Project Collections

Collections Assignment - Iterator Design Pattern

a. For this assignment, you are required to demonstrate the use of the Iterator Design Pattern and working with different objects in the Collections Framework.
b. For each of the three parts, provide an analysis of the results -- which collection is more efficient and why. Base your opinions on your timed results.
c. For each part, when you run the program, pass in a command line argument to determine which Collections data structure the program will use.

Part One

a. This program will work with maps. Use the attached file "QnotfollowedbyU". It contains the words that start with Q not followed by U.
b. Read this file into a TreeMap data structure and a HashMap data structure.
c. Time the results. Which data structure is more efficient for loading the data?
d. Create a second file that contains each of the letters in the alphabet and their scrabble point value. You can find the file at Scrabble Site.
e. For each of the words in the file, find their point value and print it out.
f. Time the results. Which data structure is more efficient for searching data?
g. Important: You cannot use both Collection objects in the run of the program. You must run the program twice -- once for each data structure.

Part Two

a. Insert all words from a large file such as the novel Alice in Wonderland "included in the dropbox", into a hash set and a tree set.
b. Time the results. Which data structure is more efficient for inserting data?
c. Choose a word that is in the novel and record the time it takes to search for that word 100 times.
d. Time the results. Which data structure is more efficient for searching data?
e. Important: You cannot use both Collection objects in the run of the program. You must run the program twice -- once for each data structure.

Part Three

a. A Scavenger Hunt is a party game that people play in teams. They are given a list of items to find throughout the surrounding neighborhood. Our challenge is to find the best data structure to store the results of the hunt.
b. Create a list of 100 items that you might have people find on the hunt. These should be items that they have a chance of finding throughout the neighborhood. For example,

yesterday's newspaper, a shoelace, a paperbag,.... Load the list anyway you want (console input, flatfile, etc.)

- You will run the program two times, once using a set of ArrayLists, and once using a set of LinkedLists. Our purpose is to find the more efficient data structure to use for a scavenger hunt.
- Use an Iterator to traverse the Collection from beginning to end. Then iterate through the Collection from end to beginning (backwards).
- Time the results. Which data structure is more efficient for looping through the entire Collection?
- Ask the user how many teams will play the game. Create this number of teams. For each team load all of the items from the list. Shuffle the list after loading the items each time. Find the total time it takes to add the items to all of the teams.
- Ask the user what position in the list to be used for retrieving and inserting elements. Retrieve that element from each of the teams. Find the total time it takes.
- Next insert a new element at that position in each of the lists. Find the total time it takes.
- Use the Random class to generate a number between 0 and the 100. Find the element in the scavenger hunt list at that position. Retrieve that element from each of the lists. Find the total time it takes.
- Display the time in milliseconds for each of these operations.
- Run the program several times with different input values and see how it affects the output.
- Important: You cannot use both Collection objects in the run of the program. You must run the program twice -- once for each data structure.

All Parts

a. Write a brief analysis of your findings along with your recommendations.
b. Make sure you use appropriate exception handling. This would be related to how you initially set up the list of items for the scanvenger hunt and how you have the user enter the values for the number of teams and the element to work worth.
c. The following code snippet will help with the timing

```
Date today = new Date();
long time1, time2;
time1=System.currentTimeMillis();
.
.
.
.
```

```
time2=System.currentTimeMillis();
System.out.println("Time for the operation is: " + (time2-time1));
```

d. If the time difference is too small, try:

```
long startTime = System.nanoTime();
// ... the code being measured ...
long estimatedTime = System.nanoTime() - startTime;
```

For all programs, catch and handle the Exceptions appropriately and validate the input data where needed.