HW07

Homework 7 (Due 11:59pm Monday, November 8, 2021)

Submit via SVN

Preliminaries

This homework should be done in Full Java (using DrJava, IntelliJ, Eclipse, or a text editor and command line compilation and execution). The Functional Java language in DrJava does not work for more complex OO code such involving the visitor pattern. In this assignment, you will re-implement some of the functions on IntLists assigned in Homework 7 using the visitor pattern.

As before, your program must support the object-oriented formulation of lists of integers defined the composite class hierarchy where

- **`IntList`** is an abstract list of **`int`**.
- **`EmptyIntList`** is an **`IntList`**
- **`ConsIntList(first, rest)`**, where **`first`** is an **`int`** and **`rest`** is an **`IntList`**, is an **`IntList`**

The Homework Support files **`IntList.java`**, **`IntListVisitor.java`**, **`LengthVisitor`**, and **`IntListTest.java`** provide a starting point for your code. Feel free to edit these files and omit files that are not needed in this homework assignment.

Problems

Apply the visitor design pattern to define visitor classes implementing the **`IntListVistor`** interface **`IntList`** and its subclasses given above to formulate all of the following methods as visitors. Develop a JUnit test class, **`IntListTest`** to test all of your new methods in the **`IntList`** class. Use the **`LengthVisitor`** example as a guide for defining your new visitor classes. Augment provided the test class **`IntListTest.java`** to include test methods for each of your visitor classes. Confine your documentation to writing contracts for each visitor using **`javadoc`** notation (a comment preceding the corresponding definition) beginning with **`/**`** and closing with **`*/`** for each visitor class. Use the documentation of **`LengthVisitor`** in the resource files on Canvas or Piazza as an example.

- (15 pts.) **`IntList reverse()`** constructs a list that is the reversal of **`this`**. Name your visitor class **`ReverseVisitor`**. **Hint**: this computation is faster and simpler if you introduce a help "method" (also a visitor).that takes an argument.
- (15 pts.) **`IntList notGreaterThan(int bound)`** returns a list of elements in this list that are less than or equal to **`bound`** . Name your visitor class **`NotGreaterThanVisitor`**.
- (15 pts.) **`IntList remove(int key)`** returns a list of all elements in this list that are not equal to **`key`**. Name your visitor class **`RemoveVisitor`**
- (15 pts.) **`IntList subst(int oldN, int newN)`** returns a list of all elements in this list with **`oldN`** replaced by **`newN`**. Name your visitor class **`SubstVisitor`**

- (20 pts.) **`IntList merge(IntList other)`** merges this list with the input list other, assuming that this list and other are sorted in ascending order. Note that the lists need not have the same length. Name your visitor class **`MergeVisitor`**. **Hint**: add a "method" **`mergeHelp(ConsIntList other)`** that does all of the work if one list is non-empty (a **`ConsIntList`**). Only **`mergeHelp`** is recursive. Use dynamic dispatch on the list that may be empty. Recall that **`a.merge(b)`** is equivalent to **`b.merge(a)`**. You can formulate help methods as visitors.
- (20 pts.) **`IntList mergeSort().`** Leveraging the **`merge`** "method" you just wrote (as a visitor), write **`mergeSort()`** that sorts an **`IntList.`** Recall that you need to write a help function that splits a list approximately in two.

Testing Tricks

In Racket, the **`equal?`** function performs structural equality. Java does not include such a built-in operation. For the **`IntList`** composite type, we overrode the inherited **`equals`** method (trivially defined in class **`Object`**) by an **`equals`** method that implements structural equality but it is slightly more complex than you might expect. Recall that the argument passed to **`equal`** has type **`Object`**. Hence, we have to worry about the class of the argument; the simple (and IMO best) definition of structural equality is to mandate that objects cannot be equal unless they are instances of the same class. Study the definition of the **`equals`** method in class **`ConsIntList`**. Unfortunately, we can write the body of this method the **`return`** of a boolean-valued expression, because Java does not support a notion of **`local`** or **`let`** at the level of expressions. So the body is an **`if`** statement where explicit **`return`** statements in the consequent statement and alternative statement. Notice that we still programmed in a functional style without any mutation.

To test the computations that yield results of composite type, we can either define structural equality over the composite type (as we did for **`IntList`**) or write an intelligible **`toString`** method for the composite type (which I strongly recommend for debugging purposes) and compare the **`toString()`** representations of the composite type. But beware that **`toString()`** equality may not imply structural equality and vice versa. You should always endeavor to make them agree.

Extra Credit

If you do the assign using generic types (**`List<T>`** instead of **`IntList)`**, you can earn an additional 25 points. See the generic version of **`LengthVisitor.`**