Version: 18/10/2021 19:39

CSC1016S Assignment 8

Interfaces and Graphical User Interfaces

Assignment Instructions

This assignment concerns (i) the use interface declarations to construct abstract data types and the use of class declarations to construct concrete (implementing) sub types; (ii) constructing programs in Java that have graphical user interfaces.

NOTE: Question 3 will be manually marked by tutors. Please do comment your code accordingly.

Question 1 [25 marks]

See the attached `Question1.java` source file. Its main method prompts the user to enter a list of animals and then prints out a list of corresponding animal noises once they are finished.

Write the code necessary to make `Question1.java` compile and produce the output below when run. You will need to create a MakesSound interface with a public method `makeNoise()` as well as Cat, Dog and Cow classes that implement the MakesSound interface.

Sample I/O

```
What animal do you see? (1) Cat (2) Dog (3) Cow (4) quit
1
What animal do you see? (1) Cat (2) Dog (3) Cow (4) quit
1
What animal do you see? (1) Cat (2) Dog (3) Cow (4) quit
1
What animal do you see? (1) Cat (2) Dog (3) Cow (4) quit
2
What animal do you see? (1) Cat (2) Dog (3) Cow (4) quit
1
What animal do you see? (1) Cat (2) Dog (3) Cow (4) quit
1
What animal do you see? (1) Cat (2) Dog (3) Cow (4) quit
3
What animal do you see? (1) Cat (2) Dog (3) Cow (4) quit
4
The animals want to say goodbye:
Meeow
Meeow
Meeow
Woof!
Meeow
```

```
Meeow
Moo!
```

Question 2 [25 marks]

Write code that will accept as input the characteristics of an undefined number of different softdrinks. Each softdrink should have a colour (of the can), name and volume.

- Create SoftDrink objects and store them in an ArrayList (see the Appendix) as each softdrink is entered.
- Alternatively, you can use an array. Take a number of items first, create an array of that size and read in the elements.
- Prompt the user to add a softdrink until they choose to quit.
- Once the user quits, print out a list of the softdrinks that the user has entered, sorted first by alphabetical order of name, then by colour, then by volume (ascending order).

Name your driver class `Question2.java`.
NOTE:

- You can use the built-in static `sort()` method in the `java.util.Collections` class to sort your ArrayList of SoftDrink objects.
- Click HERE for information on how to make sure your SoftDrink class will work with `Collections.sort()`.

Sample I/O

```
Enter option: (1) add soft drink (2) quit:
1
Enter name, colour and volume in ml separated by space
Fanta Orange 500
Enter option: (1) add soft drink (2) quit:
1
Enter name, colour and volume in ml separated by space
Fanta Orange 200
Enter option: (1) add soft drink (2) quit:
1
Enter name, colour and volume in ml separated by space
Coke Red 500
Enter option: (1) add soft drink (2) quit:
1
Enter name, colour and volume in ml separated by space
Coke Silver 500
Enter option: (1) add soft drink (2) quit:
2
Coke Red 500
Coke Silver 500
Fanta Orange 300
Fanta Orange 500
```

Question 3 [50 marks] – to be manually marked

The Scenario

The challenge on this question involves words. 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.

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:

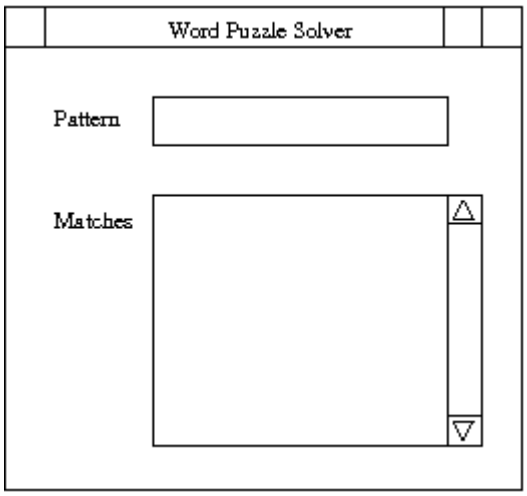
- facilities for determining whether a word is present,
- facilities for searching for words that match a given Pattern, and provides
- facilities (class methods) for loading words from a text file. (We have provided one called “dictionary.txt”.)

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

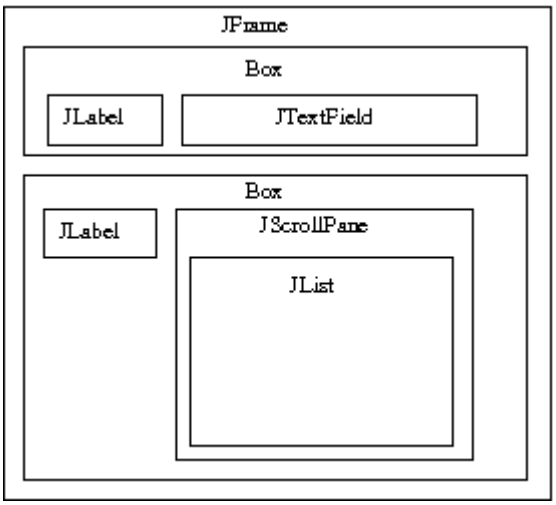
```
WordList wl = WordList.readFromFile("dictionary.txt");
for(Word w : wl) {
    System.out.println(w);
}
```

The WordSearch class

The WordSearch class implements the actual graphical user interface. This is the interface that may be used to search a dictionary of words that match a given pattern. The interaction with the user is via a graphical user interface with the following features:



At the top of the window will be a text input box. The user will enter a pattern here and then press return. Any matching words are displayed in the bottom box.
The GUI is constructed from a JFrame, a JTextField, a JList, a JScrollPane, JLabels and Boxes, and an ActionListener.
The following figure depicts the construction.



The raw components are JLabels for the labels 'pattern' and 'matches'. The user enters text in a JTextField. Output is displayed in a JList. This JList is put in a JScrollPane so that we can easily view all possible matches. The rest of the components are Boxes. These are used for structuring the interface.
The 'pattern' label and JTextField are arranged horizontally within a Box. This box is placed at the top of the main JFrame window.
The 'matches' label and the JScrollPane containing the JList are also arranged horizontally in a Box. This box is placed at the bottom of the main JFrame window.

Task

Write a program called "`PatternMatcher.java`" that may be used to search a dictionary of words for those that match a given pattern.

The program will interact with the user via a graphical user interface that has been supplied to you in the `WordSearch` class.

Your task is to implement an ActionListener type class designed to deal with the user input. The main program class in the `WordSearch class` responsible for building the GUI structure, building a WordList from the file of words, and creating an ActionListener.
An ActionListener will be responsible for dealing with the event where the user enters a pattern in the JTextField and presses return. It will be responsible for

1. when the user presses enter, obtaining the text from the field, and if valid, creating a Pattern object,
2. building an array of Object from the WordList containing words that match the Pattern,
3. putting the array of Object in the JList for display.

We recommend that the ActionListener meet the following specification:

class PatternMatcher implements ActionListener

Instance variables

private JTextField userInputField;

Constructor

PatternMatcher(JTextField userInputField, , WordList wordlist, JList searchOutputList)

// Create a PatternMatcher passing it the reference to the JTextField from which user input may
// be obtained, the WordList to be searched using the pattern, and the reference to the JList
// through which search results are to be output.

Methods

void actionPerformed(ActionEvent e)

// Called when the user inputs a pattern. Creates a Pattern object and creates a WordList from
// file, searches the list for matches, putting matches in a vector and ultimately passing the
// result to the JList.

At the end of this document you will find an appendix containing descriptions of the various GUI components mentioned above (JList, JTextField, JLabel, JScrollPane).

Note: Question 3 will be manually marked.

Marking and Submission

Submit `MakeSound.java`, `Cat.java`, `Dog.java`, `Cow.java`, `Question2.java`, `SoftDrink.java`, and `PatternMatcher.java` in a single .ZIP folder to the automatic marker.

The zipped folder should have the following naming convention:
yourstudentnumber.zip

Appendix 1: Java ArrayList Class

Java ArrayList class uses a *dynamic array* for storing elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the java.util package. It is like the Vector in C++.

The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of List interface here. The ArrayList maintains the insertion order internally.

It inherits the AbstractList class and implements List interface. Java ArrayList class can:

- contain duplicate elements
- maintain insertion order
- allow randon acess because array works at the inde basis.
- In ArrayList, manipulation is a little slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the ArrayList.

Java ArrayList class extends AbstractList class which implements List interface. The List interface extends the Collections and Iterable Interfaces in hierarchical order (see declaration below).

ArrayList class declaration

```
public class ArrayList<E> extends AbsractList<E> implements List<E>,
RandomAccess, CLoneable, Serializable
```

Constructors of ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection<? extends E> c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

For information on the methods of ArrayList class click HERE.
Here is a quick note on how to create a generic and non-generic ArrayList:

```
ArrayList list = new ArrayList(); //creating old non-generic arraylist

ArrayList<String> list = new ArrayList<String>(); //creating new generic
                                                  // arraylist
```

In a generic collection, we specify the type in angular braces. Now ArrayList is forced to have the only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

Adding and accessing an object of the ArrayList can be achieved in the following manner:

```
1 import java.util.*;
2 public class ArrayListExample1{
3    public static void main(String args[]){
4       //Creating arraylist
5       ArrayList<String> list = new ArrayList<String>();
6       list.add("Mango");//Adding object in arraylist
7       list.add("Apple");
8       list.add("Banana");
9       list.add("Grapes");
10
11       //Printing the arraylist object
12       System.out.println(list);
13    }
14 }
```

Output:

```
[Mango, Apple, Banana, Grapes]
```

Accessing the elements of an ArrayList can also be achieved using an iterator by adding the following statements, in lines 10 - 12 in the code snippet above:

```
 //Traversing list through Iterator
 Iterator itr = list.iterator(); //getting the Iterator
 while(itr.hasNext()){ //check if iterator has the elements
    System.out.println(itr.next()); //printing the element and move to
                                    //next
 }
```

Output:

```
Mango
Apple
Banana
Grapes
```

An ArrayList can also be traversed using a `for loop` in the following manner:

```
//Traversing list through for-each loop
for(String fruit:list)
   System.out.println(fruit);
```

Output:

```
Mango
Apple
Banana
Grapes
```

Appendix 2: 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.

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.

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).