5 Methods for Complex Class Hierarchies; Libraries

Etudes

5.1 Etude

Do as many as you can of the exercises 19.3 - 19.9. ∎

Note:

If you feel that you really understand the concepts covered in these etudes, you may do only a couple of them. However, make sure you know how to solve all of them.

Goals:

This assignment has three goals. The first is to understand better how to design methods for complex class hierarchies. The second goal is to learn how to determine whether two complex collections of data represent the same information from the user's perspective. The third goal is to understand better the relationship between interfaces, abstract classes and their concrete derived subclasses.

Main Assignment — Part 1: Binary Search Trees

5.2 Problem: Data elements — Books

1. Design the class *Book* that records the title of the book and the year is was published.

2. Add the method

   int compareTo(*Book other*)

   that produces a negative value if this book comes before the other book, produces zero, if *this* and *other* represent the same book, and produces a positive value if this book comes after the other book — the ordering is lexicographical, and by years, if the titles are the same — both in ascending order. *Make examples to make sure you understand the ordering.*

3. Add the method

   boolean sameBook(*Book b*)

   that determines whether the two *Book* objects represent the same book.

■

5.3 Problem: Data structure - Binary Search Tree

1. Design the classes that represent a binary search tree of *Book*s (the *abstract* class *BST* and its derived classes *Leaf* and *Node*. Use the *compareTo* method you designed to determine the ordering of the elements of the tree. Do not add duplicate books into the tree.

   In the *BST* classes design the following methods:

2. Method *insert* that inserts a book into the binary search tree.

3. Method *count* that computes how many books are recorded in the binary search tree.

4. Method *isEmpty* that determines whether the binary search tree contains any books.

5. Method *contains* that determines whether the binary search tree contains a specific book.

6. Method *getFirst* the produces the first book (in our ordering of books) among the books in the binary search tree. To invoke this method on an empty binary search tree is a grave error. For now, we produce a book with the title "NoBooksInAnEmptyTree" with the year of publication 9999. We will learn soon how to properly signal these kinds of errors.

7. Method *removeFirst* the produces a new binary search tree, after the first book (in our ordering of books) among the books in the binary search tree has been removed. To invoke this method on an empty binary search tree is a grave error. For now, we just produce an empty binary search tree. We will learn soon how to properly signal these kinds of errors.

■

Main Assignment — Part 2: Equality

5.4 Problem

1. Design the method *sameBST* and any other methods needed to determine whether two instances of the *BST* class hierarchy represent the same data.

   Save this work as part one of the assignment.

2. Design the classes *LoBooks* and the needed subclasses to represent a sorted list of books.

3. Design the method *insert* that inserts the given book into the *LoBooks* class hierarchy, preserving the ordering.

4. Design the method *sameLoBooks* and any other methods needed to determine whether two instances of the *LoBooks* class hierarchy represent the same data.

5. Method *count* that computes how many books are recorded in the list of books

6. Method *isEmpty* that determines whether the list of books contains any books.

7. Method *contains* that determines whether the list of books contains a specific book.

8. Method *getFirst* the produces the first book (in our ordering of books) among the books in the list of books. To invoke this method on an empty list is a grave error. For now, we produce a book with the title "NoBooksInAnEmptyTree" with the year of publication 9999. We will learn soon how to properly signal these kinds of errors.

9. Method *removeFirst* the produces a new list, after the first book (in our ordering of books) among the books in the list has been removed. To invoke this method on an empty list is a grave error. For now, we just produce an empty list. We will learn soon how to properly signal these kinds of errors.

   Save this work as part two of the assignment.

▮

Main Assignment — Part 3: Abstractions

5.5 Problem: Designing a common interface

We defined two ways of represented an ordered collection of data. For this part of the assignment, combine in a new file the solutions for the two parts of the code you wrote before. (Of course, do not repeat twice the definition of the *Book* class and its examples.)

1. Design a common interface *ISortedBooks* that contains all methods that are the same (or similar) between the *BST* and *LoBooks*.

   Modify the classes that implement the *BST* and the classes that implement the *LoBooks* as needed. Remember to ...

   Save this work as part three of the assignment.

2. **Extra Credit:**

   Define a common abstract class *ASortedBooks* that defines a concrete method *isSorted* using only the methods that have been lifted to the *ISortedBooks* interface.

   Save this work as the extra credit part of the assignment.

∎