Problem Set 6:

Trees

Note: You will be graded in part on your coding style. Your code should be easy to read, well organized, and concise. You should avoid duplicate code.

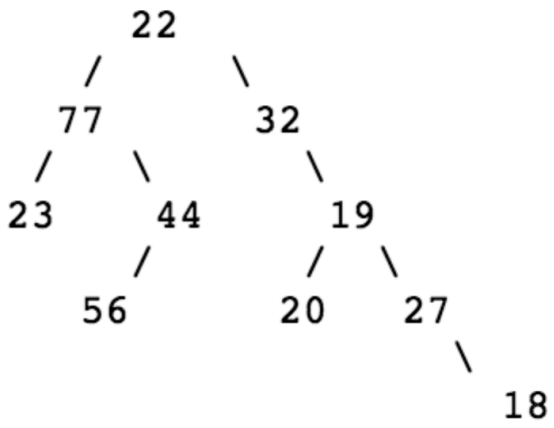
Homework Source: http://www.cs.cmu.edu/~mrmiller/

Download hw10.zip and complete the code as described below.

1. **Trees**

   Create a plain text file `trees.txt` in your hw10 folder, and answer the following questions.

   a. Consider the following tree:



   b. Write its preorder, inorder, and postorder traversals of the above tree.
   c. Write the postorder traversal of a binary tree T where each node stores a single number and
      - its preorder traversal is `24, 17, 32, 14, 3, 6, 16, 25, 12, 18, 7, 2`
      - its inorder traversal is `32, 17, 3, 14, 16, 6, 24, 12, 25, 7, 2, 18`

d. What is the maximum number leaves for a binary tree with height H, where the height of a single node tree is 0?
e. What is the maximum number internal nodes (non-leaves) for a binary tree with height H, where the height of a single node tree is 0?
f. A quad tree is tree where each node has at most 4 children. What is the minimum height of a quad tree that contains 21 nodes? What is the maximum height of q quad tree that contains 21 nodes?

---

2. Iterators

a. Complete the `ArrayIterator` class that implements the `Iterator` interface. The constructor has one parameter, an array, and constructs an `Iterator` object that iterates over the specified array. Your implementation should meet the specification in the `ArrayIterator` API.
Note: Your `ArrayIterator` should not support remove.
b. In the `IteratorPractice` class implement the following methods. You will use Java's built-in iterators for part i and iii; you will use the iterator you implemented for part ii.
   i. Write a method `printStrings` that has two parameters, an `List<String>` and a String. The method prints all the strings in the list that start with the specified string. For example, if the specified string is "121", then for the list

```
["121", "15-121", "12", "121-help", "123"]
```

it prints

```
121    121-help
```

Use Java's enhanced for-loop for Lists.

ii. Write a method `printStrings2` that has two parameters, an array of Strings and a String. Similar to the method above, it prints all the strings in the array that start with the specified string. **Use your `ArrayIterator` to print all these strings.**

iii. Write a method `removeRepetitions` that has one parameters, an `List<String>`, and removes all the repetitions immediately following any

element in the list. For example, below is the contents of a list before and after calling `removeRepetitions`.

before: `[2, 2, 7, 5, 5, 5, 3, 5]`
after: `[2, 7, 5, 3, 5]`

Use Java's built-in iterator for Lists to remove the elements.

3.

---

4. **SLLMap**

Complete the `SLLMap` class that implements the methods of the `BasicMap` interface as specified. The SLLMap maintains a sorted singly-linked list of map entries. Each entry has key, its associated value, and a reference to the next entry. The entries in the linked list must be **sorted in ascending order according to the *natural ordering* of the keys**. Therefore, all the keys must implement the Comparable interface.

Include a comment before each method (except keySet) that states the **worst-case running time** for the method assuming that the map contains *n* entries.

For the `keySet()` method you should return an instance of the `java.util.TreeSet` class. It is one of Java's implementations of the `Set` interface.

For full credit:

- Your methods must not visit more nodes than necessary (ie, take advantage that the elements are sorted); and
- You must not add more fields to the class.

Note that whenever a method has a parameter key of type `Object`, you will need to cast the reference to the appropriate type. For example, if the parameter is declared `Object key`, then you will need to write `(K) key` to cast parameter to the type `K`.

---

5. **Animal Shelter**

You will simulate adoptions at a dog pound. Customers specify the breed of the dog they want to adopt. If there are any dogs of that breed, they will be given one; othewise they will be told they cannot adopt.

You will read of file of file of dog breeds and the name of the dogs available for each breed from which you will create a Map. Then you will print the data from your map. Next, you will read of file of adoption requests, and for each determine if there is a dog still available that meets the request. Finally, you will print the map at the end.

In the `AnimalShelter` class complete the methods listed below. For this exercise you will use the Java `TreeMap` class that implements the Java `Map` interface. It has the same methods as the `BasicMap` interface.

- Complete the `readShelterData` method that reads the file specified in the String parameter. The file contains the dog breeds and the name of the dogs available for each breed. Each line of the file contains the dog names available for one breed. The first "token" is the breed, and the remaining tokens are the names of dogs available for that breed. All tokens are separated by one semi-colon character. For example, the file **dogs.txt** contains the following lines:

  ```
  poodle;fifi;snookums   beagle;Snoopy;Rover;Spot
  collie;Sam;Chauncy;Miranda;Kelly   shepherd;Fido;Jugger;
  bulldog;Spike
  ```

  Here, for example, there are two **poodle**s at the shelter, named **Fifi** and **Snookums**.

  The method already contains code to read the file data. Note that it uses two Scanners. One Scanner reads from the file and reads a whole line of data at a time from the file. The second Scanner reads from single line and reads one token at a time.
  The `useDelimiter` specifies that each tokens are separated by a semi-colon.

  You will complete the method to create a **Map** whose keys are the breeds, and whose values are **Set** of dogs (their names) available of that breed.

a. Complete the `printShelterData` method. The **breeds** should appear in alphabetical order, one per line, each followed -on the same line- by its **Set** of **dog names**. The dog names **do not** have to be in alphabetical order: print them the simplest way you can in Java, since the order is not important. The iterator for the `keySet` of the `TreeMap` will give you the breeds in alphabetical order. For example, the file above would produce:

```
Here are the animals left at the shelter (sorted by breed):
beagle: [Spot, Snoopy, Rover]   bulldog: [Spike]   collie:
[Chauncy, Kelly, Miranda, Sam]   poodle: [Fifi, Snookums]
shepherd: [Jugger, Fido]
```

b. Compete the `readAdoptions` method that reads a file that contains information about the people coming to adopt a dog. Each line of the file contains one request. The first token is the person's name and the second token is the breed they want. The tokens are separated by one semi-colon character. The code to read the file is given to you.

If **(a)** the requested breed is in the **Map** and **(b)** there are any dogs available from that breed, then the person adopts one of these dogs. To determine the name of the dog to adopt, just get the first name found when iterating through this **Set**. Remember that once a dog is adopted, it is no longer available for others to adopt (so remove it from the **Set**).

For example, the input file **adoptions.txt** contains the following lines:

```
  John;poodle   Mary;collie   Joanne;bulldog   Tim;collie
Joe;dachsund   Katie;shepherd   Sarah;beagle   Becky;poodle
Lana;bulldog
```

For each adoption request, print the name of the person followed by either **(a)** the name of the dog they adopted (e.g., **John is adopting Fifi**) OR **(b)** a statement that they can't adopt a dog of that breed (e.g.„**Joe CANNOT adopt a dachsund**). (See the sample interaction below.) Note, the breed may not even be in the **Map** (see Joe trying to adopt a dachsund) or there may be no dogs of that breed available for adoption (see Lana trying to adopt a bulldog: Joanne -adopting earlier- got the only one).

c. Complete the main method so that it follows the sample interaction below. User-typed information appears in *italics*. Your output should "match" this one. Note, though, that other adoptions are also possible. For instance, **John** (see first adoption line) wants a **poodle**, so he can adopt either **Fifi** or **Snookums**. When **Becky** (second to last adoption line) comes along, also wanting a **poodle**, she will get whichever one is left.

Of course, a dog may be adopted by only one person. And, regardless of who adopts whom, there will be 2 **beagles** and **collies** left, 1 **sheperd** left, an no **bulldogs** or **poodles** left.

Enter dog file name: *dogs.txt* Here are the animals left at the shelter (sorted by breed): beagle: [Spot, Snoopy, Rover] bulldog: [Spike] collie: [Chauncy, Kelly, Miranda, Sam] poodle: [Fifi, Snookums] shepherd: [Jugger, Fido] Enter adoptee file name: *adoptions.txt*
John is adopting Fifi. Mary is adopting Chauncy. Joanne is adopting Spike. Tim is adopting Kelly. Joe CANNOT adopt a dachsund. Katie is adopting Jugger. Sarah is adopting Spot. Becky is adopting Snookums. Lana CANNOT adopt a bulldog. Here are the animals left at the shelter (sorted by breed): beagle: [Snoopy, Rover] bulldog: [] collie: [Miranda, Sam] poodle: [] shepherd: [Fido]