6 Abstracting with Function Objects

Portfolio Problems

1. Include the solution of the Lab 6 in your portfolio.

2. Add a few more examples of tests in the `Examples` class.

3. Define a class `ImageSmallerThanAndGivenKind` that implements the `ISelect` interface with a method that selects image files that are small and of the given kind. Allow the user to decide how small should the images be (measuring the size as the number of pixels in the image) and allowing the user to choose the `kind` of images that should be selected. (All selected images must be of the same `kind`.)

   Test your class definition on several examples before you use it in your `allSuch`, `anySuch`, and `filter` methods.

4. Add test cases that will test the methods `allSuch`, `anySuch`, and `filter` with several instances of the `ImageSmallerThanAndGivenKind` predicate.

Pair Programming Assignment

6.1 Problem

During the lectures we have designed the method `filter` for a list of `Book`s, selecting books written by the given author and books that cost less than the specified price.

A. Define the class `Book` and the classes that represent a list of `Book`s, as well as the `ISelect` interface and the needed classes that implement it to select a given `author`, and to select a book cheaper than the given `price`. Make sure you test the classes that implement the `ISelect` interface.

B. Define the methods `filter`, `orMap`, and `andMap` as we have done in the lectures.

   Note: Until now you are just writing up the lecture notes as a working program.

C. Define the following interface in your project:

```
// to represent a method that determines whether
// book b1 comes before book b2
interface Ordering{
  public boolean isBefore(Book b1, Book b2);
}
```

Define at least two classes that implement this interface. (You may order the books by the price, the length of the book title, author's names, etc.)

D. Design the method `sort` for the classes that represent a list of `Book`s that uses an instance of a class that implements the `Ordering` interface to define the appropriate ordering of the books and produces the list in the correctly sorted order.

E. Design the method `isSorted` for the classes that represent a list of `Book`s that uses an instance of a class that implements the `Ordering` interface to determine whether a list of `Book`s is sorted correctly.

Note: Remember the one task one method rule.

F. Design the method `merge` for the classes that represent a list of `Book`s that merges a sorted list with another sorted list, producing a list that has all elements of both lists — again in a sorted order. An instance of a class that implements the `Ordering` interface determines the ordering of both the original two lists and the resulting list.

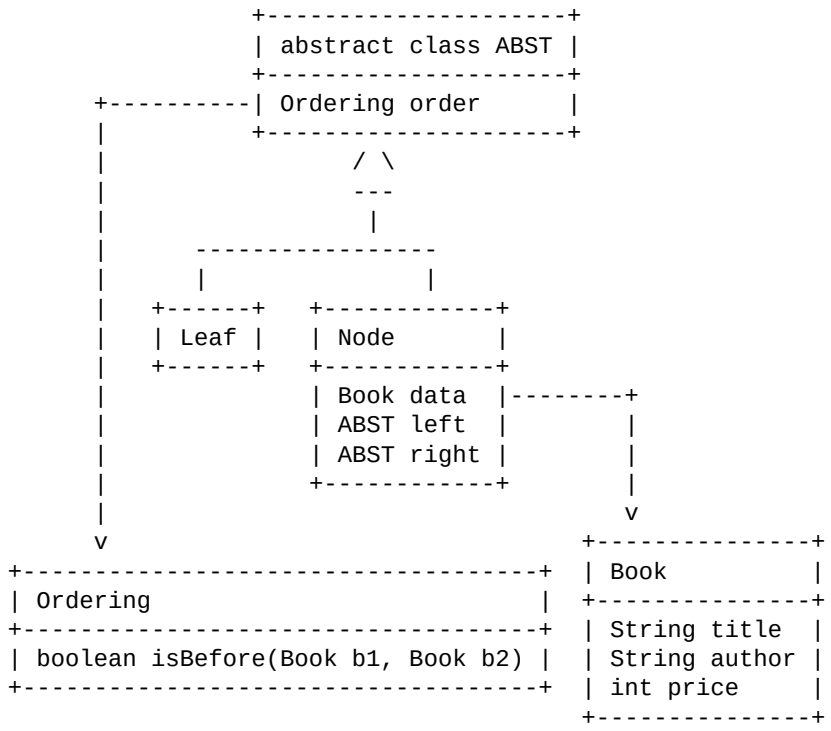
6.2 Problem

You will work with a binary search tree that represents a collection of `Book` objects. It should be very similar to the binary search trees that had only integer data.

The class diagram on the next page should help you.

A. Define the classes that represent a binary search tree of `Book` objects as shown in the class diagram above.

B. Define the method `insert` that inserts a new `Book` into the binary search tree, using the `Ordering` already defined for this tree.



C. Define the method `getFirst` that produces the first `Book` in the binary search tree (as given by the appropriate `Ordering`.

In the `Leaf` class this method should have the following body:

```
throw new RuntimeException("No first in an empty tree");
```

D. Define the method `getRest` that produces a new binary search tree with the first `Book` removed.

In the `Leaf` class this method should have the following body:

```
throw new RuntimeException("No rest of an empty tree");
```