School of
Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

Assignment 4

Intro to Programming: Assignment 4

- Due Nov 24 23:30

Goals

This assignment will give you experience in writing programs that get input from files. It will also give you more practice using conditionals and loops.

Resources and links

- Download Zip file of necessary code and data.
- Submit your answers.
- Marks and Feedback (When available)
- Video of what your ImageRenderer should do.

Summary

Programs with files, loops, and conditions

- ClassTimes: ➡ Write program to extract useful information from a file containing university timetable data.
  Submit your version of `ClassTimes.java`
- ImageRenderer: ➡ Write a program to render `ppm` image files (the simplest possible image format).
  Your version of `ImageRenderer.java`

Preparation

Copy the files to your USB stick, or download and unzip the zip file from the web.

Look at the model answers to assignment 3, and make sure you understand all the components of the programs. Also, go over the code examples from the lectures that used files.

Part 1: Class Times

The university timetable system outputs a list of all the classes of all the courses in each trimester. However, the list is very long, and reading through it takes a long time.

Complete the `ClassTimes` program that will help a user find certain information from this timetable more quickly. The program has four methods; each method will read the data from a file containing the class timetable data, and print out some useful information. All the methods are closely related and will have a very similar structure: loop through the file, reading the values on each line and processing them. See the example output of the methods at the end of the assignment.

The class timetable file is called `classdata.txt`.

Each course may have lots of different classes: it may have several lectures and/or tutorials and/or laboratory sessions on different days at different times in different rooms. Each line of the file specifies one class or session for a course, so that there may be may be many lines for a each course.

Six of the lines of the file are shown here:

```
COMP102  Comp-Lab   Wed  1600  1700  CO219
COMP102  Comp-Lab   Wed  1600  1700  CO238
COMP102  Lecture   Mon  1100  1150  KKLT303
COMP102  Lecture   Wed  1100  1150  KKLT303
COMP102  Lecture   Fri  1100  1150  KKLT303
COMP102  Tutorial   Tue  1610  1700  AM104
```

The format of the lines is as follows:

- The first token is the course code
- The second token is the type of class
- The third token is the day of the week
- The fourth and fifth tokens are the start and end time of the session (using 24 hour time)
- The last token is the room code. The room code always starts with the building code, *eg* "KK" or "WIG".

Note, the data is based on a real Victoria course timetable, but has been modified and simplified.

Core

Complete two methods:

- `printCourse(String targetCourse)`
  The method has one parameter - a course code. It should read the classdata file, printing out (to the terminal window) all the class type, day, start and end times, and room for each class session of the given course (*ie*, data from all the lines where the first token on the line is equal to the course code).

It should read the first token on each line using `next()`, and can then can either read the rest of the line using `nextLine()` or read the each value on the rest of the line separately. You will need to use some of the methods in the `String` class.

- `printInRoom(String targetRoom)`
  The method has one parameter - a room code. It should print out a title: *eg* `"Classes in KK201"` and underline it. It should then read the file and print out the course code, day, and start time for each class session that is in the target room.

Completion

- Complete `printRoomOnDay(String targetRoom, String targetDay)`
  The method has two parameters - a room code and a day. It should print out a title: `"Classes in ... on ...."` and underline it. It should then read the file and print out the course code, class type, start time, and end time for each class session in the target room on the target day. The start time and end time should have a colon in them, eg `09:50` or `11:00`.
  At the end, it should print a message saying how many classes are booked in the room on that day. Take care when there is just one class or no classes at all, printing the message like this:
  - `"There are 8 classes in MCLT103 on Thu"` or
  - `"There is 1 class in VSLT2 on Wed "` or
  - `"No classes are booked in CO218"`

Challenge

- Complete `meanClassLength(String building)`
  The method prints the average duration in minutes of all classes scheduled in the specified building. The building code is always the first part of the room code.
  Hints: be careful with the times - they are not ordinary integers (there are not 100 minutes between 1000 and 1100), and
  do not cause an error if there are no classes in the room.

Testing your program:

You should compare the output of your program to the example output at the end of this assignment to help you test your program. But you should also test it on other inputs.

Part 2: Image Renderer

Digital image files are everywhere. The core information in a digital image is the color of each individual pixel of the image, so an image file must store that information. A colour can be represented by three numbers, one for each of the red, green, and blue components of the colour, so we can represent an image in a file by storing three numbers for each pixel. Exactly how we do that depends on the *format* of the image file. A very simple format will just list all the numbers for all the pixels, so that a 100x100 pixel image (which is pretty small) would have 30,000 numbers: three numbers for each of the 10,000 pixels. However, this will end up being a rather large file, and most of the common image file formats (such as `jpg`, `gif`, and `png`) have more complex formats in which the information is encoded and compressed to save space and remove redundancy. Turning the contents of these compressed image files into a coloured image on the screen can be a very complicated process, but the files can be very much smaller than when using the simple format.

For this assignment, you will write a program to render image files in the **plain ppm** format ("portable pixel map") - one of the simplest possible (and least efficient) file formats.

A plain ppm file starts with a four pieces of information describing the image, followed by the color values for each pixel in turn, starting at the top left of the image, and working from left to right along each row and working row by row down the image.

For example, here is a little plain ppm image file (`image-tiny.ppm` , which is a small part of the back of the bee from the `image-bee.ppm` file):

```
P3
12 5
255
200 182 163 215 198 177 130 116 93 37 28 9 31 22 7
81 67 38 83 71 42 6 5 6 0 0 0 57 68 60 97 112 104
97 92 76 202 186 165 97 82 60 32 25 5 38 30 13 103 90
63 158 140 97 58 49 25 43 42 17 107 104 74 127 140
113 95 102 79 66 58 41 71 57 37 41 30 7 82 71 41 111
95 64 174 157 120 115 101 63 49 43 12 67 65 30 126
124 74 133 136 97 88 87 62 98 93 54 78 63 37 108 93
62 121 104 69 135 120 88 190 172 139 36 30 15 1 0 0
16 17 9 64 77 58 50 57 39 7 2 0 105 106 64 121 103 71
117 100 67 159 144 113 212 197 171 161 146 114 0 0 0
0 0 0 37 48 32 72 88 68 24 26 19 12 12 9 74 72 49
```

The first token is the string `P3` which says that the format of this image file is a plain ppm file.

The second and third tokens are integers that are the size of the image: the width (number of columns of pixels) and the height (the number of rows of pixels). In this case, there are 5 rows of 12 pixels each.

The fourth token specifies the maximum possible value of the colour values for the pixels. In this case, the colour values for the red, green, and blue components each go between 0 and 255. 0 would mean nothing for a component; 255 would mean the maximum strength for the component. That means that a colour consisting of three 0's would be black; three 255's would be white; a 255 0 0 would be bright red. This means that there are 256x256x256 = almost 17 million distinct possible colours.

[If the colour depth were only 7, then each colour value would be between 0 and 7 and there would only be 8x8x8 = 512 possible colors. This would correspond to a "colour depth" of 3 bits per component.]

The rest of the file is the color values of the pixels, with three numbers for each pixel. Therefore there will be 36 numbers for the first row (up to `97 92 76` at the beginning of the 3rd line), 36 numbers for the second row, etc. The line breaks don't mean anything special, (although the format specifies that the numbers should be broken into lines with no more than 70 characters).

You are to complete the `ImageRenderer` program that reads a a plain ppm file and renders it on the screen.

Core

Complete the `renderImageCore` method. It should ask the user for a file (for example, using the `UIFileChooser.open` method), and read the file, rendering the pixels to the graphics pane. It should check that the file starts with the correct "magic number" (P3). If not, it should print a message and exit the method. Otherwise, it should read the number of columns and the number of rows. It should then read the colour depth, but it can simply ignore it and assume that it is the "normal" 255. It should then read the three color values for each pixel in turn, setting the color of the UI and then drawing a square pixel on the graphics pane. It will need nested loops to work along the columns for each row and work down all the rows.

The top left corner of the image should be placed at (20,20), and each pixel should be drawn as a 2x2 square (ie, the image should be zoomed-in 200%). Use the constants defined at the top of the file.

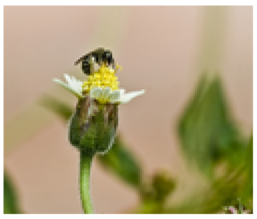
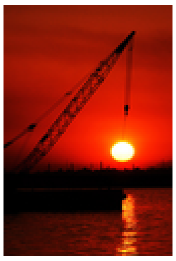
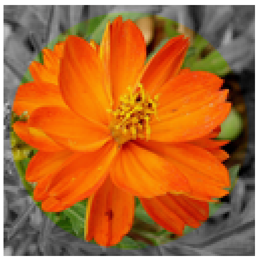
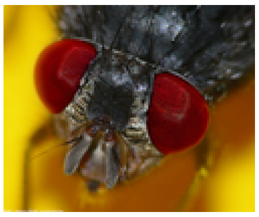
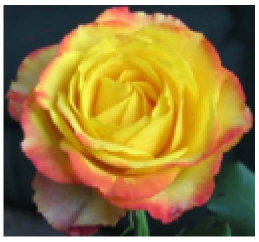
Hints:

- You will need a nested loop to do this: an outer loop that goes through each row, and an inner loop that goes along the columns of the current row. See the example code in the lecture notes.

There are several ppm images that you may test your method on:

- 1-image-bee.ppm,
- 1-image-cats.ppm,
- 1-image-crane.ppm,
- 1-image-flower.ppm,
- 1-image-fly.ppm, and
- 1-image-rose.ppm

Here are `png` forms of the same images so you can see what they look like.

1-image-bee:	1-image-cats:
	
1-image-crane:	1-image-flower:
	
1-image-fly:	1-image-rose:
	

Completion

You are to complete the `renderAnimatedImage` method that renders ppm files containing animated images.

A ppm file may contain a sequence of images right after each other, representing an "animated" image, which should be rendered by drawing the first image, pausing briefly, then drawing the second image in its place, and repeating until all the images have been drawn. Each image will have the four "header" tokens, then the pixel color values. This is the same process as an "animated gif", (though gifs are encoded more efficiently and have more options).

Your `renderAnimatedImage` method should handle animated images by continuing to render images from the file until there are no more images in the file. Use a 100 - 200 millisecond delay between images. You should make it display each image just once. (It is OK to repeat the animation multiple times, but this makes testing more difficult).

There are three animated ppm images:

- 2-multi-image-ball.ppm,
- 2-multi-image-athletes.ppm,
- 2-multi-image-flag.ppm

Challenge

The real plain ppm format is actually a little more complex than described above:

- The first four tokens do not need to be placed on separate lines, just so long as they are separated by white space or newlines.
- The file can contain comments that are ignored by the renderer. A comment starts with #, and lasts to the end of the line, and it can be placed anywhere between the first token (P3)and the maximum colour value, for example:

```
P3 # a plain ppm file
# this is a part of a bee
12 # width of image
5  # height of image
# the image was extracted from a larger image
# converted from a free gif file from the web
255
200 182 163 215 198 177 130 116 93 37 28 9 31 22 7 81
67 38 83 71 42 6 5 6 0 .....
```

Furthermore, **plain ppm** is just one in a family of related formats. **Plain pgm** ("portable grey map") is for grey scale images. pgm files start with the token "P2", and each pixel is represented by a single number (between 0 and the maximum colour value) specifying the grey level of the pixel.

Make your program

- handle comments (by noticing them and throwing them away).
- handle the colour depth properly: if the colour depth is 15, for example, it should scale up all the values by 255/15 before setting the color.
- Recognise and render plain pgm images also (use the first token to determine which version to render).

There are several images with these additional features which you can test your method on, some scaled, some with comments, some both scaled and with comments:

- `3-image-fly-scaled.ppm` (colour depth is just 3 bits - max colour value is 7)
- `3-image-fly-comments.ppm` and `image-fly-comments2.ppm` (comments in the header)
- `3-image-fly-scaled-comments.ppm` (scaled and with comments)
- `3-multi-image-flag-scaled.ppm`
- `3-multi-image-flag-comments.ppm`
- `3-multi-image-flag-scaled-comments.ppm`
- `3-grey-image-rose.pgm` (grey scale, PGM file)

Sample output for `ClassTimes`

printCourse("COMP102")

```
Classes for course COMP102
==========================
Comp-Lab  Thu     1000    1100    CO219
Comp-Lab  Thu     1000    1100    CO238
Comp-Lab  Thu     1100    1200    CO219
Comp-Lab  Thu     1100    1200    CO238
Comp-Lab  Wed     1300    1400    CO219
Comp-Lab  Wed     1300    1400    CO238
Comp-Lab  Wed     1400    1500    CO219
Comp-Lab  Wed     1400    1500    CO238
Comp-Lab  Wed     1500    1600    CO219
Comp-Lab  Wed     1500    1600    CO238
Comp-Lab  Wed     1600    1700    CO219
Comp-Lab  Wed     1600    1700    CO238
Lecture   Mon     1100    1150    KKLT303
Lecture   Wed     1100    1150    KKLT303
Lecture   Fri 1100    1150    KKLT303
Tutorial  Tue 1610    1700    AM104
==========================
```

printRoom("VS319")

```
Classes in VS319
===============================
DSDN100   on Wed starting 1030
DSDN101   on Fri   starting 830
DSDN101   on Fri   starting 1030
DSDN101   on Tue  starting 930
DSDN101   on Tue  starting 1100
INTA251   on Fri   starting 1240
INTA261   on Thu  starting 1240
LAND211   on Mon starting 930
LAND211   on Thu  starting 930
LAND261   on Tue  starting 1240
===============================
```

printInRoomOnDay("AM102", "Mon")

```
Classes in AM102 on Mon
========================
FEDU101   Tutorial    13:10-14:00
GERM315   Lecture     14:10-15:00
JAPA111   Tutorial    12:00-12:50
MAOR003   Lecture     10:00-11:50
MATH311   Lecture     09:00-09:50
PERF134   Tutorial    15:10-17:00
PERF234   Tutorial    15:10-17:00
PERF334   Tutorial    15:10-17:00
PERF434   Tutorial    15:10-17:00
There are 9 classes in AM102 on Mon
==========================
```

meanClassLength("WIG")

```
Average duration in WIG = 127.59 mins
```

meanClassLength("AD")

```
There were no classes in  AD
```