Lab 4 - Implementing Generics and Interfaces

due Sep 13th at 23:59

Objective

Today's lab will help you get familiar with

- Write your own generic class
- Further understand abstract classes and interfaces
- implement a comparable class

Abstract Classes:

Abstract Classes are classes that are partially implemented, but some methods are indicated as abstract. These abstract methods are not implemented and anyone extending an Abstract Class must implement these methods. If one or more abstract methods remains unimplemented in the inheriting class, then that class must also be abstract.

Generic Classes:

The idea of a generic is to write an algorithm/method without specifying the data type that will be used. For example, in previous labs you wrote methods that only worked for a specific type. With Generic Classes you could have implemented your program to work for a variety of data types. In previous labs, you have used generic classes, e.g., LinkedList and ArrayList. In this lab, you will implement a generic class.
The code below illustrates how you can create your own generic class. In the example two generic types are specified:

```java
public class Entry<K, V>{

        private final K mKey;
        private final V mValue;


        public Entry(K k,V v){
            mKey = k;
            mValue = v;
        }

        public K getKey(){
            return mkey;
        }

        public V getValue()   {
            return mValue;
        }

        public String toString()  {
            return "(" + mKey + ", " + mValue + ")";
        }
}
```

Inheritance can then be implemented in the following way:

```java
public class DifferentEntry<K, V> extends Entry<K, V> {

        /* CLASS BODY */
    }
```

Interfaces:

Java provides a useful tool for specifying functionality in the Class Library. Interface is a structure that defines a method set that must be implemented to meet a specific set of functionality. Often interfaces are provided in conjunction with another class that is designed to manipulate classes implemented by other developers.
For example, let's say there is a class called College that simulates any college campus. The developers of the College class also provide an interface called Student, specifying specific methods that all classes that

implement the interface must implement/provide. Then developers can create a LafayetteStudent that inherits from the College class and implements the Student interface.
This technique is called contract programming because the programmer is entering into a contract by implementing a defined interface.

Assignment:

1. Create a class RandomStuffContainer which contains a generic ArrayList. The RandomStuffContainer class must be restricted to Comparable types. This can be done via the declaration public class RandomStuffContainer<T extends Comparable<T>>. The following should be implemented as well:
   a. Create a constructor for the class to create an empty array list.
   b. Create a public method (*addToFront*) that takes one generic parameter. The element will be inserted as the first element of the ArrayList, moving all the other elements up by one position.
   c. Create a public method (*addToBack*) that takes one generic parameter. The element will be inserted as the last element of the ArrayList.
   d. Create a public method (*selectionSort*) that sorts the ArrayList using the selection sort algorithm.
   e. Create a public method (*bubbleSort*) that sorts the ArrayList using the bubble sort algorithm.
   f. Create a public method (toString) returning a String concatenating the outputs of each elements toString().
2. Create a class called RandomIntegerContainer extending RandomStuffContainer with:
   a. sum() returns the sum all all elements
3. Write an ExperimentController so that it evaluates the performance of both sorts against each other.
4. Unit test your classes.

Submission:

In addition to your code you must submit lab notes (in PDF format). Save your notes in the project folder before you compress it and upload it.

Grading:

1. unit testing - 2
2. RandomStuffContainer - 3
3. RandomIntegerContainer - 1
4. Experiment Controller - 2
5. Style/commenting – 1
6. Lab notes - 1