CSC1016S Assignment 10: Graphical User Interfaces

Assignment Instructions

This assignment concerns constructing programs in Java that have graphical user interfaces.

The specific challenge is to construct a hangman game as a means of developing and demonstrating an understanding of graphical user interfaces and event-driven programs i.e. the game is not an end in itself.

We have a large file of English words and a couple of components: a Word class, objects of which unsurprisingly represent words, and a Pattern class, objects of which represents expressions which describe one or more words.

The materials may be found (along with Java docs) on the Vula assignment page in a ZIP file called ‘Words.zip”.

The Word class

Java provides a String type and naturally, words are strings of characters. However, it seems appropriate that we introduce a distinct type since not all strings of characters form words. Another point we can make is that the empty string ("") is a useful concept but that the idea of an empty word does not make much sense. Finally, having a distinct word type serves to extend our programming vocabulary and makes our code easier to understand. For example, take the following variable declaration:

```
Word w;
```

It's pretty clear what "`w`" denotes.

The Word type that we've defined is fairly simple. A Word object represents a sequence of characters that may legitimately represent a word. Determining legitimacy is a tall order. We've opted for it to mean that the sequence must contain typical characters: letters, hyphens, apostrophes, full stops.

A Word object may be created from a suitable String - there's a class method provided that enables us to check whether a String is suitable - and given a Word object, we can compare it with others or obtain its string representation for the purposes of printing or whatever.

The Pattern class

Pattern matching is a process by which we search for words with some given characteristics. The characteristics that we might be interested in include length, character content and character order.

A pattern is a description of required characteristics. A pattern may range from a precise description that matches a single word to a very general description that matches many.

A simple way of representing a pattern is by using a sequence of letters and special "wild card" characters. When a letter appears in a pattern it indicates that the very same letter must occur in the same position in matching words. When a wild card appears in a pattern it indicates that a number of different characters or character sequences are acceptable at that position.

The Pattern class is based on two wild card characters.

CONTINUED

A. A query, '?', indicates that any character may appear at that given position in a matching word.
B. An asterisk, '*' indicates that a sequence of zero or more characters may appear at the given position in a matching word.

Here are some examples

pattern	possible word matches
letter	letter
l?ad	lead, load
l*ad	lad, Launchpad, lead, letterhead, lipread, load, loggerhead, lunkhead, ...
l?*ad	Launchpad, lead, letterhead, lipread, load, loggerhead, lunkhead, ...
*action	abreaction, abstraction, action, attraction, benefaction, coaction, ...

The Pattern type has many similarities to the Word type. Besides simply providing a representation of the requisite sorts of character sequences, Pattern objects support matching against Word objects. Given a `Pattern p` and a `Word w` we can test to see if `w` can be matched to `p`:

```
if (p.matches(w)) {
    // Do something
}
```

The WordList class

The WordList concept is that of a simple dictionary. A dictionary is an ordered list of words and definitions. Ours is simple because it only has words.

A WordList object represents a list of Word objects in alphabetical order. The WordList type provides:

- facilities for determining whether a word is present,
- facilities for searching for words that match a given Pattern, and provides
- facilities (class methods) for loading words from a text file. (We have provided one called “dictionary.txt”.)

A WordList object is ‘Iterable’. So, for example, we can write code such as:

```
WordList wl = WordList.readFromFile("dictionary.txt");
for(Word w : wl) {
      System.out.println(w);
}
```

CONTINUED

Exercise [100 marks]

Write a program called "`Hangman.java`" that implements the hangman game, interacting with the player via a graphical user interface.

Hangman is of course the slightly gruesome game that involves trying to guess a mystery word. The player is allowed a number of guesses dependent on the number of steps it takes to draw gallows complete with hanged man.

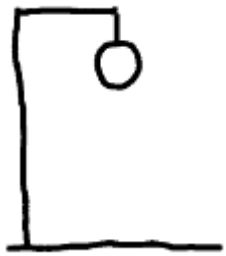
When the game begins, all that is known of the mystery word is the number of letters. A guess involves suggesting a letter. When a guess is correct all occurrences of the suggested letter are revealed e.g.

```
---
Guess 'a'
-a-
Guess 't'
-at
Guess 'h'
wrong
Guess 'c'
cat
```

A successful guess incurs no penalty. An unsuccessful guess results in another stroke being added to the hanged man drawing e.g.

```
-at
```

Guessing 'h':



```
-at
```

Program requirements

A. The program will implement a single player game of Hangman.
B. The player must guess a mystery word by suggesting letters that may appear within it.
C. The program will chose the mystery word from a store of words.
D. The player may select a difficulty level.
E. When the game begins, all that is known of the mystery word is the number of letters. Initially the mystery word will be depicted as a sequence of hyphens or underscores. Each hyphen or underscore corresponds to a letter in the word. e.g. Say the mystery word is "illustrator", it will initially be depicted as "----------".
F. Correctly guessing one of the letters of the word causes the position of all its occurrences to be revealed e.g Say the mystery word is "illustrator", and the first guess made by the player is 't', it will be depicted as "-----t--t--".
G. The player is permitted a limited number of incorrect suggestions. The number may be dependent on the level of difficulty of the game.
H. If the player identifies all the letters in the mystery word before they use up their stock of incorrect guesses then they win the game.
I. If the player uses up their stock of incorrect guesses before identifying all the letters in the mystery word then they lose the game.
J. The number of incorrect guesses made by the player will be depicted by a (partial) drawing of a hanged man. A completed drawing indicates that the player's stock of incorrect guesses has been used up.
K. The hanged man drawing will be depicted using a combination of graphical strokes. The completed hanged man drawing will be depicted as follows:

Design

The design is largely up to you. We recommend you make use of the Word and WordList classes from exercise one. You may wish to use our drawings (in the ZIP file called 'hangmandrawings.ZIP' on the Vula page).

To display a graphical file (such as those we've provided), you can use an ImageIcon object (`javax.swing.ImageIcon`).

For example, assuming the GIF file '`state11.GIF`', the following code creates a JLabel that has an icon that is the picture in the file:

```
ImageIcon icon = new ImageIcon('state11.GIF');
JLabel jLabel = new jLabel(icon, SwingConstants.CENTER);
```

(See appendix for details of javax.swing.SwingConstants.)

CONTINUED

Marking and Submission

Submit your solutions to questions one and two within a single .ZIP folder to the automatic marker. The zipped folder should have the following naming convention:

yourstudentnumber.zip

NOTE: This assignment is not automatically marked. The tutors will manually assess your work.

CONTINUED

Appendix: Useful Swing Components

We'll briefly describe some of the key features of graphical components that you will need to use. These are all part of the `javax.swing` package.

You should consult the Java API documentation if you wish to see a complete description of features.

JTextField

A user can enter data via a JTextField. A JTextField can be of specified size.

Constructor

JTextField(int columns)

// Create a JTextField of the specifiedwidth.

Methods

String getText()

// Obtain the text currently in the field.

JLabel

A JLabel simply serves to display some text.

Constructor

JLabel(String text)

// Create a JLabel that displays the given text.

JScrollPane

A JScrollPane is a very useful component. A component placed within a JScrollPane may be as large as we wish. When the available display space is smaller than the component the JScrollPane provides the necessary scroll bar facilities.

Constructor

JScrollPane(Component C)

// Create a JScrollPane that manages the view of the given component c.

CONTINUED

JList

A JList simply serves to display a list of items. When used in conjunction with a JScrollPane we can specify how many items should be visible at the one time.

There are a number of ways by which we can specify the items to be stored in the JList. (For question one, we recommend an array of Objects. This means translating a WordList (the result of searching for matches with a given pattern) by obtaining its size, creating an array of suitable size and copying from one to the other.)

Constructor

JList()
    // Create a new JList.

Methods

void setListData(Object[] listData)
    // Fill the JList with the given array of objects.

Box

A Box is a general purpose container for organising graphical components. Contents may be arranged horizontally or vertically.

To facilitate arrangement, contents can be spaced using 'struts' and 'glue'. A strut is an invisible component that takes up a specified amount of space. Glue is used to bind or attract components to one another. Another way of thinking of glue is as a rubber band stretched between components. For example, given a horizontal Box that contains a single JLabel with glue on either side, no matter how large the box, the tensions caused by the glue will always force the label to appear in the centre.

Horizontal and Vertical Boxes, struts and glue are easily created using a set of class methods.

Class methods

static Box createHorizontalBox()
    // Create a Box that will arrange contents horizontally, left to right in the order in which they were added.

static Box createHorizontalGlue()
    // Create a blob of glue for use in a horizontal Box.

static Box createHorizontalStrut(int width)
    // Create a horizontal strut of the specified width.

static Box createVerticalBox()
    // Create a Box that will arrange contents vertically, top to bottom in the order in which they were added.

static Box createVerticalGlue()
    // Create a blob of glue for use in a vertical Box.

static Box createVerticalStrut(int height)
    // Create a vertical strut of the specified height.

CONTINUED

SwingConstants

A collection of constants generally used for positioning and orienting components on the screen.

Constants

static int CENTER
    // The central position in an area.

static int EAST
    // Compass-direction East (right).

static int NORTH
    // Compass-direction North (up).

static int NORTH_EAST
    // Compass-direction North-East (upper right).

static int NORTH_WEST
    // Compass-direction North-West (upper left).

static int SOUTH
    // Compass-direction South (down).

static int SOUTH_EAST
    // Compass-direction South-East (lower right).

static int SOUTH_WEST
    // Compass-direction South-West (lower left).

static int WEST
    // Compass-direction West (left).

END

CONTINUED