CSCI-1302

Software Design
Spring 2012

Project 1

Green Screen!
Revised 1/19/12

Learning Outcomes:
By completing this programming assignment, you should be able to do the following:

1) process Java main input arguments
2) validate and respond to user input
3) look up new concepts in the Java API
4) implement iteration

Introduction

Ah, the life of being a teaching assistant at UGA. It's a rewarding experience, but lacks the perks of some more interesting jobs. The goal of this assignment is to help our illustrious head TA Sagar feel the experience of traveling without him spending a dime!

To do this, we're going to implement some trickery using Java. If you've ever watched a weather forecast, you've seen some trickery in action, referred to commonly as a greenscreen. The weather guesser stands in front of a large panel of green, and then software magically replaces the green space with images of satellite imager, radar, etc. for viewers at home. Wait, it's not magic, it's science! Computer science!

In this project, you will use Java to build a simple algorithm that you might see being used at a television station, or even in the film special effects industry. You will write a program that processes an image specified by the user, replacing the background with the background of another image, both of which are specified by the user. Finally, you will save the image to a filename as specified by the user.

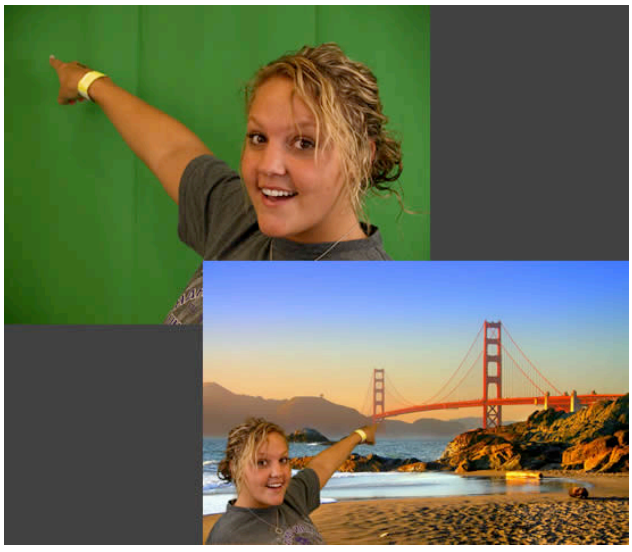

Figure 1. Example of a greenscreen in action (source: http://www.gr8pix.biz/green-screen-service.php)

This lovely person is standing in front of a greenscreen where the image is replaced with a terrific shot of the Golden Gate Bridge.

Note, her positioning is mirrored; in this project, you aren't required to mirror an image.

Here are detailed instructions:

Part I: Swapping Colors

1. The program's main class must be called GreenScreen.java
2. The main method of GreenScreen.java must handle these input parameters (in this particular order):
   `args[0]` (required) is the name of the .png image that you want to manipulate (input image 1)
   `args[1]` (required) is the name of the .png image that you want to
   `args[2]` (required) is the name of file that you will save with any modifications
   `args[3]` (required) is the image filetype you will save (either "jpg" or "png")
   `args[4]` (required) is the color of the green screen *

   * Args 3: the color of the green screen can be either **green**, or **white**, as defined by the color constants in the Java API. For extra credit, you can implement a third option, "auto", which automatically selects the background color.

3. Implement as many helper functions as needed, but you should use at least two helper functions: one to read in the image and perform the manipulations, and one to save the file of the manipulated image.
4. Write your Java source code in emacs or vi on Nike.

Here is a sample of what a successful run of the program might look like:

```
cplaue% java GreenScreen sagar.png golden_gate.png modified png green
cplaue%
```

Note that the name of the output file "mod" does not contain the ".png" extension; since you are specifying the type of image file, you know what extension to automatically tag on! (that is, the actual file should be saved as mod.png).

Input File: sagar.png

Input File: golden_gate.png

Output File: modified.png

I will provide the sagar.png and golden_gate.png files on eLC that you may use as testers, each is a resolution of 430x300. Feel free to come up with your own files to test using *tasteful* backgrounds!

You may be a bit worried at this point, wondering how the heck you'll do this. Well, we structure this assignment to help you—the key is to get started early and ask questions. **Half of being a computer scientist is not necessarily knowing the solution off the top of your head, but knowing where to look and what questions to ask.**

Here are some hints to help you out.

- Take a look in the Java API at the class Color. For the intents of this assignment, we will be using colors in `args[4]` (valid colors specified earlier) as specified by the constants listed there.
- You will need to use `try/catch` blocks for anything involving File I/O
- There are many different ways to deal with Images. My suggestion is to take a look at the very helpful class is `BufferedImage`
- Remember, the Java API and the Java Tutorials are valid resources to consult for this project.

Part II: Error Handling

Add error and exception handling to GreenScreen.java. Specifically, your program must:

- identify user input errors, and then print a meaningful error message identifying at least one of the errors and a possible solution (i.e. If someone inputs "bmp" as the file type to save as, you should print a message stating that the file type must be saved as a jpg or png).
- If an input parameter is missing or entered out-of-order, your program must detect this and print out an appropriate error message.
- The program must terminate correctly after printing out an error message or successfully processing an image. It is up to you to figure out what types of input errors and exceptions can occur and how to correctly handle those errors and exceptions without having your program crash, get stuck in a infinite loop, or have any other undesirable behavior.
- Capitalization should not matter for specifying colors.
- Your program should print out an error if the source and destination files are not the same dimension (i.e. 200 x 200 pixels, each).

Part III: Submission Instructions

As discussed in class, this is a course to not only improve your skills as Java programmers, but also give you experience in developing good software design habits. We have not formally taught design specifications, but we still want you to reflect upon what you functions, libraries, and knowledge you'll need to build the program listed in Part 1.

Prepare a 1-page document that outlines just that: the functions you think you'll need to write and the associated knowledge you'll need to implement these classes (libraries, math properties, etc). For this project, we recommend you use an outline format. USE PLAIN ENGLISH TO DESCRIBE WHAT YOU THINK YOU NEED TO DO. That's all. Seriously.

You should work on this outline and submit it *at least one week before* you implement the code for the project. Save this outline with your name, ID #, and "Project 1 Design Requirements" listed in the upper right-hand corner of the document. Save it as a PDF labeled lastnameP1.pdf and upload it to this assignment in eLC.

How to submit your source files to Nike

1. Create a folder in your Nike account called **lastname_proj1** where lastname is your last name.
2. Copy all **thoroughly commented** Java source files into **lastname_proj1**.
3. Create a makefile and place it in **lastname_proj1** that has three directives:
   a. compile: compiles all of the source code
   b. run: runs an example of your program
   c. clean: removes all class files
4. Add a readme file to **lastname_proj1** which has your name and clear instructions on how to compile and run your program.
5. Also add a .txt file to **lastname_proj1** which critiques your initial design document you previously submitted. Specifically discuss if you needed to implement things in a different manner, use different libraries, or make any compromises. Make sure this document also has your name.
5. Remove all class files before submitting.

6. Navigate to the parent directory of **lastname_proj1** on Nike, and issue the command below
   submit lastname_proj1 cs1302a
7. If the submission was successful, then you should see a file that begins with rec in your **lastname_proj1** folder.

Late Penalty:

This project is due by the 11PM on the date listed on eLC. There is a late penalty of 12% off the original amount of points for each 24-hour period after the due date for a maximum of three days.

Points:

This project is worth 50 points to your course grade. Grading of this programming project will use the appropriate rubric:

Proper documentation (pre & post statements, commenting conditionals, not excessive commenting)	5 points
Design Requirements & Reflection Documents Includes thoroughness of the design requirements and planning of the code, as well as the reflection of the initial design specifications after the code was implemented	10 points
Part I: Swapping Colors Passing test cases	20 points
Part II: Exception Handling Properly handling interesting data	15 points

Note, this grading rubric is to give you a rough idea of how we plan on evaluating your projects.

The extra credit part of this assignment is worth up to 10-extra points.

Collaboration Policy:

This project is to be worked on independently. You may not discuss code with any of your classmates. You may post conceptual questions on Piazza, use your textbooks, the Java API, and the Java Tutorials.

PS:

If any of the "foreground" contains the background color, you can make it seem like a person has a hole in him or her:

http://www.maniacworld.com/weatherman-with-a-green-tie.jpg