7 Abstracting over the Data Type

Portfolio Problems

Start with the program ou wrote for the parts A., B., and C. in Lab 7.

1. Make a examples of books and write several examples that will produce a list of all books by the given author.

   For example if your list contains books by John Steinbeck, Lewis Carroll, William Shakespeare, and others, your examples will produce first a list of all books by Steinbeck, then a list of all books by Carroll, then all books by Shakespeare.

2. Make examples of songs and write several examples that will produce a list of all songs by the given artist.

Pair Programming Assignment

7.1 Problem

Complete Parts D. of the Lab 7, dealing with the lists of books, song, and images.

7.2 Problem

A. Add the following interface to the *ILo.java* file:

```
// to represent a method that mutates the state of this object
interface Change<T>{
  void mutate();
}
```

B. Design the method `changeAll` for the classes that represent a list of items of the type `<T>` that mutates every value in this list as prescribed by the method `mutate` that is defined to implement the `IChange` interface.

C. In the class `Book` implement the interface `IChange` through a method that increases the prices of all books by 20%.

   Test the method `mutate` in the class `Book`.

Test the method `changeAll`for the classes that represent a list of items of the type `<Book>`.

D. In the class `Song` implement the interface `IChange` through a method that converts the song duration to seconds.

Test the method `mutate` in the class `Song`.

Test the method `changeAll`for the classes that represent a list of items of the type `<Song>`.

E. In the class `Image` implement the interface `IChange` through a method that crops the sizes of all images by 50%.

Test the method `mutate` in the class `Image`.

Test the method `changeAll`for the classes that represent a list of items of the type `<Image>`.

7.3 Problem

Java libraries provide the following interface that can be used to compare arbitrary two items of the same type:

```
// define ordering among items of the type T
interface Comparator<T>{

  // return value < 0 if t1 is before t2
  // return 0 if t1 is the same as t2
  // return value > 0 if t1 is after t2
  int compare(T t1, T t2);
}
```

To use it you need to include `import java.util.*;` at the top of your file.

A. Create a file *BookComparators.java* that will contain `Comparator`s for `Book`s: one that compares them by titles, one that compares them by authors, and one that compares them by price.

Test every `Comparator`.

B. Design the `sort` method for the lists of items of the type `<T>` that consumes an instance of a `Comparator<T>` and produces a list of items sorted in the order given by the `Comparator`.

C. Test your method with all three `Comparator`s defined in the previous part.

D. Add a test that sorts the songs by artists.

7.4 Problem

In the lecture on Thursday we defined the following interface:

```
// represents a function that combines list item of the type T
// with the accumulated value of the type R
// and produces a new accumulated value of the type R
interface IFunTR2R<T, R>{
  R combine(T t, R r);
}
```

and use it to define the method that emulates the *Scheme fold* functions as follows:

```
In the interface ILo<T>:
  // apply the update to combine every item in the list
  // with the accumulated value; start with the base value
  public <R> R fold(IFunT2R<T, R> update, R base);

In the class MtLo<T>:
  public <R> R fold(IFunT2R<T, R> update, R base){
    return base; }

In the class ConsLo<T>:
  public <R> R fold(IFunT2R<T, R> update, R base){
    return
      update.combine(this.getFirst(),
                     this.getRest().fold(update, base)); }
```

A. In the classes `Book`, `Song`, and `Image` design the method `toString` that produces a readable `String` representation of the data that the object represents. For example, you may produce a `String`

```
"Book: The Pearl, written by John Steinbeck, \$20"
```

Note: The Java class `Object` already implements `toString` method, and so every class has the `toString` method defined. The `String` it produces is not very helpful. However, it means that you can invoke the `toString` method on any Java object.

B. In the `Examples` class design the method `listToString` that consumes an `ILo<T>` list and produces a `String` representation of its data, one line per list item, by invoking the `fold` method with the appropriate `update` and `base` parameters.

C. Now design the method `listToStringReverse` that works just as the one above, but produces the list items in the order reversed from the previous one.