9 Mutating a Structure
Raising and Catching Exceptions
Traversals
Javadocs

Goals

In the first part of this lab you will learn to design methods that change the contents of a compound object.

In the second part of the lab you will start using the type parameters (Java generics).

The third part 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.

9.1 Designing State Change

For this part download the files in *Lab9-Part1-Student.zip*. The folder contains the files *Balloon.java*, *TopThree.java*, and *Examples.java*.

Starting with partially defined classes and examples will give you the opportunity to focus on the new material and eliminate typing in what you already know. However, make sure you understand how the class is defined, what does the data represent, and how the examples were constructed.

Create a new **Project** *Lab9-Part1* and import into it the files files *Balloon.java*, *TopThree.java*, and *Examples.java*. Also import the *TestHarness.jar* files and *jpt.jar* from the previous lab.

The class *Balloon* represents a balloon at some location, with some radius and color. The method *inflate* allows us to change the radius of the Balloon.

The class *TopThree* represents three balloons. The goal is to have the three balloons ordered, so that *the best one* is the value of the field *one*, *the second best* is the value of the field *two*, and *the third best* it the value of the field *three*.

The ordering is determined by the behavior of the *Comparator* instance that is a field of the *TopThree* object.

- Design tests that show that with the present class definitions it is possible to violate the above specification of the class *TopThree*.
- Design the method *reorder* that rearranges the three values of the *Balloon* objects, so they are ordered as described above.
- Modify the constructor for the class *TopThree* so that the three values of *Balloon*s will always be arranged in the order specified above.
- Design the method *inflateBalloons* in the class *TopThree* that inflates each of the three balloons by the given amount. (The the three balloons need not be inflated by the same amount.) Make sure that the *TopThree* object satisfies the specification after the balloons have been inflated.
- **Etude:** Design the method *floatBalloon* in the class *Balloon* that moves the balloon up by the given amount. Then design the method *floatBalloons* in the class *TopThree* that floats all three balloons and restores the order among them ofter the floting has been completed.

9.2 Java Generics: Using Type parameters

For this part download the files in *Lab9-Part2-Student.zip*. The folder contains the files *Balloon.java*, *ImageFile.java*, *ISelect.java*, *IChange.java*, *TopThree.java*, and *Examples.java*.

Create a new **Project** *Lab9-Part2* and import into it the files *Balloon.java*, *ImageFile.java*, *ISelect.java*, *IChange.java*, *TopThree.java*, and *Examples.java*. Again, import the test harness files and *jpt.jar*.

Read briefly all files and compare them to those used in *Part 1*. Add to the class *TopThree* the method *reorder* and change the constructor as you did in the *Part 1*.

Notice that we have replaced the methods that change the state of one object (either a *Balloon* or a *ImageFile*) by a method *change* and have the classes implement a common *IChange* interface.

- Look at the file *ImageFile.java*. At the end of the file is a definition of the class *ImageFileByName* that implements the *Comparator* interface. However, this time we are using the type parameter that guarantees that the two objects being compared are of the same type.

  Design the class *ImageFileBySize* that implements the *Comparator* interface by comparing two image files by their size. Use the type parameter to simplify the definition.

- Design the tests for the method *change* in the class *TopThree* that uses image files and the changes in the image size for the test cases.

- Rewrite the *Comparator* for the class *Balloon* so that it uses the type parameters and rerun the tests that use it.

- **Etude** Add a *Comparator* for the class *Balloon* that compares two *Balloon*s by their height and run all tests, including those for the class *TopThree*.

9.3 Documentation, Traversals, Exceptions, Java Libraries

For this part download the files in *Lab9-Part3-Student.zip*. The folder contains the files *Balloon.java*, *ImageFile.java*, *ISelect.java*, *IChange.java*, *TopThree.java*, and *Examples.java*. In addition, there are several new files: The file *Traversal.java* defines the *Traversal* interface, the files *AList.java*, *MTList.java*, and *ConsList.java* that define a generic cons-list that implements the *Traversal* interface. The file *IllegalUseOfTraversal.java* illustrates the definition of an *Exception* class. Finally, the *Algorithms.java* file shows an implementation of an algorithm that consumes data generated by a *Traversal* iterator.

Create a new **Project** *Lab9-Part3* and import into it all files from the zip file. Again, import the test harness files and *jpt.jar*.

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 in your project with the destination *Lab9-part3/doc* directory.

  You should be able to open the *index.html* file in the *Lab9-part3/doc* directory and see the doculemtation for this project. Compare the documentation for the class *ConsList* 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.

Defining and Handling Exceptions

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

  The files *AList.java*, *MTList.java*, and *ConsList.java* illustrate how methods can *throw* exceptions when something goes wrong.

  The method *contains* in the class *Algorithms* illustrates how the methods that *throw* exceptions are invoked.

  Add tests for the method *contains* to the *Examples* class.

- Add to the *Examples* class a test that will cause the exception to be raised and observe the consequences. Once you have seen the result, comment out this testcode.

- Add to the class *Algorithms* a method *filter*. The header for the method is already provided.

ArrayList and Java Libraries

- The class *TopThree* now stores the values of the three elements in an *ArrayList*. Complete the definition of the *reorder* method. Use the previous two parts as a model. Look up the documentation for the Java class *ArrayList* to understand what methods you can use.