COMP241-07A

Software Engineering Development

Assignment 2

1 Assignment Goals

The goal of this assignment is to study object-oriented design principles and test-first design by applying them to the implementation of the ImageOrganizer specification. The task is to study the ImageOrganizer specification and provide your own implementation of all the required functionality.

2 The Task

This assignment may be done in pairs or individually as the was the case with the first assignment. Your task is to implement all the functionality required by the ImageOrganizer specification. You have written some test cases for the first assignment; you should write test cases for the remaining functionality.

With one exception (see below) you will not have to modify any code in the imageorg package. Instead, you will create your own classes that implement the interfaces and and place them in the implementation directory (delete or move my reference implementation first). Your code will be packaged in the `implementation` package (i.e. each of your classes will have the package statement: `package implementation;`). You may name your classes however you like—it does not matter to me as I will be testing your implementation polymorphically using only the interfaces in `imageorg`. You are free to add additional methods (not defined in the `imageorg` interfaces) if it makes the job of providing the required functionality easier. Just remember that I will have no knowledge of these methods because I will only be using the `imageorg` interfaces to operate your code.

There is one case where you will have to modify code in the imageorg package. `imageorg.ImageOrganizer` contains a static factory method called `create()` that is designed to return an `ImageOrganizer` object which is a concrete subclass of `ImageOrganizer`. You will have to replace the single line in this method which returns null with one that returns a concrete `ImageOrganizer`. For example, with assignment 1 I provided a concrete subclass of `ImageOrganizer` called `implementation.MarksImageOrganizer`. In the `imageorg.ImageOrganizer` source code I added the following line to the static `create()` method:

```
return new implementation.MarksImageOrganizer();
```

You will need to replace this line with the class name of your concrete implementation of `ImageOrganizer`.

A good design would probably have an abstract class that implements `FileElement` and provides as much general purpose, reusable code as possible. Each subtype of this abstract class would implement any left over abstract methods as well as one of the sub-interfaces (`Folder` or `Image`).

3 Concerning Files

To complete this assignment you will need to make use of classes in the `java.io` package. IO in Java (including object serialization) will be covered later in the course. In the meantime you can read the javadocs for the `java.io.File` class, which contains methods that will be useful for providing the functionality required by the image organizer library. In order to ensure platform independence when dealing with paths, you should always use path separator constants provided in the `File` class. **I strongly suggest that you test your program on a lab machine as I will be using Linux when I mark the assignments.**

The `imageorg.ImageOrganizer` abstract class declares a constant which holds the path to the home (or root) of the image organizer's repository. This is defined—in a platform independent way—to be a directory called `image_org_home` in the user's "home" directory. The image organizer should only allow images (jpegs and gifs) and folders to be stored in the repository. The class `imageviewer.ImageViewer` contains a static method that loads gif and jpeg images. You can use this method to determine whether or not a file is an image.

Since you have to associate annotation data with images and folders, it is necessary to store this data somewhere. Object serialization makes this easy because you can save an entire object (including all objects it references) to a binary file. Therefore, any internal data structure you use to keep track of annotations can be easily saved. If you take this approach, you should save such data to a special file outside of the image organizer's repository. Under Unix like environments it is common practice to save configuration data to a file that begins with a '.' in the user's "home" directory. The `ImageOrganizer` interface has a method called `save()`. When this method is called by a client, your program should save any internal data structures. The specification of the `save()` method states that any serialized data structures must be saved to a file called `.imageorg` in the user's home directory.

Your program should be robust. You need to handle situations such as:

- The image organizer repository (or parts of it) being deleted.
- Any data stored as serialized objects being deleted.
- Inconsistencies between serialized data structures and actual disk files come about through files being manually removed or added to the repository.

However, for this assignment you can assume that the image organizer's files will be free from external tampering during a session of usage—that is, from the point where a client program calls `imageorg.ImageOrganizer.create()` to the point where they call the `save()` method in `imageorg.ImageOrganizer`.

This assignment may be completed individually or in pairs.

4 Deliverables

Your complete submission should include:

- The well written and documented source code of your implementation.
- An ANT build script with targets to compile and run tests from the sources.

Please create an archive (`.zip or .tar.gz`) containing all your files and send it by e-Mail to the address `mhall@cs.waikato.ac.nz`. Please indicate clearly (in a README) the names and ID numbers of each person involved in the assignment.

Due date: Friday, 4th May 2007: 5 P.M