Worksheet Six on Java Collections

Principles of Programming II

Part-time cohort 2018

1. (a) Write a class called `LinkedListRunner` with a `main` method that instantiates a `LinkedList<String>`. Add the following strings to the linked list:

   `aaa, bbb, ccc, ddd, eee, fff, ggg, hhh, iii`

   (for the comma replace with a newline)

   (b) Write a `ListIterator<String>` and use it to walk sequentially through the linked list using `hasNext` and `next`, printing each string that is encountered. When you have printed all the strings in the list, use the `hasPrevious` and `previous` methods to walk backwards through the list.

   Along the way, examine each string and remove all the strings that begin with a *vowel*.

   When you arrive at the beginning of the list, use `hasNext` and `next` to go forward again, printing out each string that remains in the linked list.

   (c) Write a class called `HashedSetRunner` with a `main` method that instantiates a `HashedSet<String>`. Add the following strings to the linked list:

   ```
   aaa, bbb, ccc, ddd, eee, fff, ggg, hhh, iii,
   aaa, bbb, ccc, ddd, eee, fff, ggg, hhh, iii
   ```

   (for the comma replace with a newline)

   Notice that each string appears twice. After adding all the strings, use an *enhanced* `for` loop to walk sequentially through the hash set, printing each string that is encountered.

   How many times does each string appear in the hash set?
   Are the strings in the hash set printed in the same order that they were added?

   (d) Write a class called `TreeSetRunner` with a `main` method that instantiates a `TreeSet<String>` object. Add the following strings to the tree set in order:

   ```
   iii, hhh, ggg, fff, eee, ddd, ccc, bbb, aaa,
   iii, hhh, ggg, fff, eee, ddd, ccc, bbb, aaa,
   ```

   (for the comma replace with a newline)

   Notice that each string appears twice. After adding all the strings, use an *enhanced* `for` loop to walk sequentially through the tree set, printing each `String` that is encountered.

   How many times does each string appear in the tree set?
   Are the strings in the tree set printed in the same order that they were added?

2. Use the following `Person` class and `PersonRunner` class for this problem:

```
package exercises;

public class Person {
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
```

```
    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public String toString() {
        return "Name: " + firstName + " " + lastName;
    }
}

package exercises;

import java.util.TreeSet;

public class PersonRunner {
    public static void main(String[] args) {
        TreeSet<Person> mySet = new TreeSet<Person>();
        mySet.add(new Person("Sally", "Brown"));
        mySet.add(new Person("Fred", "Kelly"));
        mySet.add(new Person("Bill", "Akins"));
        mySet.add(new Person("Julie", "Wilkins"));
        mySet.add(new Person("James", "Langdon"));
        for (Person p : mySet) {
            System.out.println(p);
        }
    }
}
```

(a) Execute the `PersonRunner` class. What causes the run-time error?

(b) Fix the problem by changing the `Person` class and then run the `PersonRunner` class again.

(c) Do the `Person` objects appear in the correct sequence after you have modified `Person`?

3. Repeat the previous problem but develop an alternate solution without changing the `Person` class. (Hint: research the `TreeSet` class for alternate constructors.) What code needs to be added or modified?

4. (a) Consider the following line of code:

   ```
   Set<String> names = new HashSet<>();
   ```

   Why is the variable names of type `Set` rather than type `HashSet`?

   (b) Look in the Java documentation for `HashSet`. Which methods are listed?

   Notice that method `addAll` is not listed as a method of `HashSet` but it is contained in the documentation for interface `Set`. If you include the following code in a program that imports `java.util.*`, it will compile cleanly even though `addAll` is not listed directly as a method of the instantiated object.

   ```
   Set<String> names1 = new HashSet<String>();
   Set<String> names2 = new HashSet<String>();
   names1.addAll(names2);
   ```

   Why?

   (c) Write a program that creates two `HashSet`s of `String`s. Demonstrate that the contents of the second `HashSet` can be added to the first using `addAll`.

   (d) Write a program that adds thirty `String` objects to a `HashSet`. Use an `Iterator` to print the elements of the `HashSet`. Is the entry sequence of `String`s identical to the printed `Iterator` sequence?

   Why or why not?

5. (a) Unlike the `ListIterator` interface, the `Iterator` interface for a set cannot add an item to a set. Is there a reason for this? If so, what is it?
   (b) The `Iterator` interface implemented for set iterators contains a next method. Why does the interface not contain a `previous` method?

6. Write a program that demonstrates that `HashSet` objects do not store duplicate elements and that adding a duplicate element has no effect on the set.

   Explain how your program demonstrates this idea.

7. For each of the following problems, indicate whether it would be more appropriate to use a `HashSet` or a `HashMap`, and describe the elements or keys/values:

   (a) A program that keeps track of bank account information (name, account number, account type, balance)
   (b) A phonebook program that stores your friends' names and phone numbers
   (c) A program to inventory the contents of your refrigerator
   (d) A program that stores a list of your favourite fruits
   (e) A program that stores a list of the favourite fruits of all the students in your class

8. Suppose you write a program to keep track of the friends of all the students in your class using a `HashMap`.

   What would you use for the keys?
   What would you use for the values?

9. Write a program that stores the following information in a `HashMap`:

   Sue is friends with Bob, Jose, Alex, and Cathy
   Cathy is friends with Bob and Alex
   Bob is friends with Alex, Jose, and Jerry

   After storing the information, prompt the user to enter a name. If the name that is entered is Sue, Cathy, or Bob, print out the name and the list of friends. Otherwise print a message indicating that the name is not in the `HashMap`.

10. For this question you will create an `Employee` class. The class contains the id (of type `long`), name, phone number, and job title of the employee.

    (a) You must implement the methods:

    ```
    public boolean equals(Object other)
    public int hashCode()
    ```

    You should include appropriate constructor and accessor methods.
    (b) Now write a program that stores a set of employees in a `HashSet<Employee>` and prints out the elements in the set.

    Try adding the same employee twice.
    (c) How can you simplify the `equals` and `hashCode` methods if you know that employees are uniquely determined by their ID?

11. Write a program called `PriorityQueueRunner` that uses the `Person` class below. Notice that this class implements `Comparable<Person>`.

```
package exercises;

public class Person {
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
```

```
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public String toString() {
        return "Name: " + firstName + " " + lastName;
    }
}
```

Use this class to create `Person` objects for each of the names below, and add them to a `PriorityQueue<Person>` object.

Empty the queue by polling it and printing the `Person` objects that are returned.
Are `Person` objects returned in sequence?

```
Sam Smith
Charlie Black
Betty Brown
Jessica Stewart
John Friday
Frank Foley
```

12. A *stack* is a useful tool for reversing a list of items. Pushing items onto a stack and then popping them off results in the items being returned in the reverse order from which they were pushed.

    Use the following class for this assignment.

```
package exercises;

public class PalindromeRunner {
    public static void main(String[] args) {
        PalindromeTester pt = new PalindromeTester();
        if (pt.isPalindrome("bob"))
            System.out.println("bob is a palindrome");
        else
            System.out.println("bob is not a palindrome");

        if (pt.isPalindrome("amanaplanacanalpanama"))
            System.out.println("is a palindrome");
        else
            System.out.println("amanaplanacanalpanama is not a palindrome");

        if (pt.isPalindrome("abcdefghijklmnopqrstuvwxyz"))
            System.out.println("abcdefghijklmnopqrstuvwxyz is a palindrome");
        else
            System.out.println("abcdefghijklmnopqrstuvwxyz is not a palindrome");
    }
}
```

You will need to build a `PalindromeTester` class and provide it with a with the `boolean isPalindrome(String s)` method that returns `true` if the string `s` is a palindrome, otherwise it returns `false`.

Populate the class with a `Stack<Character>` object. Push each character of the passed string onto the stack. Java will *autobox* each `char` into a `Character` object.