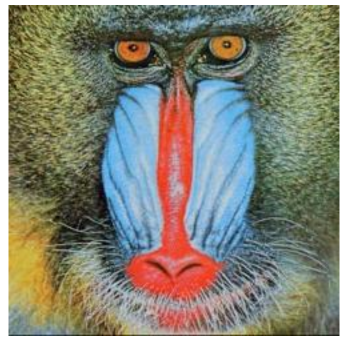
In this assignment you will create a data type called `SeamCarver` that resizes a W-by-H image using the seam-carving technique. Seam carving is a content-aware image resizing technique where the image is reduced in size by one pixel of height (or width) at a time. A *vertical seam* in an image is a path of pixels connected from the top to the bottom with one pixel in each row; *a horizontal seam* is a path of pixels connected from the left to the right with one pixel in each column. Below left is the iconic 298-by-298 pixel image of a Mandrill; below right is the image after removing 50 vertical and horizontal seams. Unlike standard content-agnostic resizing techniques (such as cropping and scaling), seam carving preserves the most interesting features (aspect ratio, set of objects present, etc.) of the image.



Although the underlying algorithm is simple and elegant, it was not discovered until 2007 by Shai Avidan and Ariel Shamir. Now, it is a core feature in many computer graphics applications. Your task is to implement a mutable data type `SeamCarver`, with the following API:

method	description
`public SeamCarver(Picture picture)`	create a `SeamCarver` object based on the given picture
`public Picture picture()`	current picture
`public int width()`	width of current picture
`public int height()`	height of current picture
`public double energy(int x, int y)`	energy of pixel at column x and row y
`public int[] findHorizontalSeam()`	sequence of indices for horizontal seam
`public int[] findVerticalSeam()`	sequence of indices for vertical seam
`public void removeHorizontalSeam(int[] seam)`	remove horizontal seam from current picture
`public void removeVerticalSeam(int[] seam)`	remove vertical seam from current picture

Finding and removing a seam involves three parts and a tiny bit of notation. In image processing, pixel (x, y) refers to the pixel in column x and row y, with pixel $(0, 0)$ at the upper-left corner and pixel $(W-1, H-1)$ at the lower-right corner. This is consistent with the `Picture` data type in `stdlib.jar`. Note that this is the opposite of the standard mathematical notation used in linear algebra, where (i, j) refers to row i and column j and $(0, 0)$ is at the lower-left corner. We also assume that the color of each pixel is represented in RGB space, using three integers between 0 and 255. This is consistent with the `java.awt.Color` data type.

Problem 1. (*Energy Calculation*) The first step is to implement the `energy()` method to calculate the energy of a pixel, which is a measure of its importance — the higher the energy, the less likely that the pixel will be included as part of a seam (as you will see in the next problem). To compute the energy of a pixel, use the *dual-gradient energy function*. The energy of pixel (x, y) is $\Delta_x^2(x, y) + \Delta_y^2(x, y)$, where the square of the x-gradient $\Delta_x^2(x, y) = R_x^2(x, y) + G_x^2(x, y) + B_x^2(x, y)$, and where the central differences $R_x(x, y), G_x(x, y)$, and $B_x(x, y)$ are the absolute value in differences of red, green, and blue components between pixel $(x + 1, y)$ and pixel $(x - 1, y)$. The square of the y-gradient $\Delta_y^2(x, y)$ is defined in an analogous manner. To handle pixels on the borders of the image, calculate energy by defining the leftmost and rightmost columns as adjacent and the topmost and bottommost rows as adjacent. For example, to compute the energy of a pixel $(0, y)$ in the leftmost column, use its right neighbor $(1, y)$ and its "left" neighbor $(W - 1, y)$.

Consider the 3-by-4 image with RGB values (each component is an integer between 0 and 255) as shown in the table below:

(255, 101, 51)	(255, 101, 153)	(255, 101, 255)
(255, 153, 51)	(255, 153, 153)	(255, 153, 255)
(255, 203, 51)	(255, 204, 153)	(255, 205, 255)
(255, 255, 51)	(255, 255, 153)	(255, 255, 255)

- *Non-border pixel example.* The energy of pixel $(1, 2)$ is calculated from pixels $(0, 2)$ and $(2, 2)$ for the x-gradient:

$$R_x(1,2) = 255 - 255 = 0,$$
$$G_x(1,2) = 205 - 203 = 2,$$
$$B_x(1,2) = 255 - 51 = 204,$$

yielding $\Delta_x^2(1,2) = 2^2 + 204^2 = 41620$; and pixels $(1, 1)$ and $(1, 3)$ for the y-gradient

$$R_y(1,2) = 255 - 255 = 0,$$
$$G_y(1,2) = 255 - 153 = 102,$$
$$B_y(1,2) = 153 - 153 = 0,$$

yielding $\Delta_y^2(1,2) = 102^2 = 10404$. Thus, the energy of pixel $(1, 2)$ is $41620 + 10404 = 52024$. Similarly, the energy of pixel $(1, 1)$ is $204^2 + 103^2 = 52225$.

- *Border pixel example.* The energy of the border pixel $(1, 0)$ is calculated by using pixels $(0, 0)$ and $(2, 0)$ for the x-gradient

$$R_x(1,0) = 255 - 255 = 0,$$
$$G_x(1,0) = 101 - 101 = 0,$$
$$B_x(1,0) = 255 - 51 = 204,$$

yielding $\Delta_x^2(1,0) = 204^2 = 41616$; and pixels $(1, 3)$ and $(1, 1)$ for the y-gradient

$$R_y(1,0) = 255 - 255 = 0,$$
$$G_y(1,0) = 255 - 153 = 102,$$
$$B_y(1,0) = 153 - 153 = 0,$$

yielding $\Delta_y^2(1,2) = 102^2 = 10404$. Thus, the energy of pixel $(1, 2)$ is $41616 + 10404 = 52020$.

The energies for all the pixels of the above 3-by-4 image are show below:

20808.0	52020.0	20808.0
20808.0	52225.0	21220.0
20809.0	52024.0	20809.0
20808.0	52225.0	21220.0

The client `PrintEnergy` takes the name of an image as a command-line argument and prints energy calculated for each pixel.

```
$ java PrintEnergy data/6x5.png
6-by-5 image
Printing energy calculated for each pixel.
    57685      50893      91370      25418      33055      37246
    15421      56334      22808      54796      11641      25496
    12344      19236      52030      17708      44735      20663
    17074      23678      30279      80663      37831      45595
    32337      30796       4909      73334      40613      36556
```

Problem 2. (*Seam Identification*) The next step is to implement `findVerticalSeam()` to find a vertical seam of minimum total energy — implementing `findHorizontalSeam()` to find a horizontal seam is analogous. This is similar to the classic shortest path problem in an edge-weighted digraph, but there are three important differences:

- The weights are on the vertices instead of the edges.
- The goal is to find the shortest path from any of the W pixels in the top row to any of the W pixels in the bottom row.
- The digraph is acyclic, where there is a downward edge from pixel (x, y) to pixels $(x-1, y+1)$, $(x, y+1)$, and $(x+1, y+1)$, assuming that the coordinates are in the prescribed ranges.

Seams cannot wrap around the image (e.g., a vertical seam cannot cross from the leftmost column of the image to the rightmost column).

The `findVerticalSeam()` method returns an array of length H such that entry i is the column number of the pixel to be removed from row i of the image. For example, consider the 6-by-5 image below (supplied as `6x5.png`).

(78, 209, 79)	(63, 118, 247)	(92, 175, 95)	(243, 73, 183)	(210, 109, 104)	(252, 101, 119)
(224, 191, 182)	(108, 89, 82)	(80, 196, 230)	(112, 156, 180)	(176, 178, 120)	(142, 151, 142)
(117, 189, 149)	(171, 231, 153)	(149, 164, 168)	(107, 119, 71)	(120, 105, 138)	(163, 174, 196)
(163, 222, 132)	(187, 117, 183)	(92, 145, 69)	(158, 143, 79)	(220, 75, 222)	(189, 73, 214)
(211, 120, 173)	(188, 218, 244)	(214, 103, 68)	(163, 166, 246)	(79, 125, 246)	(211, 201, 98)

The corresponding pixel energies are shown below, with a minimum energy vertical seam highlighted in pink. In this case, the method `findVerticalSeam()` returns the array `{3, 4, 3, 2, 2}`.

57685.0	50893.0	91370.0	25418.0	33055.0	37246.0
15421.0	56334.0	22808.0	54796.0	11641.0	25496.0
12344.0	19236.0	52030.0	17708.0	44735.0	20663.0
17074.0	23678.0	30279.0	80663.0	37831.0	45595.0
32337.0	30796.0	4909.0	73334.0	40613.0	36556.0

The client `PrintSeams` takes the name of an image as a command-line argument and prints the minimum-energy horizontal and vertical seams, along with their energy values.

```
$ java PrintSeams data/6x5.png
6-by-5 image

Horizontal seam:
 57685   50893   91370   25418   33055   37246
 15421   56334   22808*  54796   11641*  25496
 12344*  19236*  52030   17708*  44735   20663*
 17074   23678   30279   80663   37831   45595
 32337   30796    4909   73334   40613   36556

Total energy = 104400


Vertical seam:
 57685   50893   91370   25418*  33055   37246
 15421   56334   22808   54796   11641*  25496
 12344   19236   52030   17708*  44735   20663
 17074   23678   30279*  80663   37831   45595
 32337   30796    4909*  73334   40613   36556

Total energy = 89955
```

Problem 3. (*Seam Removal*) The final step is to implement `removeVerticalSeam()` to remove from the image all of the pixels along the vertical seam — implementing `removeHorizontalSeam()` to remove from the image all of the pixels along the horizontal seam is analogous.

The client `RemoveSeams` takes as command-line arguments the name of an image and the number of vertical and horizontal minimum-energy seams to remove, removes those seams from the image, and prints the pixel energies of the resized image.

```
$ java RemoveSeams data/6x5.png 1 1
5-by-4 image
Printing energy calculated for each pixel.
    57685      50893      49196      45397      37246
    18803      33246       9172      17549      33926
     8192      58360      11431      37831      42155
    32337      29222      26170      40613      36556
```

Implementation Details

- The data type must not mutate the `Picture` argument to the constructor.
- Your code should throw an exception when called with invalid arguments, as documented here:
  - By convention, the indices x and y are integers between 0 and $W-1$ and between 0 and $H-1$, respectively, where W is the width and H is the height of the current image. Throw a `java.lang.IndexOutOfBoundsException` if `energy()` is called with either an x-coordinate or y-coordinate outside its prescribed range.
  - Throw a `java.lang.NullPointerException` if either `removeVerticalSeam()` or `removeHorizontalSeam()` is called with a `null` argument.
  - Throw a `java.lang.IllegalArgumentException` if either `removeVerticalSeam()` or `removeHorizontalSeam()` is called with an array of the wrong length or if the array is not a valid seam (either an entry is outside the height/width bounds or two adjacent entries differ by more than 1).
  - Throw a `java.lang.IllegalArgumentException` if either `removeVerticalSeam()` or `removeHorizontalSeam()` is called when the width or height of the current picture is 1, respectively.
- The `width()`, `height()`, and `energy()` methods should take constant time in the worst case. All other methods should run in time proportional to WH (or better) in the worst case.
- To implement `findHorizontalSeam()` and `removeHorizontalSeam()`, transpose the picture and call `findVerticalSeam()` and `removeVerticalSeam()`. Don't forget to transpose the picture back, when needed.

Data Under the `data` directory, we provide several sample input files for testing, along with some reference solutions.

Visualization Clients In addition to the client programs described above, you may use the following visual client programs to test and debug your code:

- `ShowEnergy` takes the name of an image as a command-line argument and displays the pixel energies on the screen.

```
$ java ShowEnergy data/mandrill.jpg
298-by-298 image
Displaying energy calculated for each pixel.
```

- `ShowSeams` takes the name of an image as a command-line argument and displays the minimum-energy horizontal and vertical seams on the screen.

```
$ java ShowSeams data/mandrill.jpg
298-by-298 image
Displaying horizontal seam calculated.
Displaying vertical seam calculated.
```

Files to Submit:

1. `SeamCarver.java`
2. `report.txt`

Before you submit:

- Make sure your programs meet the input and output specifications by running the following command on the terminal:

```
$ python run_tests.py -v [<problems>]
```

where the optional argument `<problems>` lists the problems (`Problem1`, `Problem2`, etc.) you want to test; all the problems are tested if no argument is given.

- Make sure your programs meet the style requirements by running the following command on the terminal:

```
$ check_style <program>
```

where `<program>` is the `.java` file whose style you want to check.

Acknowledgements This project is an adaptation of the Seam Carving assignment developed at Princeton University by Josh Hug, Maia Ginsburg, and Kevin Wayne.