CS1026a: Assignment 4

Due: December 6th, 9pm
Weight: 12%

Purpose:
In this assignment, you will use many of the Java concepts we have learned this term. Specifically, you will gain experience with:

- Strings and text files
- Writing your own classes

Task:

Many different kinds of applications exist that help people organize their photos. These applications often allow a person to organize them into albums or catalogs, add captions, dates, etc. In this assignment, you will write a simple version of such an application. Your "photo organizer" will read a file that contains information about a collection of photos, including a caption and a date, and will produce new "labeled photos" with the photo information just below the picture (a sample of a "labeled photo" is shown below). Then you will create an "album" (a directory) in which the new photos are sorted by date.

Pets
How embarrassing
4/5/2011

Part A:
First, you will write a new class called PhotoInfo that will model the information needed about an individual photo in order to create the labeled photo. The attributes of an object of this class will be a *unique identifier* for the photo, the *category* that the photo belongs to, the *date* that the photo was taken, a "*caption*", and the *name of the file* with the photo (a .jpg file). The class will contain the methods described below, one of which is to generate the labeled photo.

Functional Specifications for Part A

1. You are to write a new class `PhotoInfo` that will be stored in the file *PhotoInfo.java*.

2. It will have the following attributes (instance variables):

- `id`, of type String;
- `category`, of type String;
- `year`, of type int;
- `month`, of type int;
- `day`, of type int;
- `caption`, of type String, but with underscores ('_') instead of spaces;
- `photoFile`, of type String, which is the name of the file that holds the photo.

3. The class `PhotoInfo` will have a constructor that takes six parameters; the constructor will have the header:

```
public PhotoInfo(String ident, int nday, int nmonth, int
nyear, String cat, String cap, String picFile)
```

The class will also have an instance variable `photoPic`, which will be of type `Picture` and which will be created within your constructor; this will be the "labeled picture" that you will create.

To add text to a picture, you will need to add a method called `drawString()` to your `Picture` class. The `drawString()` method provided below is a variation of the one on page 224 of the textbook. The parameters are: the string to be put into the picture on which the method is invoked, the x and y coordinates of the position in the picture at which the string should be placed, and the font size of the string when it is placed in the picture.

```
public void drawString(String text, int xPos, int yPos, int
fontSize)
{
   Graphics g = this.getGraphics();
   g.setColor(Color.BLACK);
   g.setFont(new Font("Arial", Font.BOLD, fontSize));
   g.drawString(text, xPos, yPos);
}
```

The following code segment is an example of how one would call the `drawString()` method:

```
Picture picObj = new Picture(500,500);
String test = ("This is a test");
int fontsize = 16;
```

```
picObj.drawString(test, 100, 100, fontsize);
```

In your constructor you will be able to read in the picture using the name of the file stored in `photoFile` as you did in previous assignments and then using `drawString()` you will be able to create a new `Picture` containing the original picture with the category, caption and date beneath the original photo. The labeled photo will be a `Picture` object, and you can think of it as a simple "collage" consisting of an image with text beneath it.

4. The public methods for the class will be:

- An **accessor** (getter) method for each of the attributes, that returns the value of the attributes, including the attribute `photoPic`.

- A method `toString()` that returns a string containing the information about the photo, in the format:
  `caption(identifier, category, filename, date)`

  Note that "date" should contain the day, month and year; the format is of your choice. This method will have the header

  ```
  public String toString()
  ```

- You may, of course, add other methods to the class that you might find useful.

5. One nice way to test a class that you have written yourself is to add a main method to the class, after all the other methods. This allows you to test your new class without writing a separate test program (as we did in Assignments 2 and 3). This main method will have the usual header, and will contain statements that try out the methods of the PhotoInfo class, for example:

- Create a `PhotoInfo` object with sample parameters first name.

- Invoke the getter methods on it and print the return values.

- Call the `toString()` method and print the return value.

- Invoke a getter method to get the new "labeled photo" and display it.

Part B:

You will write another new class called `PhotoList` that models a list of photos. The information for all the photos will be read in from a text file which consists of one line of information per photo. Each line of the file will be of the form

identifier, day, month, year, category, caption, filename

An example of a line in the provided test file *myPhotoLists.txt* is:

```
cat02 4 5 2011 Pets How_embarrassing granny_cat.jpg
```

The list of photos will be kept in one array of type PhotoInfo.

The ClassList class will contain the methods described below.

Functional Specifications for Part B

1. You are to write a new class `PhotoList` that will be stored in the file *PhotoList.java.* It will start as follows:

```
import java.util.*;
import java.io.*;
public class PhotoList {
```

2. It will have the following attributes (instance variables) declared by

```
private PhotoInfo[] photoArray;
private int numPhoto = 0;
```

The array `photoArray` will hold the `PhotoInfo` objects created from each line of the photo information file. The variable `numPhoto` will be the number of photo, i.e. the number of lines in the file of photo information that we start with. (Note: your instance variable declarations should be **exactly** as above.)

3. The class will have a constructor that takes as it parameter the name of the file of photo information. The constructor will have the header

```
public PhotoList(String fileName)
```

The constructor will need to initialize the instance variables from the contents of the file. The algorithm for the constructor is:

- Read the lines of the file whose filename is the parameter into an array of `String`. (This array will be a local variable in the constuctor.) Get the number of photos from the size of the file. (Hint: see Lab 10).
- Create the array object `photoArray` that was declared as an attribute. Its size will be the number of photo. (Note that at this step, you are just creating an array object referenced by the instance variable `photoArray`; you are **not** yet creating the individual `PhotoInfo` objects in the array.)
- For each line that was read in from the file:

- Get the photo information (identifier, day, month, year, category, caption, filename) from the line, retrieved as tokens from the line. (Hint: use the `StringTokenizer` class as in Lab 10.).
- Create a `PhotoInfo` object using the photo information, and store it as the next element of `PhotoArray`.

4. The public methods for the class will be:

- A method `listPhotos()` that displays the photo list on the screen as text; you can make use of the `toString` method that you wrote for your class `PhotoInfo`. The header for this method will be

```
public void listPhotos()
```

- A method `showPhotos()` that shows all the info photos for the class on the screen. (The user can then move them around on the screen.) The header for the method will be

```
public void showPhotos()
```

- A method `storePhotos()` that stores all the labeled photos as .jpg files in a directory whose name is passed as a parameter. The header for the method will be

```
public void storePhotos(String directory)
```

This method will write each `Picture` object in `photoArray` to a file in the directory. The filename for each `Picture` object will be the unique identifier for that photo concatenated with "_" (a single underscore) and concatenated with the photo's category. From the example above, if the information in the file was

```
cat02 4 5 2011 Pets How_embarrassing granny_cat.jpg
```

Then the name of labeled photo would be "cat02_Pets.jpg". So, if the directory passed in is ***myPhotos,*** then this photo would be written to *myPhotos/cat02_Pets.jpg.*

You will need to add the following code segment right at the beginning of this method; it creates a new directory if the parameter directory does not yet exist. It also adds a "slash" at the end of the directory name if the user has omitted it:

```
char end = directory.charAt(directory.length() - 1);
if (end != '/' || end != '\\')
     directory = directory + '/';
File directoryFile = new File(directory);
if (!directoryFile.exists())
    directoryFile.mkdirs();
```

- A method `sortPhotosByDate()` that can sort `photoArray` by dates. displays the photo list on the screen as text; you can make use of the `toString` method that you wrote for your class `PhotoInfo`. The header for this method will be

```
public void sortPhotosByDate()
```

  There are many sorting algorithms; two of the most straightforward ones are the ***bubble sort*** and ***insertion sort***. You can find examples of these on the web. (Hint: Remember that you are sorting by *dates* – that means that you must account for year, month and day!).

5. You will be using the methods of `PhotoList` in Part C, which will also serve as a test of your code.

Part C:

You will now write the application program (the class that contains the main method) that will allow the user to create an "album" (well, really just a directory) with labeled photos sorted by dates. The class will be called `CreatePhotoLists` and will be stored in the file *CreatePhotoLists.java.* The algorithm for the main method is:

- Get the filename of the photo information file from the user.
- Get the name of the directory for storing the class info photos from the user.
- Display the list of photos on the screen as text.
- Display (show) the labeled photos on the screen.
- Write the labelled photos as .jpg files to the directory.

Testing:

An example file which you can use for testing is *myPhotoList.txt*, available on the Assignment 4 web page along with a collection of photos as .jpg files; these photos are in the zipped file *myPhotos.zip.* You may create your own test file with your own photos in order to test your application; if you would like your TA to see *your* test data, make sure you submit your version of the photo info file and your .jpg files as well!

Non-functional Specifications

1. **Commenting:**
   - ✓ Each class should start with a brief description of the class and the author's (your) name.
   - ✓ Each method should start with a brief description of its purpose, its parameters, and

its return value (if any).
- ✓ Variables should be commented as to their purpose within your code if their variable names are not self-explanatory.
- ✓ Include "block comments" that explain the purpose of sections of code, as you have seen in code examples in the Lecture Notes. These are important to let the reader of your code know the purpose of each section of code.

2. Use Java conventions and good Java programming techniques, for example:
- ✓ Java naming conventions
- ✓ Meaningful names for variables
- ✓ Named constants instead of "magic numbers"
- ✓ Readability: indentation, white space, consistency
- ✓ Appropriate loop structures and conditional statements

3. Remember that assignments are to be done individually and must be your own work.

What to Hand In

PhotoInfo.java, PhotoList.java, CreatePhotoLists.java

(If you want the marker to use your own test data, submit that as well; see the note under Testing above.)

What You Will Be Marked On

1. Functional specifications:
i. Are the required methods written according to specifications?
ii. Do they work as specified?
iii. Are they called as specified?

2. Non-functional specifications: as described above. **Especially note the Commenting section.**