10 Using ArrayLists, Traversals, Loops and Function Objects

Goals

In this lab you will first learn how to use and mutate ArrayList objects. We will use the generics (type parameters), but will do so by example, rather than through explanation of the specific details.

In the second part of the lab you will learn how to use Java while statement and Java for statements to implement imperative loops.

Throughout the lab we will work on lists of music albums.

10.1 Class for Albums

Launch Eclipse, start a Project called Lab10 and import the files from lab10.zip.

Take a look at the Album class. You’ll notice that the fields are private and we provide getter methods for the user who wants to access a field outside the class. This way, the user can retrieve the value of a field without changing it.

Task 1:

Design the class BeforeYear that implements the ISelect interface with a method that determines whether the given album was recorded before some fixed year. Remember to test the method.

10.2 Using ArrayList with Mutation

Open the web site that shows the documentation for Java libraries

http://java.sun.com/j2se/1.5.0/docs/api/

Find the documentation for ArrayList.

Here are some of the methods defined in the class ArrayList:

// how many items are in the collection
int size();

// add the given object of the type E at the end of this collection
// false if no space is available
boolean add(E obj);

// return the object of the type E at the given index
E get(int index);
// replace the object of the type E at the given index
// with the given element
// produce the element that was at the given index before this change
E set(int index, E obj);

Other methods of this class are isEmpty (checks whether we have added
any elements to the ArrayList), contains (checks if a given element exists in
the ArrayList — using the equals method), set (mutate the element of the list
at a specific position), size (returns the number of elements added so far).
Notice that, in order to use an ArrayList, we have to add

import java.util.ArrayList;

at the beginning of our class file.
The methods you design here should be added to the Examples class,
together with all the necessary tests.

Task 2:

- Design the method that determines whether the album at the given
  position in the given ArrayList of Albums has the given title.

- Design the method that determines whether the album at the given
  position in the given ArrayList of Albums was recorded before the
given year.

- Design the method that produces a String representation of the album
  at the given position in the album list.

- Design the method that swaps the elements of the given ArrayList at
  the two given positions.

10.3 Converting recursive loops into iterative while loops

We will look together at the first two examples of orMap in the Examples
class.

We first write down the template for the case we already know — the
one where the loop uses the Traversal iterator. As we have done in class,
we start by converting the recursive method into a form that uses the accu-
mulator to keep track of the knowledge we already have, and passes that
information to the next recursive invocation.

Read carefully the Template Analysis and make sure you understand the
meaning of all parts.
TEMPLATE - ANALYSIS:

return-type method-name(Traversal tr){
  +--------------------+
  // invoke the methodAcc: | acc <-- BASE-VALUE |
  +--------------------+
  method-name-acc(Traversal tr, BASE-VALUE);
}

return-type method-name-acc(Traversal tr, return-type acc)
... tr.isEmpty() ... -- boolean ::PREDICATE
if true:
  ... acc -- return-type ::BASE-VALUE
if false:
  +---------------+
  ...| tr.getFirst() | ...
  +---------------+
  ... update(T, return-type) -- return-type ::UPDATE
  i.e.: ...| update(tr.getFirst(), acc) | ...
  +---------------+
  ... tr.getRest() | -- Traversal<T> ::ADVANCE
  +---------------+

Based on this analysis, we can now design a template for the entire problem — with the solution divided into three methods as follows:

COMPLETE METHOD TEMPLATE:

<T> return-type method-name(Traversal<T> tr){
  +------------+
  method-name-acc(Traversal tr, | BASE-VALUE |);
  +------------+
}

<T> return-type method-name(Traversal<T> tr, return-type acc){
  +------------+
  if (| tr.isEmpty() |)
  +------------+
  return acc;
  else
  return method-name-acc( | tr.getRest() |
  +------------+
  | update(tr.getFirst(), acc) |);
  +------------+
}

<T> return-type update(T t, return-type acc){ ...

}
Task 3:

- Look at the first two variants of the orMap method (the recursively defined variant and the variant that uses the while loop. Identify the four parts (BASE-VALUE, Termination/Continuation PREDICATE, UPDATE, and ADVANCE) in each of them.
  Look also at the tests in the Examples class.

- After you understand how the while loop works, design two variants of the method that produces a new ArrayList that contains all elements of the original list that satisfy the given ISelect predicate.
  Test the methods by producing all albums released before the given year.

- Design and test two variants of the andMap method that determines whether all elements of a given list satisfy the given ISelect predicate.
  Test the methods by producing all albums released before the given year.

10.4 Converting while loops into for loops

If you have the time left, repeat all the parts of Task 3 with the remaining two variants of the orMap — namely the one that uses the for loop with the Traversal and the one that uses counted for loop.