6 Abstracting with Function Objects

Goals
In this lab you will learn how to abstract over the functional behavior.

6.1 Abstracting with Function Objects


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 Lab6-sp10 and import into it all of the given files. Also import tester.jar from the previous lab.

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 (a predicate method). An instance of this class can then be used as an argument to a method that deals with ImageFiles.

1. Start with defining in the ExamplesImageFile class the missing tests for the class SmallImageFile.

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

3. We now want to determine whether the name in the given ImageFile object is shorter than 4. Design the class NameShorterThan4 that implements the ISelectImageFile interface with an appropriate predicate method.

   Make sure in the class ExamplesImageFile 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.
method should take an instance of the class `NameShorterThan4` as an argument.

5. Design the method `allSuchImageFile` that determines whether all items in a list satisfy the predicate defined by the `select` method of a given instance of the type `ISelectImageFile`. In the `ExamplesImageFile` class test this method by abstracting over the method `allSmallerThan40000` and the method `allNamesShorterThan4`.

6. Design the class `GivenKind` that implements the `ISelectImageFile` interface with a method that produces `true` for all `ImageFile`s that are of the given kind. The desired kind is given as a parameter to the constructor, and so is specified when a new instance of the class `GivenKind` is created.

   **Hint:** Add a field to represent the desired kind to the class `GivenKind`.

7. In the `ExamplesImageFile` class use the method `allSuch` and the class `GivenKind` to determine whether all files in a list are `jpg` files. This should be written as a test case for the method `allSuchImageFile`.

   Do it again, but now ask about the `giff` files.

8. If you have some time left, design the method `filterImageFile` that produces a list of all `ImageFile`s that satisfy the `ISelectImageFile` predicate. Test it with as many of your predicates as you can.

9. Follow the same steps as above to design the method `anySuchImageFile` that determines whether there is an item in a list that satisfies the predicate defined by the `select` method of a given instance of the type `ISelectImageFile`.

10. Finish the work at home and save it in your portfolio.

    **Food for thought:** Think how this program would be different if we have instead worked with lists of `Books`, or lists of `Shapes`. 