9 Abstracting over Types and Methods

Portfolio Problems

1. Finish Part 9.2 Lab 9.
2. In Assignment 3 you have worked with classes that represent cities in the USA. Modify the City class so that it implements the `Comparable` interface comparing the cities first by state, then by the city name within the state. Make examples of a *binary search tree*s of cities.

Pair Programming Assignment

9.1 Problem: Abstracting over Datatypes

Finish Part 9.3 Lab 9.

9.2 Problem: Abstracting over Traversals

Finish Part 9.4 Lab 9.

9.3 Problem: Abstracting over Methods

We have seen that a class can implement several interfaces, each defining a method that the class has to implement. We now turn to a different problem. In that past we have written methods that determined whether all items in a list of data satisfied some condition, as well as method that determined whether a list of data contained at least one item that satisfied the given condition. We can also count the number of items that satisfy some condition.

A. Download the files

- *Acct.java*
- *Book.java*
- *ILo.java*
- *MtLo.java*
- *ConsLo.java*
- *Examples.java*

Create a new Java Project with all these files and run the project.

B. Our method `contains` determines whether the list of data contains one specific book — using the same-ness criterion defined in the method that implements the `same` interface. We would like to see if the list contains any book that satisfies a broader criterion.

   We define the interface `Selectable` as follows:

```
// to represent a method for selecting items of the type T
interface Selectable{
  boolean isOK();
}
```

   Add a file *Selectable.java* to your project.

   In the class `Book` implement this interface so it selects all books published before 1950.

   In the class `Acct` implement this interface so it selects all accounts with the balance below $2500.

C. Define the method `countSuch` that counts the number of items in the list that satisfy the criterion given by the `isOK` method.

9.4 Problem: Abstracting over Algorithms

We now want to define the methods that determine whether all items in the list satisfy some condition, and whether there is at least one item in the list that satisfies some condition.

In HtDP these two problems have been abstracted into the *andmap* and *ormap* loops. The contracts and purpose statements for these methods were given as:

```
;; andmap: (X -> boolean) (Listof X) -> boolean
;; to determine whether p holds for every item in alox
;; that is (andmap p (list x-1 x-2 ... x-n)) =
;;         (and (p x-1) (and ... (p x-n)))
(define (andmap p alox) ...)

;; ormap: (X -> boolean) (Listof X) -> boolean
;; to determine whether p holds for at least one item in alox
;; that is (ormap p (list x-1 x-2 ... x-n)) =
;;         (or (p x-1) (and ... (p x-n)))
(define (ormap p alox) ...)
```

We see that the each function consumed not only the list of items, but also a function that determined for each item whether or not it should be selected.

If we know that the list contains instances of a class that implements `Selectable` interface, we can use the method `isOK` to check whether any of the items satisfied the desired condition. However, the solution is not sufficient. We may want to first check if all books are old, written before 1950, at other time we may want to see if the list contains only the books by the given author, changing the criterion for the books to be considered *OK*.

We need to be able to define several different methods that determine whether a book is to be chosen.

Instead of the `Selectable` interface, we define an interface `Selector<T>` parametrized over the type `<T>` that can be implemented outside of any class. The method `choose` consumes an item of the type `<T>` and returns `true` or `false` as appropriate:

```
// to represent a method that specifies a selection criterion
interface Selector<T>{

// determine whether the given item should be selected
boolean choose(T t);
}
```

A. Add the file *Selector.java* that contains the definition of the `Selector` interface to your project.

The following two classes illustrate how to choose books written before 1950 and books written by the given author. The definition of the first class is straightforward:

```
// to represent a boolean method that selects old books
class BookBefore1950 implements Selector<Book>{
  BookBefore1950(){}

  // was the given book written before 1950?
  boolean choose(Book b){
    return b.year < 1950;}
}
```

B. Add the file *BookBefore1950.java* that contains the definition of the `BookBefore1950` interface to your project. In the `Examples` class define an instance of the class `BookBefore1950`. Design tests for the method `choose` in this class.

To determine whether the book was written by a given author, we must provide the name of the author we are interested in. If we define a field that will hold the name we are looking for in the class that implements the `Selector` interface, the method `choose` can then compare against it every time it it invoked. Look carefully at the following definition and include this class in your project:

```
// to represent a boolean method that selects
// books written by the given author
class BookWrittenBy implements Selector<Book>{
  String authorname;

  BookWrittenBy(String authorname){
    this.authorname = authorname;
  }

  // was the given book written by 'authorname'?
  boolean choose(Book b){
    return b.author.equals(this.authorname);}
}
```

C. In the `Examples` class define an instance of the class `BookWrittenBy` that selects books written by Hemingway. Design tests for the method `choose` in this class.

D. Design a class `AcctBelow2500` that implements the `Selector` interface with a method that determines whether an account has a balance below $2500. Make sure to include examples of the data and tests.

E. Design the class `AcctOwnedBy` that implements the `Selector` interface with a method that determines whether an account is owned by the owner with the specified name. Make sure to include examples of the data and tests.

F. Design the method `orMap` that determines whether a list contains at least one item that satisfies the `choose` method in the given `Selector`.

   The method purpose and header for the interface `ILo<T>` will be:

```
// In the interface ILo<T>:
// does this list contain an item
// that satisfies the given condition?
boolean orMap(Selector<T> select);
```

G. Run the tests for both lists of books and lists of accounts, using each of the selection methods defined earlier.

H. **Bonus:** Design the method `andMap` that determines whether every item in a list satisfies the `choose` method in the given `Selector`.

   The method purpose and header for the interface `ILo<T>` will be:

```
// In the interface ILo<T>:
// does this list contain an item
// that satisfies the given condition?
boolean andMap(Selector<T> select);
```