CS 2510 Exam 3 – Summer 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		/ 3
1B		/ 5
1C		/ 4
1D		/ 4
1E		/ 6
1F		/ 8
1G EXTRA		/ 8
Total		/30

30 Points

Problem 1

We are trying to define the most efficient way for encoding the letters of alphabet using only sequences of bits (values 0 and 1). David Huffman gave us some suggestions.

In each of the following problems you will follow one part of Huffman's suggestions.

Start by looking at the text we want to encode. It is represented as one `String`. For example, the text may be the following 45 letters:

```
In the midst of the word he was trying to say
....x....x....x....x....x....x....x....x....x
```

or, for the examples you should use, just 20 letters long:

```
oh, say, can you see
....x....x....x....x
```

Your first task is to produce a histogram that records how often each character (including the space) occurs in the given String.

A. 3 POINTS

We decided to use the following the data representation for the histogram data, so that you can easily tell how often each letter appears in the text. The histogram is then a simple `ArrayList` of item of the type `CHD`.

Class CHD (Character Histogram Data):

```
+------------+
| CHD        |
+------------+
| char c     |
| int occurs |
+------------+
```

Make examples of the histogram you would get from the second text given on the previous page.

B. 5 POINTS

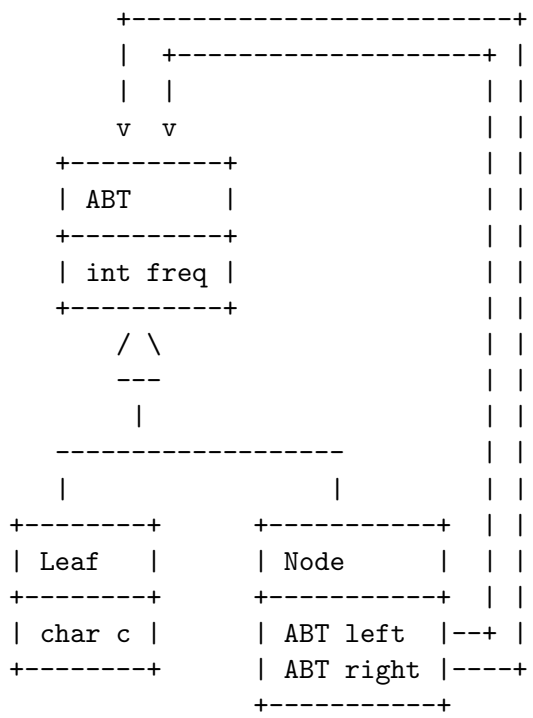
Design the method `computeHisto` that computes the histogram for a given text.

C. 4 POINTS

The encoding for each letter in our `String` is given by a binary tree we can build, once we know the frequencies of all letters, including the space character.

We first show examples of the trees that represent such encodings.

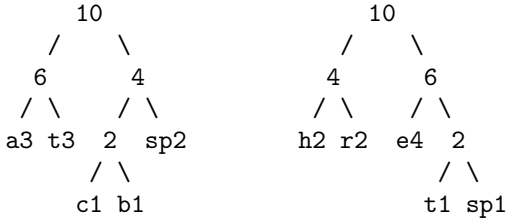
The ABT class hierarchy is define as:



The code that defines these classes is included as well:

```
abstract class ABT{
  // the frequency of this character or a collection of such
  int freq;
}

class Leaf extends ABT{
  char c;

  Leaf(char c, int freq){
    this.c = c;
    this.freq = freq; }
}

class Node extends ABT{
  ABT left;
  ABT right;

  Node(ABT left, ABT right){
    this.left = left;
    this.right = right;
    this.freq = left.freq + right.freq; }
}
```

Make examples of the following two trees (the first one represents the optimal encoding for the `String` *cat at bat*, the second one represents the optimal encoding for the `String` *here there*):



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

D. 4 POINTS

To design our encoding, David Huffman tells us that we should save the histogram data in a priority queue where the highest priority item is the one with the lowest frequency of occurrence, and then convert it into a binary search tree is a clever way that then defines the encoding for every character in our original `String`.

The only thing you need to know at this point is that the priority queue you have to build will contain as data instances of `ABT` and will use the `freq` field to determine the priority.

That means we need a `Comparator` that compares two instances of `ABT`.

Design such `Comparator`.

E. 6 POINTS

The encoding for the character `t` in your first example is the `String "01"`, in the second example it is the `String "110"`. Notice that the numbers `0` and `1` tell us whether we should traverse the tree to the left or right.

Design the method `findPath` for the `ABT` classes that consumes a character and produces a `String` that represents this encoding and ends with the given character.

So, for the input `t` your first tree would produce the `String "01t"`, your second tree would produce the `String "110t"`.

F. 8 POINTS

Design the method `decode` for the `ABT` classes that consumes a `String` that represents an encoding of a character and produces a `char` that the encoding represents. The method should throw an exception if the given `String` is not a valid encoding for the given tree.

So, in your first tree the input `String "01"`, would produce the `char t`, in the second tree the input `String "110"` would produce the `char t`.

G. 8 POINTS EXTRA CREDIT

Design the method `buildTree` that consumes the priority queue we have built and produces the `ABT` that represents the encoding. It works as follows:

- if the size of the priority queue is empty, no tree can be built and it throws an exception.
- if the size of the priority queue is 1, the priority queue contains just one `ABT`, and the method removes it from the priority queue and returns it.
- otherwise, the method removes the two items with the highest priority and adds to the priority queue a new `ABT` that has the two removed `ABT`s as its left and right subtrees.

The documentation for the needed methods for the Java`PriorityQueue` is attached.

The method `buildPQ` shown below has been used to build the initial priority queue.

```
// insert the frequency data into a priority queue
// produce an priority queue of CF data
public PriorityQueue<ABT> buildPQ(ArrayList<CHD> chdList){

  PriorityQueue<ABT> pq = new PriorityQueue<ABT>(20, new FreqComp());

  for (CHD ch: chdList){
    pq.offer(new Leaf(ch.c, ch.occurs));
  }
  return pq;
}
```

... 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 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 char value at the specified index
char charAt(int index)

// 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)


class PriorityQueue<T>:

// the constructor that reserves the given capacity
// that orders the elements according to the specified comparator
PriorityQueue(int initalCapacity, Comparator<T> comparator)

// Inserts the specified element into this priority queue
boolean add(T t)

// Inserts the specified element into this priority queue
boolean offer(T t)

// Retrieves, but does not remove, the head of this queue,
// or returns null if the queue is empty
T peek()

// retrieves and removed the head of this queue
// or returns null if the queue is empty
T poll()

// retrieves and removed the head of this queue
// or returns null if the queue is empty
T remove()

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