Computer Science 111L Intro to Algorithms and Programming: Java Lab

Programming Project #4 PictureViewer (30 points)

Due: 11/1/18

Phase 1 (3 points)

Design your methods

In this phase, we will revisit the concept of an algorithm. Remember, an algorithm is a detailed set of instructions which describe how you intend to solve a specific problem in a finite amount of time. Therefore, to complete this phase, you must design 3 **algorithms**. Write out **both** a flow chart and pseudocode for the algorithms, then show these algorithms to me in lab class.

Here is a description of each algorithm:

Given an integer-type *parameter* named **`current_number`**, a **global** integer variable named **`image_number`**, and two **global** symbolic constants:

```
final static int MIN_NUMBER = 1;
final static int MAX_NUMBER = 8;
static int image_number = 1;
```

Write an algorithm named **`forward`** which increases the value passed to its parameter **`current_number`** by one each time the algorithm is invoked. In other words, when **`forward`** is invoked and passed a number like 1, then **`forward`** returns a value of 2. When **`forward`** is invoked and passed a value of 2, then **`forward`** returns a value of 3. However, when `MAX_NUMBER` is reached, the algorithm should "wrap around" and return `MIN_NUMBER`. This algorithm must **never** return a value larger than `MAX_NUMBER`.

Write an algorithm named **`backward`** which decreases the value passed to its parameter **`current_number`** by one each time the algorithm is invoked. In other words, when **`backward`** is invoked and passed a number like 8, then **`backward`** returns a value of 7. When **`backward`** is invoked and passed a number like 7, then **`backward`** returns a value of 6. However, when `MIN_NUMBER` is reached, the algorithm should **STOP** and return the value of `MIN_NUMBER`. This algorithm will **NEVER** "wrap around".

Write an algorithm named **`createFileName`** which concatenates and returns a String like **pictureX.jpg** where X is the value of the input parameter **`current_number`**. In other words, when **`createFileName`** is invoked and passed a number like 8, then **`createFileName`** returns the String **picture8.jpg**. When **`createFileName`** is invoked and passed a number like 1, then **`createFileName`** returns the String **picture1.jpg**.

BE SURE TO HAVE ME CHECK AND APPROVE YOUR ALGORITHMS BEFORE PROCEEDING TO PHASE 2. NO ADDITIONAL POINTS WILL BE AWARDED IF YOU DO NOT SHOW ME YOUR ALGORITHMS.

Phase 2 (12 points)

Implement your algorithms

In this phase, you will implement your algorithms in an actual Java class by adding six methods to the attached PictureViewer class. **Do not modify the existing code in the PictureViewer class, simply add your own methods to this class**. Your methods should behave as described in your Phase 1 algorithms, that is, the **`forward`**()method should "wrap around" when it reaches the last number, the **`backward`**() method should STOP when it reaches the first number, and the **`createFileName`**() method should return the String **picture8.jpg** when the method is invoked and passed a number like 8.

Your **`forward`** () and **`backward`** () methods **MUST** use a parameter named **`current_number`** for input and MUST return an int-type value, they **MUST NOT** use the global **`image_number`** variable directly. Also, both methods SHOULD use the symbolic constants `MIN_NUMBER` and `MAX_NUMBER` instead of the hardcoded, literal numbers 1 and 8. Don't try to make the methods overly complex, they should be very simple; when **`forward`** () is passed a **`current_number`** of 3, the method **returns** a 4. When **`backward`**() is passed a **`current_number`** of 6, the method **returns** a 5.

The **`createFileName`**() method will use a parameter named **`current_number`** for input, and will return a String representing a filename like **pictureX.jpg**, where X is the value of **`current_number`**.

Use the following method headers:

```
public static int forward(int current_number) {
      // return the new number
}
public static int backward(int current_number) {
      // return the new number
}
// The static methods above should use the symbolic constants
// MIN_NUMBER, MAX_NUMBER.  Do not use hardcoded 1 or 8

public static String createFileName(int current_number) {
      // return a String representing a filename like pictureX.jpg
}
public static String createRandomName() {

      // Return a String representing a filename like pictureX.jpg
      // where X is a random number between MIN_NUMBER and MAX_NUMBER
}
```

```
public static void showMenu() {
      //write a loop
            /*
            1. Inside the loop, display a menu with options
               1..N for each of the each of the methods above
               as well as an exit option.
            2. Print the value of the global image_number variable.
            3. Prompt the user for a menu option choice.
            4. Get the user's menu option choice, then invoke
               the correct method using a SWITCH.  For example,
               if your menu shows option 1. Forward, and the user's menu
               option choice is 1, the switch should then invoke the
               forward() method.
            */
}
public static void main(String[] args) {
      // Invoke showMenu() here.
      // Learning how to invoke a method is crucial to this chapter,
      // so DO NOT ask a fellow student or even the instructor how
      // to do this.
}
```

The **createRandomName** () method has NO input and returns a String representing a filename like **pictureX.jpg**, where X is a random number between MIN_NUMBER and MAX_NUMBER, inclusive.

The menu displayed by **showMenu**() should have options for invoking **forward**(), **backward**(), **createFileName**() and **createRandomName**(). Be sure to use global **image_numbe**r as the argument when invoking the **forward**(), **backward**() and **createFileName**() methods. Remember to update **image_numbe**r by assigning it the numbers returned by the **forward**() or **backward**() method. Also, when the exit option is chosen in **showMenu**(), use `System.exit(0);` to terminate the program. At this point, your program should display a menu, invoke methods and print the **image_numbe**r in the NetBeans console output window.

Phase 3 (9 points)

Add overloaded methods

This phase will require you to write two **overloaded** methods. An overloaded method is one that has the SAME NAME as another method. We can create a method with the same name as another, as long as it has a different list of parameters. The difference must be in either the **number** of parameters or in the **data types** of those parameters. Add the following OVERLOADED methods to your PictureViewer class:

```
public static void forward() {
      // Overloaded method.  This method is allowed to
      // use the global image_number variable for both
      // input and output.  In other words, the method
      // should check if image_number + 1 is less than
      // or equal to MAX_NUMBER.  If it is, then the
      // method increases image_number by 1.  Othwerise,
      // the method sets image_number to MIN_NUMBER.
      // This method should "wrap around" like the first
      // forward() method does.
}
public static void backward() {
      // Overloaded method.  This method is allowed to
      // use the global image_number variable for both
      // input and output.  In other words, the method
      // should check if image_number - 1 is greater
      // than or equal to MIN_NUMBER.  If it is, then
      // the method decreases image_number by 1.
      // Otherwise, the method sets image_number to
      // MIN_NUMBER.  Like the first backward() method,
      // this method should NOT "wrap around".
}
```

In addition to having different parameters (no parameter at all versus one parameter), these NEW method overloads will operate differently. Instead of using a parameter for input and a returned value for output, these new method overloads are allowed to use the global **`image_number`** variable.

Add two new options to your program's menu to invoke these new, overloaded methods. Both the **`forward`**() and **`backward`**() method overloads should do basically the same thing as the original **`forward`**() and **`backward`**() methods, but these overloaded versions do NOT require a parameter and do not return anything. Test your program. Make sure you can page forward using both **`forward`**() methods and backward using both **`backward`**() methods. **DO NOT CONTINUE TO PHASE 4 UNTIL YOU ARE SURE YOUR PROGRAM WORKS AS INTENDED.**

Phase 4 (6 points)

Show the images

Add some code to invoke the existing **`showWindow`**() method, pass the filenames the **`createFileName`**() and **`createRandomName`**() methods create.

In your **`showMenu`**() method's switch, under the case which invokes your **`createFileName`**() method, add a statement which invokes the **`showWindow`**() method, passing the String returned by **`createFileName`**() as the argument. Also in your **`showMenu`**() method's switch, under the case which invokes your **`createRandomName`**() method, add a statement which invokes the **`showWindow`**() method, passing the String returned by **`createRandomName`**() as the argument.

Test your program. You should be able to page forward and backward with proper behavior when limits are reached. When either **`createFileName`**() or **`createRandomName`**() methods are invoked, a frame containing an image should appear onscreen (you may need to click a button the taskbar to see it). The correct filename should be shown at the top of the frame. In other words, if the **`createFileName`**() method is invoked when **`image_number`** is 7, a frame will open and the frame should show a picture and the title bar of the frame should be **picture7.jpg**. If the **`createRandomName`**() method is invoked, a frame will open and the from should show a picture and the title bar of the frame should be a random filename between and including **picture1.jpg** to **picture8.jpg**. All 8 pictures in this project are different from one another.

Remember to write a Universal comment header at the top of your source code file.

When you have completed the exercise, ZIP the entire project folder and upload this ZIP file to Canvas. In NetBeans, click **File**, select **Export Project**, click **To ZIP…**, select the location where you want the ZIP file to be saved, type the name of the ZIP file, **Project4.zip** in the textbox, click the **Save** button, then click the **Export** button.

If the IDE you are using does not have an export-project-to-zip option, you may have to manually ZIP the file by navigating to the project folder in a file browser and selecting your OS's "compress folder" option. Under Mac OS, the file browser is called **Finder** which offers a **Compress Folder** option. Under Windows, the file browser is called **File Explorer** and it offers a **Send to => Compressed (zipped) folder** option.