作业5
计算机程序设计简介：作业5
截止日期为2018年11月19日晚7点
目标

此作业将为您提供编写程序的经验，这些程序可以从文件中读取某些数据并对数据执行某些操作。它还将为您提供更多使用条件和循环的练习。

资源和链接

下载zip文件并解压缩。
提交你的答案。
标记和反馈（如果可用）
与此作业相关的帮助页面。

摘要
ImageRenderer：
 编写程序来渲染ppm图像文件（最简单的图像格式）。
反射。
 写下你对这项任务的反思。

提交
您的ImageRenderer.java版本
您的Reflection.txt文件

概观

该任务涉及两组练习和一个程序（ImageRenderer），它在图形窗格上呈现图像文件。图像文件必须采用纯ppm格式，这是彩色图像最简单的格式。

制备

下载作业5的zip文件并将其解压缩到主文件夹中的Assig5文件夹。它应该包含您要完成的Java程序的模板，以及ImageRenderer程序的数据文件。仔细阅读整个作业，看看你需要做什么。

查看作业4的模型答案，并确保您了解程序的所有组件。另外，从使用文件的讲座中查看代码示例。
图像渲染器

数字图像文件无处不在，我们很高兴地希望我们的计算机能够在各种环境中显示图像。数字图像中的核心信息是图像的每个单独像素的颜色，因此图像文件必须存储该信息。颜色可以由三个数字的三个表示，一个用于颜色的红色，绿色和蓝色分量，因此我们可以通过为每个像素存储三个数字来表示文件中的图像。具体如何我们这样做取决于图像文件的格式。一个非常简单的格式只列出所有数字，因此100x100像素图像（非常小）将有30,000个数字：10,000个像素中的每一个都有三个数字。但是，这将最终成为一个相当大的文件，并且大多数常见的图像文件格式（例如jpg，gif和png）具有更复杂的格式，其中信息被编码和压缩以节省空间并消除冗余。将这些压缩图像文件的内容转换为屏幕上的彩色图像可能是一个非常复杂的过程，但文件可能比使用简单格式时小得多。

对于此分配，您将编写一个程序来以纯ppm格式（“便携式像素图”）呈现图像文件 - 这是最简单（和最低效）文件格式之一。

普通的ppm文件以描述图像的四条信息开始，然后依次为每个像素的颜色值，从图像的左上角开始，沿着每一行从左到右工作，逐行工作图片。

例如，这里有一个简单的ppm图像文件（image-tiny.ppm，它是1-image-bee.ppm文件中蜜蜂背面的一小部分）：

```
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
```

第一个标记是字符串P3，它将该图像文件的格式标识为普通ppm文件。

第二个和第三个标记是图像大小的整数：宽度（像素列数）和高度（像素行数）。在这种情况下，每行有5行，每行12个像素。

第四个标记指定像素的颜色值的最大可能值。在这种情况下，红色，绿色和蓝色分量的颜色值均在0到255之间.0对于该组件没有任何意义; 255表示组件的最大强度。这意味着由三个0组成的颜色将是黑色;三个255是白色的;一个255 0 0将是鲜红色。这意味着有256x256x256 =近1700万种不同的可能颜色。

[如果颜色深度仅为7，那么每个颜色值将介于0和7之间，并且只有8x8x8 = 512种可能的颜色。这将对应于每个组件3比特的“颜色深度”。

文件的其余部分是像素的颜色值，每个像素有三个数字。因此第一行将有36个数字（第3行开头最多97 92 76），第二行有36个数字等。换行并不意味着什么特别的（尽管格式指定了数字应分成不超过70个字符的行）。

您将完成ImageRenderer程序，该程序读取纯ppm文件并在屏幕上呈现它。

核心

完成renderImageCore方法。它应该询问用户一个文件（例如，使用UIFileChooser.open方法），并读取文件，将像素渲染到图形窗格。它应检查文件是否以正确的“幻数”（P3）开头。如果没有，它应该打印一条消息并退出该方法。否则，它应该读取列数和行数。然后它应该读取颜色深度，但它可以简单地忽略它并假设它是“正常”255.然后它应该依次读取每个像素的三个颜色值，设置UI的颜色然后绘制正方形图形窗格上的像素。它需要一个嵌套循环来处理每一行的列，并处理所有行。

图像的左上角应放置在（20,20），每个像素应绘制为2x2正方形（即图像应放大200%）。使用文件顶部定义的常量。

提示：
您将需要一个嵌套循环来执行此操作：遍历每一行的外部循环，以及沿当前行的列的内部循环。
虽然您可以在renderImageCore中实现所有代码，但是如果在renderImageCore中实现文件打开然后使用renderImageHelper实际从扫描程序中绘制图像，则更容易完成。

有几个ppm图像，您可以测试您的方法：
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

这是相同图像的png形式，因此您可以看到它们的外观。

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

1-image-fly:	1-image-rose:

完成

您将完成渲染包含动画图像的ppm文件的renderAnimatedImage方法。

一个ppm文件可能包含一个紧接着的图像序列，代表一个“动画”图像，应该通过绘制第一个图像，暂停，然后在其位置绘制第二个图像，并重复直到所有图像都有被画了。每个图像将具有四个“标题”标记，然后是像素颜色值。这与“动画gif”的过程相同，（虽然gifs编码效率更高，并且有更多选项）。

您的renderAnimatedImage方法应该通过继续从文件渲染图像来处理动画图像，直到文件中没有更多图像。在图像之间使用100 - 200毫秒的延迟。您可以让它只显示一次图像，或重复固定次数，或无限重复（尽管我建议不要使用最后一个选项，因为它会使测试更加困难）。

有三个动画ppm图像：
2-多图像athletes.ppm，
2-多图像ball.ppm，
2-多图像flag.ppm

挑战

真正的普通ppm格式实际上比上面描述的要复杂一些：
前四个令牌不需要放在单独的行上，只要它们被空格或换行符分隔即可。
该文件可以包含呈示器忽略的注释。注释以 # 开头，并持续到行的末尾，并且可以放在第一个标记（P3）和最大颜色值之间的任何位置，例如：

```
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 .....
```

[实际上，评论比这更灵活，但这已经足够了]

此外，普通ppm只是相关格式系列中的一种。 普通pgm（“便携式灰色地图”）用于灰度图像。 pgm文件以标记“P2”开始，并且每个像素由指定像素的灰度级的单个数字（在0和最大颜色值之间）表示。

制作你的节目
处理评论（通过注意并抛弃它们）。
正确处理颜色深度：例如，如果颜色深度为15，则应在设置颜色之前将所有值按比例放大255/15。
识别并渲染普通的pgm图像（使用第一个标记来确定要渲染的版本）。

有几个图像具有这些附加功能，您可以测试您的方法，一些缩放，一些带注释，一个缩放和注释：
3-image-fly-scaled.ppm (colour depth is just 3 bits - max colour value is 7
3-image-fly-comments.ppm and 3-image-fly-comments2.ppm (comments in the header)
3-image-fly-scaled-comments.ppm (scaled and with comments)
3-multi-image-ball-comments.ppm
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)

Assignment 5

Introduction to Computer Program Design: Assignment 5

- Due 19 Nov 2018 7 pm

Goals

This assignment will give you experience in writing programs that read some data from a file and do something with the data. It will also give you more practice using conditionals and loops.

Resources and links

- Download zip file and unzip it.
- Submit your answers.
- Marks and Feedback (When available)
- A help page relevant to this assignment.

Summary

- ImageRenderer:
  ➡ Write a program to render `ppm` image files (the simplest possible image format).
- Reflection.
  ➡ Write up your reflections on this assignment.

To Submit

- Your version of `ImageRenderer.java`
- Your `Reflection.txt` file

Overview

The assignment involves two sets of exercises and a program (`ImageRenderer`) that renders an image file on the graphics pane. The image files must be in plain ppm format, which is the simplest possible format for color images.

Preparation

Download the zip file for assignment 5 and extract it to the `Assig5` folder in your home folder. It should contain templates for the Java program you are to complete, along with data files for the `ImageRenderer` program. Read through the whole assignment to see what you need to do.

Look at the model answers to assignment 4, and make sure you understand all the components of the programs. Also, go over the code examples from the lectures that used files.

Image Renderer

Digital image files are ubiquitous, and we happily expect our computers to show images in all sorts of contexts. 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 a triple of 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, 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 `1-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 identifies the format of this image file as 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 that 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 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 a nested loop 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:

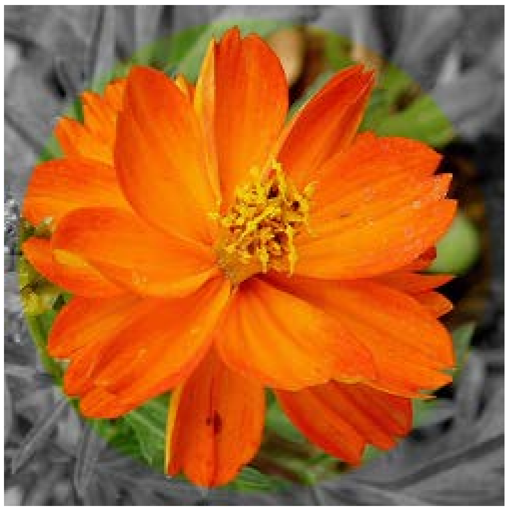
- You will need a nested loop to do this: an outer loop that goes through each row, and an inner loop that goes along the columns of the current row.
- While you can implement all of the code in `renderImageCore`, it will be easier to do the completion if you implement the file opening in `renderImageCore` and then use `renderImageHelper` to actually draw the image from the scanner.

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 may make it display the images just once, or repeat a fixed number of times, or repeat infinitely (though I recommend against the last option since it makes testing more difficult).

There are three animated ppm images:

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

[Actually, the comments are even more flexible than this, but this is good enough]

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, one 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 `3-image-fly-comments2.ppm` (comments in the header)
- `3-image-fly-scaled-comments.ppm` (scaled and with comments)
- `3-multi-image-ball-comments.ppm`
- `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)

Reflection

Answer the following questions in the `Reflection.txt` file, and make sure you submit it.

1. For the Core of ImageRenderer, you didn't need to use while(scan.hasNext()), but for the Completion (animated images) you did need to use while(scan.hasNext()). Explain the difference.
2. If an image was standard SVGA size (800x600), how big would a ppm file of the image be, given that each character in the file (including the spaces) requires 1 byte? How much worse is this than a typical jpg or png file of the same number of pixels? (you may need to go and find some typical images that are about 800x600 pixels).