COSC 1P02 Assignment 5

Shh! Just blend in!

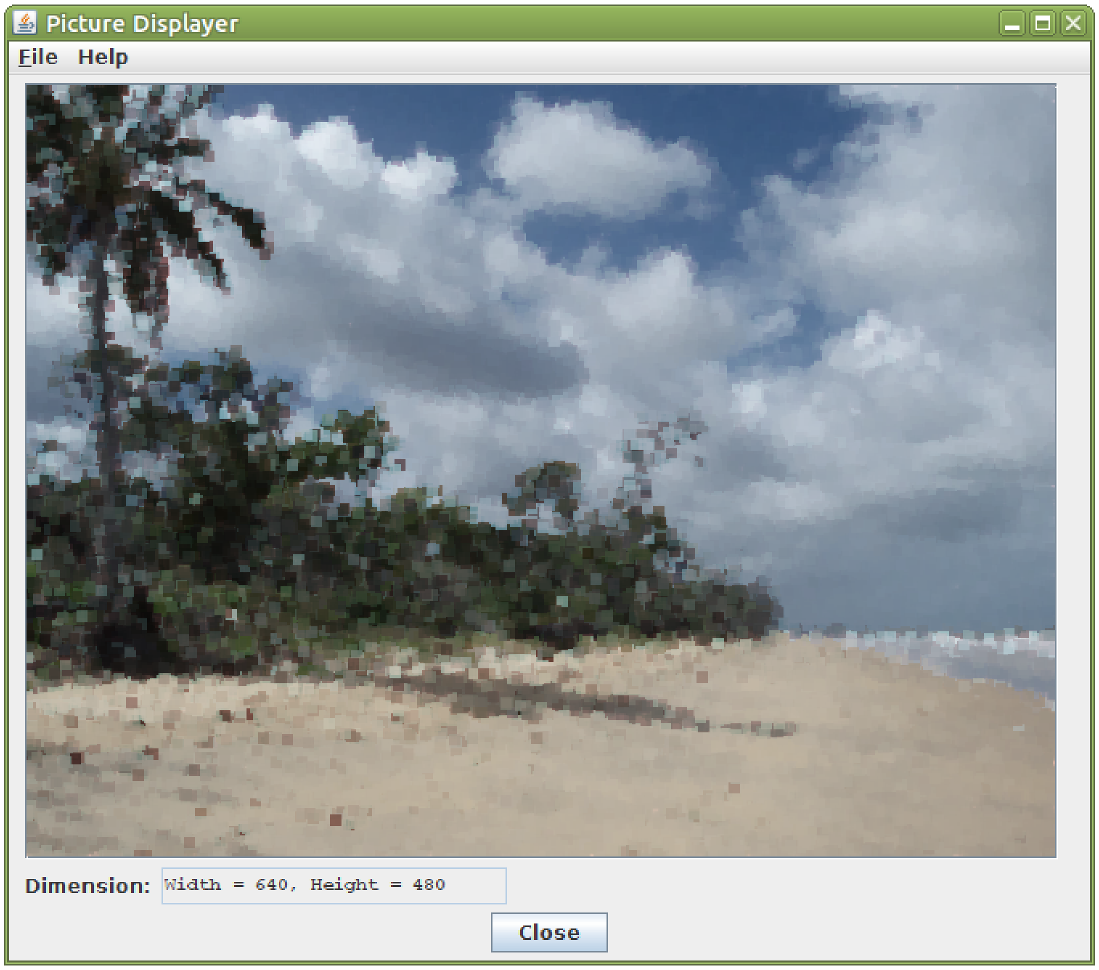
Refer to Sakai for due date

This assignment deals with accessing collections, using the indexes of specific datas.

For this assignment, you're assigned one task, comprised of three steps.
Write a .java file for *at least* the final task; under an `Assign_5` package.

The final task: blended cubistical impressionism

Your final goal is to produce something like the following:



The basic idea is:

- Provide a `Picture` to a function (`cubistical`), and an `int`eger for how many 'dots' should be in the final Picture
  - For simplicity, each 'dot' is actually a 7×7 box
- The `cubistical` function creates a *new* `Picture` (with dimensions matching the original), draws the dot-pattern onto that new `Picture`, and then returns it to the constructor (which then displays the new version)
- Each 7×7 box-dot blends a colour into the target image:
  - Pick a random coordinate
  - Take the original colour at that location, and 'blend' it with the current colour at the same location in the new/target picture
- Repeat for however many dots were requested.

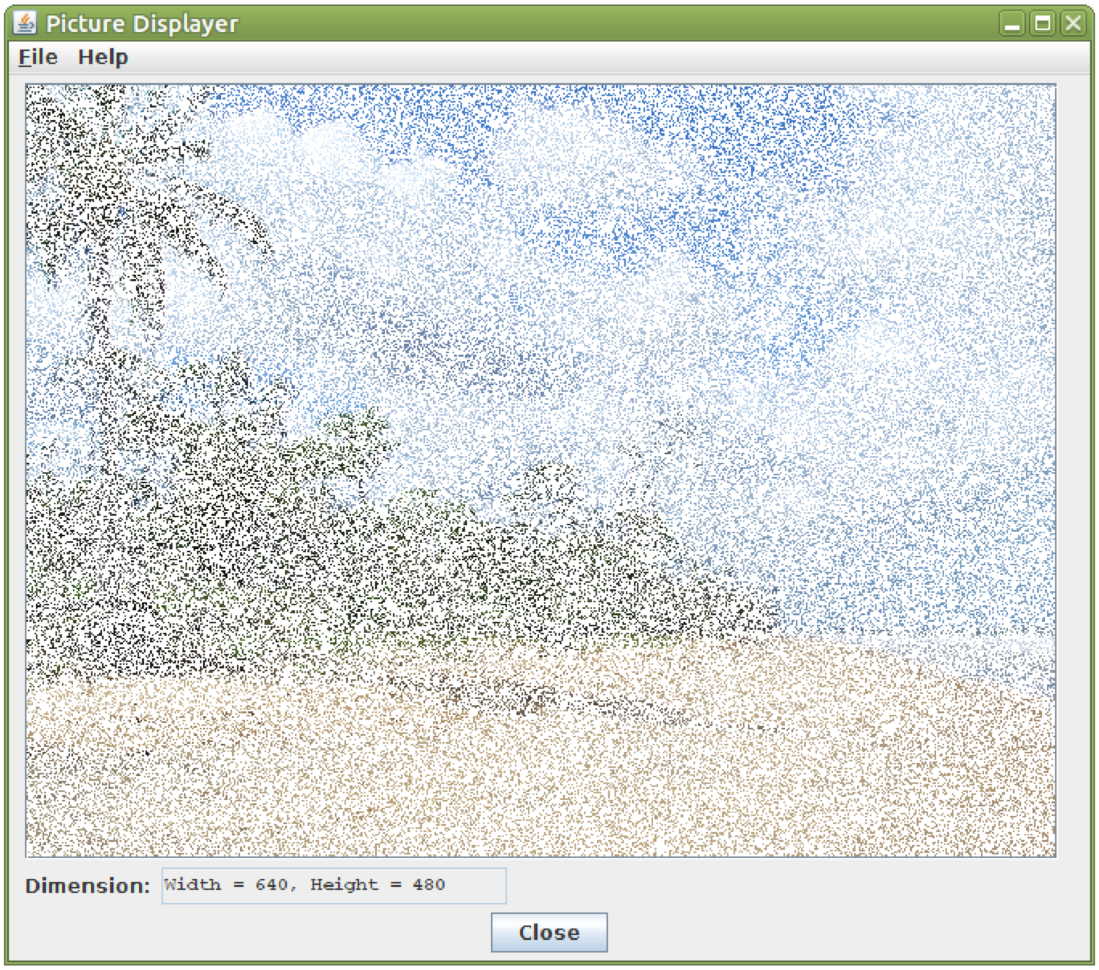
To help you in decomposing this task, the following pages contain a suggested approach.

Step One: creating a basic pointillism

To be clear, this step is optional, ***but*** it should walk you through some of the steps.

(Also, creating a random placement of dots is part of the marking scheme, so you can still get partial credit even for just this part)

A pointillism version of the image might look like:



In this case, it has 100,000 single-pixel "dots". Here's what I did:

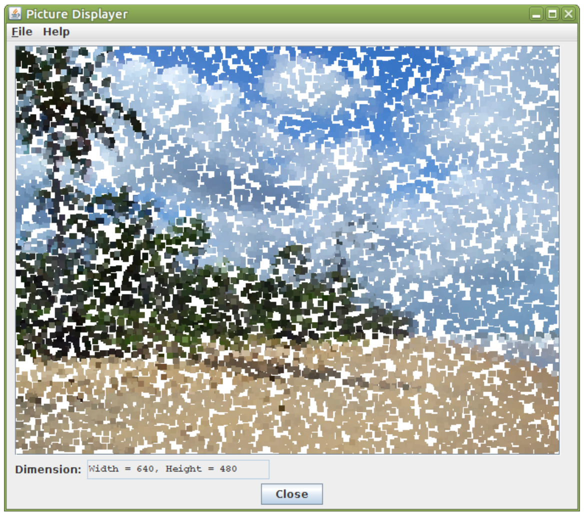
- My constructor loads the user-selected image, and displays it
- It then gives the image as a parameter, along with 100000, to a `pointify` function
  - That `pointify` first creates a *new* image with the same dimensions
    - Of course, that new copy is initially white
  - The function then iterates the specified number of times (in this case: 100,000)
    - Each time, it generates a random *x* and *y* value, to fit the image
    - It retrieves the Pixel at those coordinates from the original and the new image
    - It gets the `Color` (or red/green/blue) from the source `Pixel`, and copies it over into the target `Pixel`

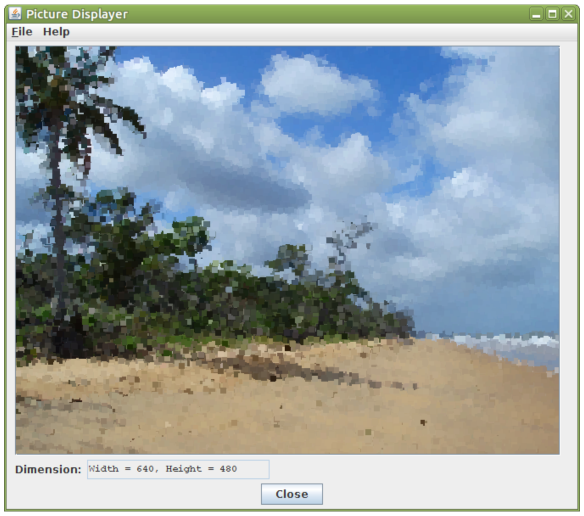
Some reminders:

- The `Math.random()` function gives a number from [0..1). Multiply that by a width or height, and you'll get a value that fits within that range (so long as you convert to `int`)
- Remember you need to create a *new* `Picture`! Otherwise you won't see any change!
- If you test with a 'quantity of dots' of around 100 or so, you should see a mostly-white canvas, with a few coloured specks randomly-placed. That's a good sign!

Step Two: enlarging the dots

The 'dots' we draw with shouldn't be single-pixels. Instead, we want small boxen:





The left has 10,000 dots; the right, 100,000.

There are two considerations when making these larger dots:

- Since each dot is 49 pixels (7×7), that suggests another loop or two
- If you generate a 'main' coordinate per-dot, the other 48 will either be around that coordinate, or extending out (with the pixel being its corner)
    - Regardless of which you go with, you run the risk of *leaving the bounds of the image*!

To simplify this, after retrieving the `Color` (or `Pixel`) from the original image, I pass that, as well as the coordinate, to a *separate method*.

- That separate method (for drawing a *single* box-dot) can deal with the *conditional statement* needed to ensure it only *actually* draws if it's a legal position

When actually doing this, you might even want to break *this* up into two steps:

- First just move the drawing of a single-pixel-dot into a separate method
- Then replace the contents of that method with code for drawing a box-shape

Note: if you can get *this* far, so long as your commenting/style/etc. Are fine, you'll already have enough to pass this assignment.

What's left?

If you get the above completely working, the only thing left is the blending.

- Presumably you can find the 'average of two numbers', right?
    - Can you do it three times?

Submission:

For submission, you must submit a **.zip** file containing the following:

- A folder called `Assign_5`, containing your submission
    - Your `.java` source file
    - Your `.drjava` file, that you used to *create* the assignment

    - A `.pdf` copy of your starlight pattern
        - When the program finishes, before clicking *Close*, just click *File → Print Image of Window...* → and then select to print to a file
        - If you're having any trouble doing this on Windows, CutePDF might make it easier
        - If you find yourself struggling with the output shortly before you need to submit, for *only this assignment*, you can just use *File → Save Window as Image...* and save it as a picture instead

On Sakai, submit your **.zip** file as an attachment, and click to Submit.

Note: Do not submit a `.rar`, `.tar.gz`, `.7z`, etc. *Every* major operating system supports **`.zip`**, so it is mandatory if you wish to receive a grade for your assignment.

Standards:

Ostensibly, you'll be graded for following basic coding standards and documentation requirements. At the very minimum:

- A comment at the top of all source (`.java`) files including your *name* and *student number*
- Variable names that are either standard or descriptive
    - Using things like `i` and `j` for loop counters? Standard
    - Using a variable like `count`? We can guess what you meant
- For any *blocks* of code (e.g. loops) insert a brief comment so the reader knows what's inside (e.g. *what*'s being repeated)
- Put a comment in front of any *method*, to briefly explain what it does
- Anything you think the marker might not understand? Add a quick comment
    - This *probably* isn't a concern for this assignment

Additionally, try to remember to fix the indentation of your source files. Jagged margins can be harder to follow along with.

DrJava (and other platforms)

If you aren't planning on using DrJava, you should consult with the course coordinator (Maysara) to ensure that's okay.

Make sure to include ***everything*** (i.e. the whole folder), to avoid forgetting an important file. (e.g. even if you include a .class file, and a .pdf copy of the source file, the marker can't confirm that compiles)