CSCI 3353 Object Oriented Design

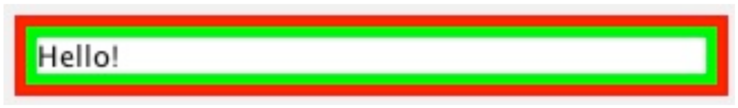
Homework Assignment 5

Due Wednesday, October 19

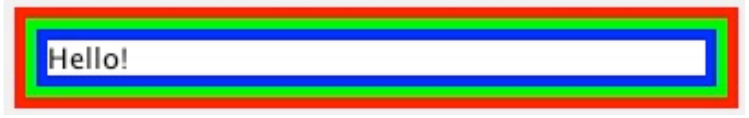
The first two questions of the assignment are concerned with how to add borders and scroll panes to Swing components. You need to examine the Java API documentation for the class *JScrollPane*, the interface *Border*, and the classes *TitledBorder*, *CompoundBorder*, *LineBorder*, and *BevelBorder*. Note that borders are added to Swing components via the method *setBorder* in class *JComponent*.

1. Write a Java program, called *BorderTest*, that displays a panel containing five text fields and two text areas. The text fields should have the following borders:

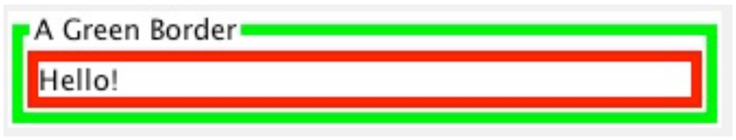
a) A 2-colored border:



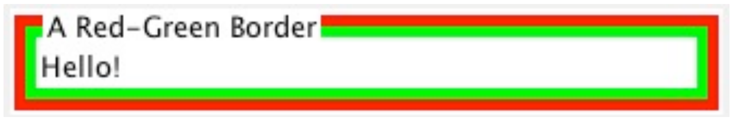
b) A 3-colored border:



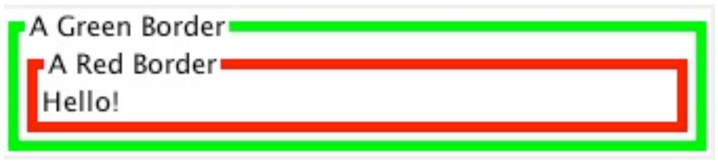
c) A 2-colored border with a title on the outside border:



d) A 2-colored border with a title across both borders:

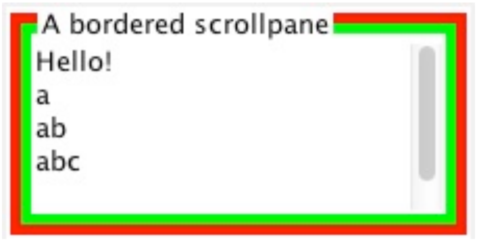


e) A 2-colored border with different titles across each border:

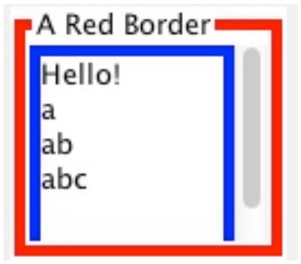


The text areas should be surrounded by a scroll pane, and have the following borders:

f) A two-colored titled border (as in part d), surrounding the scroll bars:



g) A titled red border surrounding the scroll bars and a blue border surrounding the text area but inside the scroll bars:



```
Hello!
a
ab
abc
```

2. Consider the classes *Border*, *TitledBorder*, *CompoundBorder*, *LineBorder*, *BevelBorder*, *JScrollPane*, and *JComponent.*

a) Draw a class diagram depicting the connections between these seven classes and interfaces. Don't be sloppy. Your diagram should convey that you understand exactly how they are related.

b) Explain where the *strategy* design pattern is used in this diagram. Which classes are strategy classes?

c) Explain where the *decorator* design pattern is used. In particular, which classes are the base classes? Which classes are the decorators?

3. This question is concerned with how to create decorator classes for the *Collection* interface.

In class #7, we talked about how to use the abstract class *AbstractCollection* to write your own collection classes, and gave the demo class *HashTableCollection* as an example. The idea is that each decorator collection class will extend *AbstractCollection*, and will implement the abstract methods *size* and *iterator*. (For this assignment, you don't need to implement the *add* and *remove* methods.) Your classes may optionally implement other methods; for example, the point of *HashTableCollection* was that it had a very efficient implementation of the method *contains*.

For another example, download the file *MyStringCollection.java*. This class uses a single (large) string to store a collection of strings.

Recall that the constructor of a decorator class will always have an argument that is the object that it is decorating. The methods of the decorator class will either override or modify the behavior of the corresponding methods of its component.

In particular, I want you to write the following decorator classes:

- *FilteredCollection*: The arguments to its constructor will be a predicate and a component collection. The constructor should create a new collection containing those values from the component collection that satisfy the predicate. Its *iterator* and *size* methods use the corresponding methods of that collection.

- *SortedCollection*: The arguments to its constructor will be a comparator and a component collection. The constructor should create a list containing a copy of the values from the component collection. It then sorts that list based on the comparator. Its *iterator* and *size* methods use the corresponding methods of the sorted list.

- *MergedCollection*: The arguments to its constructor will be two component collections. Unlike the other two classes, this class should not create a new local collection. Instead, its *iterator* method should gets its values "on demand" from the iterators of its components. In effect, the merged collection is the union of the two component collections.

The *FilteredCollection* class uses predicates, so I need to explain how predicates work. A *Predicate* is a built-in interface in the Java 8 library. (It's in the package *java.util.function*.) Its definition looks something like this:

```
interface Predicate<T> {
   public boolean test(T e);
}
```

Classes that implement *Predicate* are strategy classes. The *test* method determines the strategy for determining if the argument value is "satisfactory". For an example, here is the code for a predicate class that tests whether an integer is divisible by a supplied value:

```
class isDivisibleBy implements Predicate<Integer> {
   int val;
   public isDivisibleBy(int val) {
      this.val = val;
   }
   public boolean test(Integer n) {
      return n.intValue() % val == 0;
   }
}
```

The following example code uses the predicate to determine if 86 and 88 are divisible by 4:

```
Predicate<Integer> pred = new isDivisibleBy(4);
boolean b1 = pred.test(86);
boolean b2 = pred.test(88);
```

To implement the *MergedCollection* class, you will need to write an *iterator* method. This method must return an appropriate *Iterator* object. You will find it necessary to write customized *Iterator* classes for this purpose. (Recall that an *Iterator* class needs to implement the methods *next* and *hasNext*.) The demo class *MyStringCollection* shows you how to write an *Iterator* class, if you are unclear about how to do it.

In addition to writing the three decorator classes, I also want you to write the following two predicate classes that implement *Predicate<String>*:

- A class *StringPrefix*. Its constructor should receive a string denoting a prefix, and its *test* method should determine if the input string has that prefix.
- A class *StringLength*. Its constructor should receive an integer, and its *test* method should determine if the length of the input string is no longer than that value.

Finally, you need to write a test program, called *CollectionTest.java*. I will supply you with a text file *dictionary.txt*. This file contains a large number of common English words. In your test program, you should use a scanner to read the words of this file into an *ArrayList*. You should then use your decorator classes to create the following two collections and print their contents:

- A collection containing the words having the prefix "ka", having length no more than 5, and sorted in reverse alphabetical order.
- A collection containing words having the prefix "vivi" or "pse", and sorted by word length ascending. (Use the *MergedCollection* decorator to merge the "vivi" and "pse" words.)

WHAT TO SUBMIT: You should submit the following files to Canvas:

- the file *BorderTest.java*, for problem 1;
- a file containing the answers to problem 2;
- a file containing the following six Java files, for problem 3: *CollectionTest.java*, *FilteredCollection.java*, *MergedCollection.java*, *SortedCollection.java*, *StringPrefix.java*, and *StringLength.java*.

Please make sure that your Java classes are not in any package. Then create a zip file containing the files, and submit it to Canvas.