9 Javadocs, Raising Exceptions Traversals, Mutating ArrayList

Goals

The first part of the lab introduces a several new ideas:

- Documenting programs in the Javadoc style.
- Generating documentation web pages from a properly documented project.
- Defining and using Java *Exception*s.
- Defining and using functional iterators (*Traversal*s).
- Using classes from Java Libraries.

The second part introduces *ArrayList* class from the **Java Collections Framework** library, lets you practice designing methods that mutate *ArrayList* objects. We will continue to use the generics (type parameters), but will do so by example, rather than through explanation of the specific details.

In the third part of the lab you will learn how to how to convert recursive loops to imperative (mutating) loops using either the **Java** *while* statement ot the **Java** *for* statements to implement the imperative loops.

Lab Materials

Start the lab by downloading the files in *Lab9-fl2007.zip*. The folder contains two folders named **sources** and **interfaces**. The folder **sources** contains the following files:

- *Album.java* that defines a class of music albums
- *BeforeYear.java* that implements the *ISelect* interface in a method that determines whether the given *Album* was released before the given year
- *AlbumListTraversal.java* that defines enables us to access an *ArrayList* of *Album* data through the *Traversal* interface

- *Algorithms.java* that defines several *Traversal*-based algorithms for a general use.
- *Examples.java* that contains our examples of data and the tests for all defined methods.

The folder **interfaces** groups together in the *interfaces* package all interfaces that we will use over several different projects. That means, that later, once we understand how these interfaces are designed, we can distribute them as a pre-compiled compressed *jar* file, just like we distribute the test harness.

It consists of the following four files:

- *Traversal.java* that defines the *Traversal* interface
- *IllegalUseOfTraversal.java* illustrates the definition of an *Exception* class
- *ISelect.java* the interface we have been using to define a predicate for the given object
- *ISame.java* that allows you to define your own equality comparison in a class that implements this interface.

Create a new **Project** *Lab9* and import into it all files from the **sources** folder. Then, import the files from the **interfaces** folder, but identify to import the whole folder, not just the individual files. This will bring in the files as a *package*. Finally, add the *tester.jar* variable — making sure you use the version supplied with this lab.

The previous version included the Traversal and ISame interfaces — we have moved them to a separate package, so that you can see the definitions.

9.1 Documentation, Traversals, Exceptions, Java Libraries

Spend no more than 20 minutes on this part. Do the rest at home.

Generating Documentation

- Once Eclipse shows you that there are no errors in your files select **Generate Javadoc...** from the **Project** pull-down menu. Select to generate docs for all files your project that are in the *default* package with the destination *Lab9/doc* directory.

You should be able to open the *index.html* file in the *Lab9/doc* directory and see the documentation for this project. Compare the documentation for the class *Album* with the web pages. You see that all comments from the source file have been converted to the web document.

Observe the format of the comments, especially the /** at the beginning of the comment. If you do not understand the rules, ask the TA or one of the tutors, or experiment with new comments. From now on all of your work should have a proper Javadoc style documentation.

- Now use the documentation to see what are the fields in various classes and what methods have been defined already.

- Repeat the above for the *interfaces* package, but select a different directory where to store the documentation. Look at the documentation to see how these interfaces are defined.

Defining and Handling Exceptions

- The file *IllegalUseOfTraversal.java* illustrates the definition of an *Exception* class.

  The class *Traversal* shows you the headers for the methods that throw an *Exception*.

  The class *AlbumListTraversal* illustrates how the user designs the method that throws an exception and reports on the discovered problem.

  Look at the code briefly, but mainly use it as a reference for your programs later on.

9.2 Using ArrayList with Mutation

Spend no more than 25 minutes on this part. Do the rest at home.

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:

- Design the method that determines whether the album at the given position in the given *ArrayList* of *Album*s has the given title.
- Design the method that swaps the elements of the given *ArrayList* at the two given positions.

9.3 Converting Recursive Loops into Imperative while Loops

Allow at least 35 minutes on this part. Do the rest at home.

In this part we will see several different ways to implement loops in Java. The given code in the *Algorithms* class illustrates how the different variants of loops are designed. Your task will be to design for each variant of the loop the method *filter* that produces an *ArrayList* of all items that satisfy the given predicate.

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

We first write down the template for the case we already know — the one where the loop uses the *Traversal* iterator. We start by converting the recursive method into a form that uses the accumulator 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() | ...                    -- E           ::CURRENT
    +---------------+

 ... update(T, return-type)                  -- return-type ::UPDATE
          +---------------------------+
 i.e.: ...| update(tr.getFirst(), acc) | ...
          +---------------------------+
     +--------------+
 ... | tr.getRest() |                        -- Traversal<T> ::ADVANCE
     +--------------+

 ... method-name(tr.getRest(), return-type)  -- return-type
 i.e.: ... method-name-acc(tr.getRest(), update(tr.getFirst(), acc))
```

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 *filter* 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.

- *Optional — skip and finish at home, if necessary*

  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.

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.

Portfolio

At home, add to your lab project a new class that represents data of your choice, design an *ISelect* predicate for that class of data, and test all methods on *ArrayList*s of your data.