CS 2510 Exam 3 – Fall 2010

Name: _____

Student Id (last 4 digits): _____

• Write down the answers in the space provided.
• You may use all parts of the Java language we have learned. If you need a method and you don't know whether it is provided, define it. You do not need to include the curly braces for every `if` or every `else`, as long as the statements you write are correct in standard Java.
• For tests you only need to provide the expression that computes the actual value, connecting it with an arrow to the expected value. For example `s.method() -> true` is sufficient.
• Remember that the phrase "develop a class" or "develop a method" means more than just providing a definition. It means to design them according to the **design recipe**. You are *not* required to provide a method template unless the problem specifically asks for one. However, be prepared to struggle if you choose to skip the template step.
• We will not answer *any* questions during the exam.

Good luck.

Problem	Points	/
1A		/ 6
1B		/ 4
1C		/ 8
2A		/ 2
2B		/ 8
2C		/ 6
Total		/34

30 Points

Problem 1

We would like to know which letters of alphabet occur most frequently in some text. Assume that we have a limited alphabet that consists only of the following letters: `a b c d e f space`.

A. 6 POINTS

Design the method `computeHisto` (in the `Examples` class) that consumes a `String` and produces a histogram of the letters that occur in the given `String`.

The data that represents the histogram should be of the type `ArrayList<LF>`, where the class `LF` that represents a letter and its frequency is given by the class definition on the next page.

Use helper method as appropriate.

Note: For the entire exam we represent single letters as `String`s of length one, to avoid the conversion between `char` and `String` data types.

```
// to represent the letter frequency information
class LF{
  String letter;
  int freq;

  LF(String letter){
    this.letter = letter;
    this.freq = 0;
  }

  // increment the frequency count for this letter
  void inc(){
    this.freq = this.freq + 1;
  }

  // define hashCode to match the equals method
  public int hashCode(){
    return this.letter.hashCode();
  }

  // two objects are equal if they represent the same letter
  // regardless of the frequency recorded
  public boolean equals(Object o){
    if (o == null)
      return false;
    else {
      LF lf = (LF)o;
      return this.letter.equals(lf.letter);
    }
  }
}

class Examples{
  LF aa = new LF("a");
  LF bb = new LF("b");
  LF cc = new LF("c");
  LF dd = new LF("d");
  LF ee = new LF("e");
  LF ff = new LF("f");
  LF spc  = new LF(" ");


  void testLF(Tester t){
    t.checkExpect(this.aa.freq, 0);
    this.aa.inc();
```

```
    t.checkExpect(this.aa.freq, 1);
    LF aaa = new LF("a");
    t.checkExpect(this.aa.equals(aaa), true);
    t.checkExpect(this.aa.hashCode(), aaa.hashCode());
    // reset
    this.aa = new LF("a");
  }

  ArrayList<LF> histoList = new ArrayList<LF>();

  // initialize the histogram list for our alphabet
  void initHistoList(){
    this.histoList.clear();
    this.histoList.add(this.aa);
    this.histoList.add(this.bb);
    this.histoList.add(this.cc);
    this.histoList.add(this.dd);
    this.histoList.add(this.ee);
    this.histoList.add(this.ff);
    this.histoList.add(this.spc);
  }
}
```

B. 4 POINTS

Before we can sort the histogram to give us the letters in the decreasing order of frequencies, we need to define the class `FreqComp` that implements the `Comparator<T>` interface for the data of the type `LF`.

The `Comparator<T>` interface is defined as follows:

```
// to define a method that compares two objects
interface Comparator<T>{

  // return < 0 if the first objects comes before the second one
  // return 0 if the objects are equal
  // return > 0 if the first object comes after the second one
  public int compare(T t1, T t2);
}
```

Design this class.

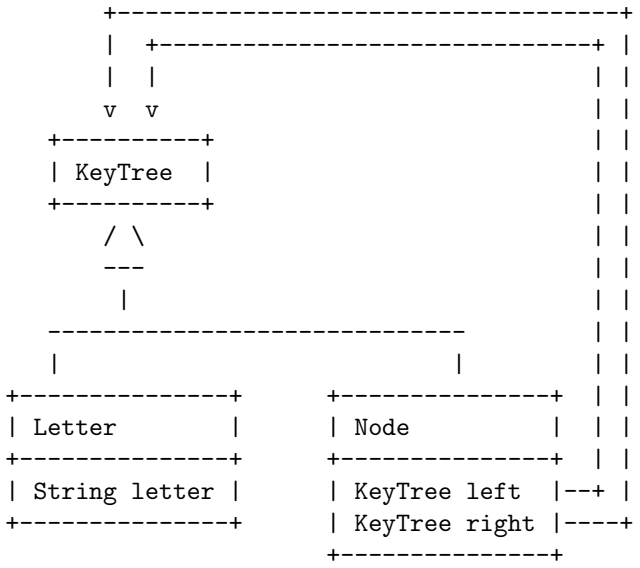
C. 8 points

- Design the method `sort` in the `Examples` class that sorts the given `ArrayList` using the given `Comparator`. You may use any of the sorting algorithms we have learned (but not the Java libraries sort method).
- Design the tests for this method that uses the `ArrayList<LF>` and your `Comparator<LF>` implementation.

30 Points

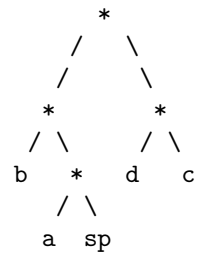
Problem 2

The following class diagram represents a small class hierarchy `KeyTree`.



We can use an instance of this class as a key used to decode a secret message as follows.

Suppose the object `kt` of the type `KeyTree` represents the following information:



The `String "lllrlrllrrrrlrlll"` encodes the message `bad cab`. The `String ll` represents `b`: go left, left; the `String lrl` represents `a`: go left, right, left; the `String rl` represents `d`: go right, left; etc.

A. 2 points

Define the object `kt` of the type `KeyTree` that represents the information shown above.

B. 8 points

Design the method `nextLetter` for the `KeyTree` class hierarchy that produces the next (first) letter encoded in the given `String`. The method should throw a `RuntimeException` if the given `String` is empty or contains letters other than `l` or `r`.

... This page is intentionally left blank ...

```
*** class ArrayList<T>: ***

//remove all elements from this list
void clear()

// add the given element at the end of this list
boolean add(T t)

// add the given element at the given index, shifting all items after
// to the right
void add(int index, T t)

// Returns the element at the specified position in this list
T get(int index)

// Returns the element at the specified position in this list
// set the value of the element at the specified position
// to the given value
T set(int index, T t)

// Returns true if this list contains no elements
boolean isEmpty()

// Removes the element at the specified position in this list
T remove(int index)

// Returns the number of elements in this collection
int size()

*** class String: ***

// Returns the length of this string
int length()

// Tests if this string ends with the specified suffix
boolean endsWith(String suffix)

// Returns a new string that is a substring of this string,
// starting with the character at the given index
String substring(int beginIndex)

// Returns a new string that is a substring of this string,
// starting with the character at the given index
// and ending at the given endIndex
String substring(int beginIndex, endIndex)
```