of the dashed word (i.e. the *guesser's* guessing) and what letters have been guessed so far.

- Generate and update state of the tree using an `AppleGameDisplay` object.

- Generate and update the textual display of the game.

The constructor for `AppleGame` should have the following signature:

```
public AppleGame(int numApples, String wordfile, DrawingCanvas canvas)
```

and should have the following methods:

```
public void guessLetter(char c)
```

which takes a character `c` and, if the character hasn't already been tried, determines whether or not it appears in the word, and updates both the text and the picture accordingly (remember our guesses are case insensitive).

```
public void newGame()
```

which resets both the text and tree display, and which chooses a new word from the dictionary. The constructor for `AppleGame` should have the following signature:

```
public AppleGame(int numApples, String wordfile, DrawingCanvas canvas)
```

The `AppleGameController` class The `AppleGameController` class manages the overall game. The constructor should setup the GUI so that there's a `JButton` labeled "New Game" and a `JTextField` for entering guesses.

The user interacts with the game by entering a letter into the field and pressing enter. If the letter string entered is more than one character or is not a letter then the program should not respond, and should again prompt the user to enter a letter.

Design

This week we will again require that you prepare a written "design" for your program before lab. You only need to provide a design for the `AppleGame` and `AppleGameController` classes since we provide the code for the other two classes. At the beginning of the lab, we'll come around and briefly examine each of your designs to make sure you are on the right track. At the same time, we'll assign a grade to your design.

Implementation

As usual, we suggest a staged approach to the implementation of this program. This allows you to identify and deal with logical errors quickly. The size of your applet should be 300 pixels wide by 525 pixels tall.

- Start by getting the initial GUI setup so that the textfield and the button are in the right place.
- Now write the constructor and the `newGame` method for the `AppleGame` class, ignoring the tree display. Make sure you can choose a new random word from the dictionary, and that you can display the right number of dashes. Add in the listener in the `AppleGameController` class and verify that the `New Game` button causes a new word to be chosen.
- Next write the `guessLetter` method for the `AppleGame` class. Again, ignore the tree display, and just make sure that your code correctly determines whether the letter appears in the word and that it updates the text display appropriately. Add in the listener for the textfield and verify that the GUI works correctly.
- Finally, add in the code that uses the `AppleGameDisplay` class.

The challenging string maniupulation question for this lab is how to update dashed word when a new letter is entered. There are many ways to do this, one reasonable way is to iterate through the characters in the word to be guessed, and if the character occurs in it, then update the dashed word. To do this, you can build up a new, temporary, dashed word as you go along, then when you get to the end replace the original dashed word with the temporary one. Think about this problem before beginning coding the `guessLetter` method!

Submitting Your Work

The lab is due at 11 PM on Monday as usual. When your work is complete you should deposit in the dropoff folder a folder that contains your program and all of the usual files needed by Eclipse. Make sure the folder has your name and lab number on it (if you worked with a partner, include both names). Also make sure that your name(s) are included in the comment at the top of each Java source file.

Before turning in your work, be sure to double check both its logical organization and your style of presentation. Make your code as clear as possible and include appropriate comments describing major sections of code and declarations.

Sketch of `Dictionary` class

```
// Initializes the dictionary with the words in filename; assumes
// one word per line.
// only loads in first MAX_WORDS if filename is larger than that
public Dictionary(String filename)

// returns a randomly chosen word from the dictionary
public String getRandomWord()
```

Sketch of `AppleGameDisplay` class

```
  // constructor takes the number of apples and the canvas
  // draws the initial tree with numApples apples, randomly
  // placed.
  public AppleGameDisplay(int numApples, DrawingCanvas canvas)

  // resets the tree to show all apples
  public void fillTree()

  // returns the number of apples left on the tree
  public int getNumApplesLeft()

  // removes an apple from the tree
  public void removeApple()

}
```

Table 1: Grading Guidelines

Value	Feature
	Design preparation (4 points total)
2 points	`AppleGameController` class
2 points	`AppleGame` class
	Syntax style (5 points total)
2 points	Descriptive comments
1 points	Good names
1 points	Good use of constants
1 point	Appropriate formatting
	Semantic style (4 points total)
1 point	Conditionals and loops
1 point	General correctness/design/efficiency issues
1 point	Parameters, variables, and scoping
1 point	Miscellaneous
	Correctness (7 points total)
1 point	GUI works
1 point	game stops and restarts correctly
3 points	handles letters input correctly
1 point	text display correct
1 point	tree display correct
	Extra Credit (2 points maximum)
.5 point	Better text/tree graphics
.5 point	Better interface