8 Designing Tests for State Change Abstracting with Function Objects Introducing Type Parameters

Goals

In the first part of this lab you will learn how to correctly design tests for the methods that change the state of an object.

In the second part of the lab you will learn to abstract over the functional behavior.

In the third part you will get the first introduction to the abstraction over the type of data using type parameters.

8.1 Designing Tests for State Change

For this part download the files in *Lab8Sp2007-s1.zip*. The folder contains the files *ImageFile.java*, *ISame.java*, *ISelect.java*, *SmallImageFile.java*, *AList.java*, *MTList.java*, *ConsList.java*, and *Examples.java*, as well as *TestHarness.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** *Lab8Part1* and import into it all of the given files. Also import *jpt.jar* from the previous lab.

- Design the method *crop* that changes the dimensions of an *ImageFile* object to the given *width* and *height*. The *Examples* class contains comments on what needs to be done to design the tests. Follow the outline given by the comments to design the needed tests.

- Design the method *changeName* that allows us to change the name field of an *ImageFile* object. Design the tests.

8.2 Abstracting with Function Objects

We will now practice the use of *function objects*. The only purpose for defining the class *SmallImageFile* is to implement one method that determines whether the given *ImageFile* object has the desired property. An instance of this class can then be used as an argument to a method that deals with *ImageFile*s.

1. In the *Examples* class design the tests for the class *SmallImageFile*.

2. Design the method *allSmallerThan40000* that determines whether all items in a list are smaller that 40000 pixels. The method should take an instance of the class *SmallImageFile* as an argument.

3. Design the class *NameShorterThan4* that implements the *ISelect* interface with a method that determines whether the name in the given *ImageFile* object is shorter than 4.

   Make sure in the class *Examples* you define an instance of this class and test the method.

4. Design the method *allNamesShorterThan4* that determines whether all items in a list have a name that is shorter than 4 characters. The method should take an instance of the class *NameShorterThan4* as an argument.

5. Design the method *allSuch* that that determines whether all items in a list satisfy the predicate defined by the *select* method of a given instance of the type *ISelect*. In the *Examples* class test this method by abstracting over the method *allSmallerThan40000* and the method *allNamesShorterThan4*.

6. For the first etude, at home, follow the same steps as above to design the method *anySuch* that that determines whether there is an item a list that satisfies the predicate defined by the *select* method of a given instance of the type *ISelect*.

8.3 Abstracting over the Datatype: Generics

For this part download the files in *Lab8Sp2007-s2.zip*. The folder contains the files *ImageFile.java*, *ISame.java*, *ISelect.java*, *SmallImageFile.java*, *AList.java*, *MTList.java*, *ConsList.java*, and *Examples.java*, as well as *TestHarness.java*.

Create a new *Project Lab8Part2* and import into it all the given files. Again, use *jpt.jar* from the previous lab.

Look at the files. The interface *ISelect* now includes a *type parameter T*:

```
public interface ISelect<T> {
  /* Return true if this Object of the type T should be selected */
  public boolean select(T o);
}
```

That means that the implementing class can decide what type of data should be used as the argument to the *select* method. This greatly simplifies the class *SmallImageFile*:

```
/* Select image files smaller than 40000 */
public class SmallImageFile implements ISelect<ImageFile> {

  /* Return true if the size of the given ImageFile is smaller than 40000 */
  public boolean select(ImageFile o) {
    return o.height * o.width < 40000;
  }
}
```

Similarly, the *ISame* interface also provides a type parameter for its argument:

```
interface ISame<T>{
  // is this object the same as the given one?
  boolean same(T that);
}
```

The implementation of the *same* method in the class *ImageFile* is then greatly simplified:

```
  // is this imagefile the same as the given one?
  public boolean same(ImageFile that){
    return this.name.equals(that.name) &&
           this.width == that.width &&
           this.height == that.height &&
           this.kind.equals(that.kind);
  }
```

Look at the class definition for the class *ImageFile* to see the use of the type parameter there.

Moreover, the classes that represent a list of arbitrary items can now specify the type of items that can be included in the list construction. However, because the class implements a parametrized interface *ISame*, the use of type parameters is quite complicated. For now, you can just use the implementation and do not have to follow every detail of the use of the type parametrs there.

Re-do all of the problems from the previous part, but using the type parameters

1. In the *Examples* class design the tests for the class *SmallImageFile*, just as you did before.

2. Design the method *allSmallerThan40000* that determines whether all items in a list are smaller that 40000 pixels. The method should take an instance of the class *SmallImageFile* as an argument.

3. Design the class *NameShorterThan4* that implements the *ISelect*<*ImageFile*> interface with a method that determines whether the name in the given *ImageFile* object is shorter than 4.

   Make sure in the class *Examples* you define an instance of this class and test the method.

4. Design the method *allNamesShorterThan4* that determines whether all items in a list have a name that is shorter than 4 characters. The method should take an instance of the class *NameShorterThan4* as an argument.

5. Design the method *allSuch*<*T*> that that determines whether all items in a list (of items of the type *T*) satisfy the predicate defined by the *select* method of a given instance of the type *ISelect*<*T*>. In the *Examples* class test this method by abstracting over the method *allSmallerThan40000* and the method *allNamesShorterThan4*.

6. For the second etude, at home, follow the same steps as above to design the method *anySuch* that that determines whether there is an item a list that satisfies the predicate defined by the *select* method of a given instance of the type *ISelect*.