# CS580 Advanced Software Engineering
# Assignment 5 - Photo Filters & Be Social!

## Due Date

Wednesday, November 5, 2014

## Score

30

## Questions and Directions

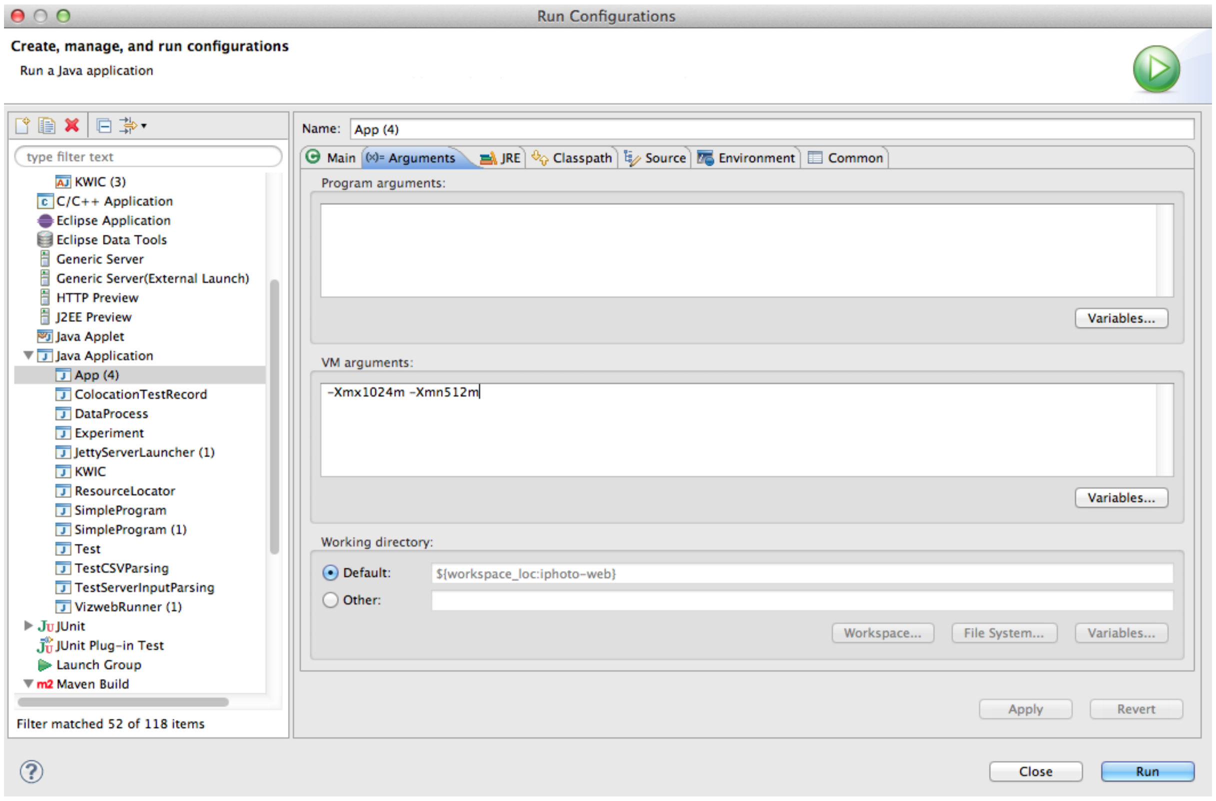
In this assignment, you will be adding two important features for iphoto-web: 1) *photo filters* - allows users to apply different image filters to produce different effects on your uploaded photos; 2) *be social* - allows users to follow or unfollow other users and see the news feed about all the activities from the followed users.

The goal of this assignment is to apply design patterns to build your functional modules in the clean and extensible way.

### *1. Photo Filters*

You will use the image processing library provided by ImageJ (http://imagej.nih.gov/ij/) to handle the different filtering effects. Please find the sample code to test the filtering below. Make sure you have the ImageJ dependency configured correctly in Maven (You should have already done it in Assignment 3).

In addition, you might need to configure the JVM heap size in your Eclipse when running the program (as well as our web `App.java`) in order to avoid the `OutOfMemoryException`. See the instructions in the comments, as well as the screenshot below on how to configure it.



Name: App (4)

Program arguments:

VM arguments: -Xmx1024m -Xmn512m

Working directory:
Default: ${workspace_loc:iphoto-web}
Other:

"Run As" -> "Run Configurations…" -> "Arguments" -> "VM Arguments" -> "-Xmx1024m -Xmn512m"

Figure 1. Increase the JVM heap size when using ImageJ

```
import ij.IJ;
import ij.ImagePlus;
import ij.process.ImageProcessor;

import java.io.File;

/**
 * This is a test class to demonstrate how to use ImageJ
 * to trigger image filter effects on photo files.
 *
 * You need to configure the ImageJ dependency in your Maven
 * configuration.
 *
 * Note:
 * Using ImageJ to do image processing needs a lot of memory,
 * so you might see the OutOfMemeory exception when running
 * this program. Thus, you will need to configure the JVM
 * parameter when running it:
 *     -Xmx1024m -Xmn512m
 *
 * When running your program in Eclipse, please use:
 * "Run As" -> "Run Configurations..."
 * In "Arguments" tab, put the parameters in "VM arguments".
 *
 */
public class TestJImage {

    public static void main(String[] args) {
        // Locate the new file and access it using standard Java File
        File imageFile = new File("/Users/yusun/Desktop/front.jpg");
        // Initialize ImagePlus object with the input image file
        ImagePlus image = new ImagePlus(imageFile.getAbsolutePath());


        // Initialize the image processor
        ImageProcessor ip = image.getProcessor();
        // The image processor provides a number of image filters to use
        // the following line applies the filter to
        ip.findEdges();
        // Some other filters to use can be found at:
        // http://rsb.info.nih.gov/ij/developer/api/ij/process/ImageProcessor.html
        // e.g., ip.invert();
        //       ip.rotate(30);


        // Instead of overwriting the input file, we create a new file
        // to save the filtered photo
        File newImageFile = new File("/Users/yusun/Desktop/front-edge.jpg");
        // We use ImageJ utility to save the file
        IJ.save(image, newImageFile.getAbsolutePath());
    }
}
```

Knowing how to use ImageJ to process photo files, you can integrate this feature to iphoto-web. Specifically, you need to implement the following two methods in `WebController.java`.

## 1.1 Get Available Filters

```
    @RequestMapping(value = "/iphoto/filters/list", method = RequestMethod.GET)
    List<String> getAvailableFilters() {

    }
```

This methods returns a list of Strings to represent the available filters to choose. You should provide some meaningful names such as “Edge”, “Invert”, etc. depending on the filters you would like to use. **Note, you should provide at least 3 types of filter effects.**

After implementing this method, you should be able to test it in your home page:
http://localhost:8080/iphoto/user1/home

Above each of the uploaded photos, you should be able to see a list of checkboxes to allow you choose the filters to apply.

## 1.2 Apply Filters

```
@RequestMapping(value = "/iphoto/{userId}/photo/{photoId}/filter/{filters}",
        method = RequestMethod.GET)
void getFilteredPhoto(
        @PathVariable("userId") String userId,
        @PathVariable("photoId") String photoId,
        @PathVariable("filters") String filters,
        HttpServletResponse response) {

}
```

This method takes `userId`, `photoId`, and `filters` as inputs, and produce the filtered photo as the output.

For `filters`, please note that **multiple filters are supported**. If you want to apply more than one filters, you should use "-" to join the filter names together. For instance, if you want to apply "edge" and "invert", the input value for filters will be "edge-invert".

In our test page (http://localhost:8080/iphoto/user1/home), when you click on multiple checkboxes, and then click on Apply, it will produce the right `filters` format with "-" automatically, and call this method. Therefore, you will use this test page to test your filter function, without having to manually input the filter names by yourself.

You need to make sure this method does the following things:

- When applying each filter, you will NOT overwrite the old file. Instead, you should create a new file for the filtered photo. For example, if we have the photo file myphoto1.jpg, and we want to apply two filters "edge-invert". After executing this method, you should create two more files in the user's folder: myphoto1-edge.jpg and myphoto1-edge-invert.jpg. In other words, you will do the filter effect iteratively one by one, and generate a new file each time (the order does not matter).
- When the multiple filters are passed, you need to split the filters and handle them one by one.
- This method should also return the final filtered photo as an output stream as we did last time, with the same code below:

```
// 1. Get the final filtered photo file
File finalPhoto = ...

// 2. Return the photo file as an output stream using the code below
IOUtils.copy(new FileInputStream(finalPhoto), response.getOutputStream());
response.setContentType("image/jpeg");
```

```
        response.flushBuffer();
```

**Of course, the most important thing is to figure out which design pattern to use to implement this feature, so that your code looks cleaner and easy to change.**

## *2. Be Social*

Just like most of other websites, we also want to implement some simple social news feed function for iphoto-web.

Please take a look at `User.java` class, and we are going to use this class to record the users that you are following, the users that follow you, and your news feed information list.

Also, you are going to extend the `UserManager.java` and `FSUserManager.java` to implement other functions needed for this feature.

Note, the `updateUser()` method in `FSUserManager` persists the user information in your disk as a file so that the information will not be lost after restarting the server. You should always call this method to whenever you change the user related information. The method uses JSON mapper to save object as the JSON format (http://json.org/).

You need to implement the following 2 methods in WebController.java:

### 2.1 Follow a User

```
@RequestMapping(value = "/iphoto/{userId}/follow/{followedUserId}",
        method = RequestMethod.POST)
String followUser(
        @PathVariable("userId") String userId,
        @PathVariable("followedUserId") String followedUserId) {

    return "OK";
}
```

This method basically allows `userId` to follow `followedUserId`. You simply return “OK” as long as you can record the correct information to the user object.

### 2.2 Unfollow a User

```
@RequestMapping(value = "/iphoto/{userId}/unfollow/{followedUserId}",
       method = RequestMethod.POST)
String unfollowUser(
        @PathVariable("userId") String userId,
        @PathVariable("followedUserId") String followedUserId) {
```

```
        return "OK";
    }
```

This methods allows `userId` to remove `followedUserId` from his or her followed list.

To test 2.1 and 2.2, simply go to the test page (http://localhost:8080/iphoto/user1/home) to add or remove users by userId.

The test page can automatically show all the user information including the news feed. **However, you need to implement how to update the news feed based on the following/followed relationship.**

## 2.3 Update News Feed

Basically, whenever a user **adds a photo/deletes a photo/applies certain filters**, you should generate a String as the news feed message for the activity, and notify all the users that are following this user (update the followers' news feed).

When a user A stops following another user B, A will no longer get any updates in his or her news feed.

We only want to add news feed. You don't have to worry about deleting old news feed if a user stops following.

Again, you can use the test page to test everything. **And use design patterns!**

Have fun!