[**Department of Computer Science**](http://www.cs.umd.edu/)

[**CMSC131:**](http://www.cs.umd.edu/class/fall2017/cmsc131-FC01/index.shtml) Fall 2017

**Project:** Photo Processing System

**Due Date:**Thu Oct 26, 8:00 pm / Tue Oct 31, 8:00 pm

## Overview

For this project you will implement two classes. The first one called **Address** represents a person's address. The second provides access to the functionality associated with an image processing system. The image processing system will allow you to perform several image processing tasks (e.g., turning an image into black and white). Additional information about this classes is provided below.

### Closed Project

This project is considered a **closed** project. That means you must implement the project by yourself and only with the assistance of TAs and instructors.

### Deadlines

There are two deadlines associated with the project. Those deadlines are:

**Thu Oct 26, 8:00 PM**

Your code must pass the first 5 public tests. That is the only requirement for this deadline. We will not grade the code for style. This first part is worth .5% of your course grade (NOT .5% of this project grade). Notice you can still submit late for this part.

**Tue Oct 31, 8:00 PM**

Final deadline for the project. Notice you can still submit late (as usual).

Keep in mind you can release-test your project even if you have not passed all the public tests. Information about public/secret/release tests can be found at [tests](http://www.cs.umd.edu/~nelson/classes/utilities/TestTypes.html).

## Objectives

To practice writing classes, using **this**, exceptions, StringBuffer and parsing strings.

## Academic Integrity

As the complexity of class projects increases some students decide to violate academic integrity rules. We want to remind you that we have software that allow us to compare your projects against all students in cmsc131 (across all sections). Students found responsible for academic integrity violations will be reported to the Office of Student conduct. The Computer Science Department takes academic integrity matters seriously.

## Grading

* (50%) Public Tests
* (20%) Release Tests
* (20%) Secret Tests
* (10%) Style

## Code Distribution

The project's code distribution is available by [checking out the project](http://www.cs.umd.edu/eclipse/manage.html) named **PhotoProcessing**. The code distribution provides you with the following:

* A file named **Address.java** - This is a class you need to implement.
* A file named **PhotoProcessingSys.java** - This is a class you need to implement.
* A file named **PublicTests.java** - This represents the public tests for the project.
* A file named **PictureManager.java** - We have implemented this class for you. Do not modify it. It provides all the image processing support you need. You may want to run the main() method in that class for a demo of the image processing options.

**You can ignore the classes that are present in the cmsc131PictureLib package. You can complete this project without knowing what they do. Just look at the classes in the sysutilities and tests packages.**

## Specifications

### PictureManager Class

We have implemented this class for you. This class provides several image processing options. You need to use methods of this class during the implementation of the PhotoProcessingSys class. A description of the methods follows.

1. **displayPicture -** Displays an image. The image could be in the Eclipse project or it could be a url that corresponds to an image on the web.
2. **clearScreen -** Removes any image that have been displayed.
3. **displayLastPicture -** Displays the last image that was displayed.
4. **displayPictureBlackWhitePosterize -** Takes an image and turns into black and white, posterizes it or both.
5. **displayPictureSelectRedGreenBlue -** Takes an image and select the red, green and blue components.
6. **graphicalModeOff/graphicModeOn -** In order to run public/secret/release tests we need to disable the image processing (e.g., displaying images). When graphicalModeOff is called only a string will be returned after calling any image processing function.

All the methods described above (except graphicalModeOff/graphicalModeOn) return a string representing the processing that took place.

### Address Class

1. **Constructors with four parameters -**  The parameters are the street, city, state, and zip code. The names of the parameters must be the same names used for the corresponding instance variables. The constructor will throw an IllegalArgumentException exception with the message "Invalid Address Argument" if any parameter is null, or if the zip code has characters others than digits. Notice that any of the parameters can have spaces surrounding them (e.g., "   MD "), but you must get rid of them before using the values to initialize the object. The method Character.isDigit can help during the implementation of this method. See the Java API (Character class) for additional information.
2. **Default Constructor -**  Initializes the object with "8223 Paint Branch Dr.", "College Park", "MD" and "20742" for street, city, state and zip code, respectively. The names of the parameters must be the same names used for the corresponding instance variables. You must implement this constructor using "this".
3. **Copy Constructor**
4. **Constructor with one parameter (street) -**  It will initialize the city, state, and zip code with the default values defined above. Notice the street parameter can have spaces surrounding it (e.g. "  4800 Java St. ").
5. **getStreet, getCity, getState, getZipCode -**  Get methods.
6. **equals -** Two objects are considered equals if they represent the same address (i.e., both have the same street, city, state, and zip code values).
7. **toString -** Returns a string with the street, city, state, and zip code where each field is separated by a space.
8. You should define default values using static final.
9. Feel free to add any private methods you understand can help.
10. **This class in an immutable class.**

### PhotoProcessingSys Class

This class provides access to the image processing facilities the PictureManager class provides. Each PhotoProcessingSys object is associated with a customer and it keeps track of a balance that reflects the cost for any image processing the customer requested. See the PublicTests.java class to have a better understanding of the functionality expected for this class. Remember that some tests have a .txt file associated with them (e.g., pubTest1.txt).

* **Constructor -** Takes as parameters a customer's name (string), and customer's address (four string values representing street, city, state, and zip code). An Address object will be created and used to initialize an Address instance variable. To practice exceptions, make sure you catch the IllegalArgumentException while creating the Address object. If an invalid address is provided, just use the default address associated with the Address class. Notice you do not need to generate any error message in this case. There are other tasks associated with the constructor (e.g., initializing balance to 0) that you need to figure out as part of your project.
* **Default Constructor -** Uses "NONAME" for the customer's name and relies on the default address associated with the Address class.
* **toString -** Prints the customer's name, address, and balance. Each piece of information is printed on a line by itself using the messages "Customer Name: ", "Customer Address: ", "Balance: ". See the public tests for format information.
* **imageTransaction -** Takes an image's name (imageName), a task description string (task), a task options string (taskOptions), and a boolean value (graphicalMode). This method is responsible for processing image requests.
  + The method will first call PictureManager.graphicalModeOn() if graphicalMode is true and PictureManager.graphicalModeOff() otherwise.
  + Process the specified task. The tasks we can have are:
    1. **display -** Displays the picture associated with imageName.
    2. **clear -** Clears the screen of any pictures that have been displayed.
    3. **displaylast -** Displays the last picture that was displayed.
    4. **blackandwhite -** Displays a black and white version of imageName.
    5. **posterize -** Displays a posterized version of imageName.
    6. **blackandwhiteposterize -** Displays a version of imageName that is in black and white and posterized.
    7. **selectcolors -** The taskOptions includes the colors that will be selected. We will use r or R to select red, g or G for green, and b or B for blue. Notice that many instances can appear in the string and spaces can appear anywhere. Here some examples of strings: "rb", "rBrrrrG",  
       "   r R     G  ". Once any character appears, that color will be selected.
    8. **Invalid option -** If one of the above options is not provided, the method will not do any processing.
  + **showMessageDialog message-** After one of the seven image processing options has been processed (or an invalid option has been provided) your program must display the message "Continue" using JOptionPage.showMessageDialog. **Only call JOptionPane.showMessageDialog if graphicalMode is true.**
  + **Return value -** The method will return a string that describes the task that took place. For all the seven image processing options the string to return is the string returned by the corresponding PictureManager method. If an invalid image processing option was provided, the method will return "Invalid photoProcessing option".
  + **Cost -** Each image processing transaction cost $1.00 except selectcolors which is worth $2.00. You must keep track of the balance associated with all the image processing requests.
  + **Transaction's log -** After each image processing transaction, you need to store the string (see return value above) in a StringBuffer. This string will be preceded by "Transaction #{NUMBER}" where {NUMBER} represents the transaction's number. Transaction numbers start at 1. Notice that an invalid transaction counts as a transaction.
* **getTransactions -** Returns a String (not StringBuffer) with the transactions.
* **getBalance -** Returns the balance.
* Feel free to add any private methods you understand are needed.

## Submission

Submit your project as usual. Make sure you check the results in the submit server.

## Academic Integrity

Please make sure you read the academic integrity section of the syllabus so you understand what is permissible in our programming projects. We want to remind you that we check your project against other students' projects and any case of academic dishonesty will be referred to the [Office of Student Conduct](http://www.jpo.umd.edu/).