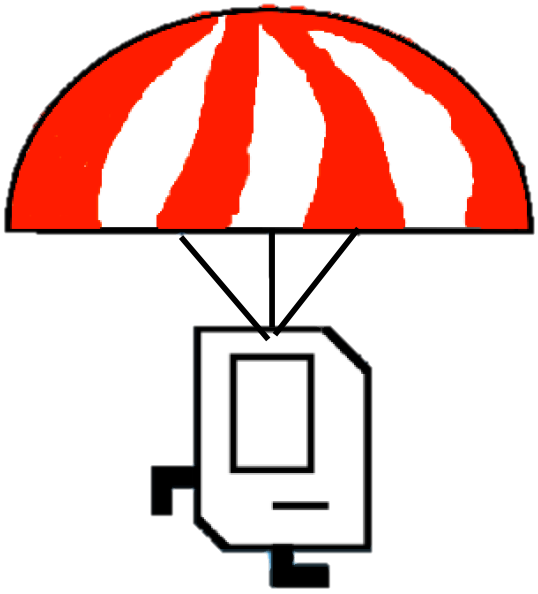
Sarai Gould & Laura Cruz-Albrecht
CS106A
Summer 2019
Aug 8th, 2019

CS106A Practice Final 1

This handout is intended to give you practice solving problems that are comparable in format and difficulty to those which will appear on the final examination. We do not guarantee that the number of questions in this sample exam will match the number on the real exam, nor that every kind of problem shown here will exactly match the kinds of problems shown on the final exam (though the real exam will be generally similar overall).

General instructions

The exam is 3 hours. Answer each of the questions included in the exam. Unless otherwise indicated as part of the instructions for a specific problem, comments will not be required on the exam. Uncommented code that gets the job done will be sufficient for full credit on the problem. On the other hand, comments may help you to get partial credit if they help us determine what you were trying to do.



Problem 1: Short Answer (34 points)

1a) High Five (7 points)

Write a loop that prints out all multiples of five between 0 and 100 inclusive, from largest to smallest. You should print one number per line. i.e. print: 100, 95, 90, and so on, until 0.

```
public void run() {









}
```

1b) Keys to the Map (7 points)

Write a method **`printKeys`** that prints all of the keys in a given `HashMap`, one per line.

```
private void printKeys(HashMap<String, String> map) {









}
```

1c) LargestLetter (10 points)

Write a method **`largestLetter`** that takes in a string that contains only lowercase letters and returns the "largest letter" in that string. We define the largest letter to be the one that is furthest along in the alphabet (eg 'z' is the largest possible letter and 'a' is the smallest possible). You can assume that the input string is at least one character long.

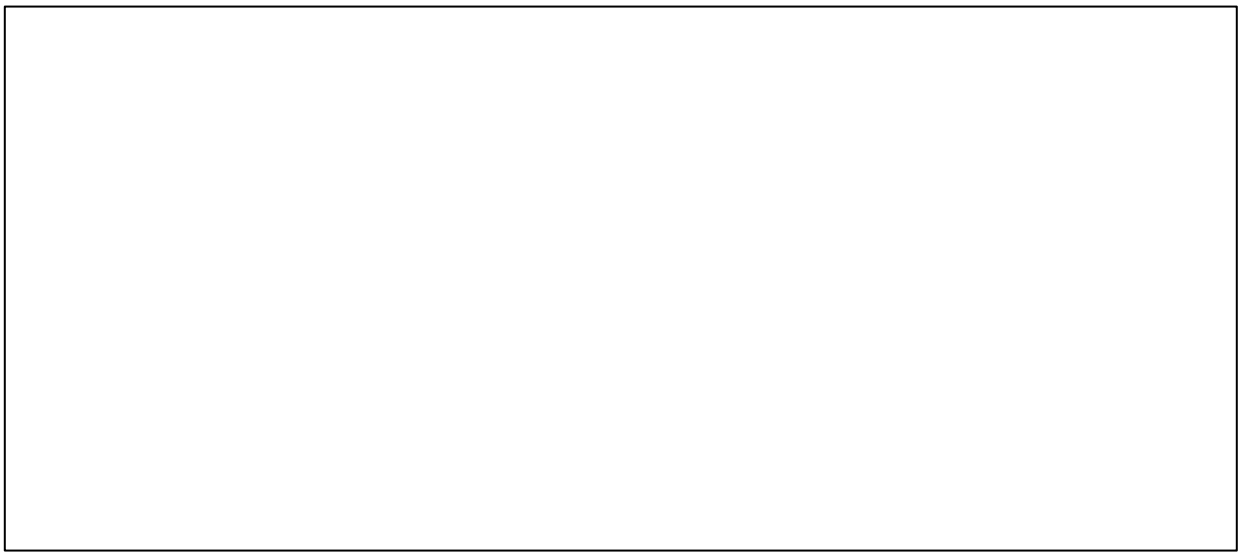
As an example: `largestLetter("helloworld")` would return `'w'`

```
private char largestLetter(String str) {




}
```

1d) Trace the following method (10 points)

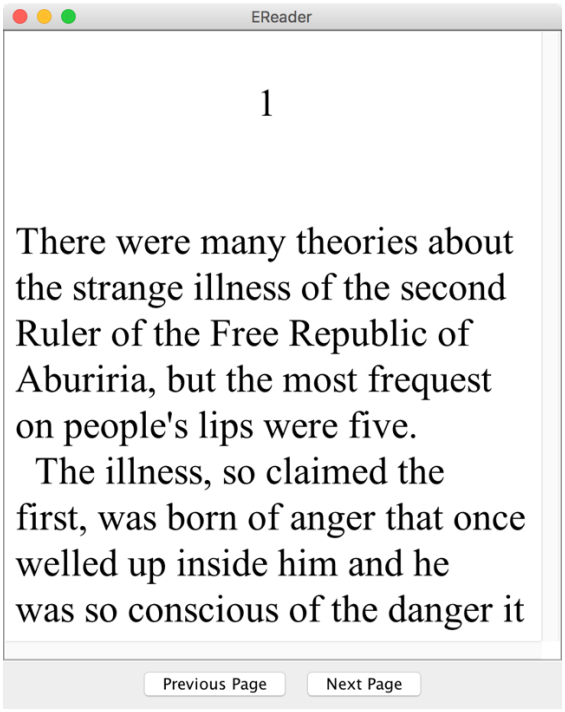
```
public void run() {
   int[] flowers = new int[2];
   flowers[0] = 1;
   flowers[1] = 2;
   int[] bees = new int[2];
   bees[0] = 2;
   bees[1] = 7;
   pollinate(flowers, bees);
   // what is the value of flowers here?
}

private void pollinate(int[] a, int[] b){
   for(int i = 0; i < a.length; i++) {
      a[i] += b[i]/2;
      allogamy(a[i]);
   }
}

private void allogamy(int v){
   v *= 2;
}
```

Draw the value of flowers just before the run method returns:



Problem 2: Kindle (18 points)

Write a `ConsoleProgram` that allows the user to read through a 100 page novel, "The Wizard of the Crow" by Ngũgĩ wa Thiong'o. The user can turn the page, or go back, using buttons on the south of the console.



The 100 page book is stored in 100 files named `"page1.txt"`, `"page2.txt"` and so on, up until the last file, `"page100.txt"`. Use a method that we define for you:

```
private void printFile(fileName)
```

That clears the console and prints out the contents of the file in Times New Roman font. You do not need to define `printFile` and you should not write any file reading code for this problem.

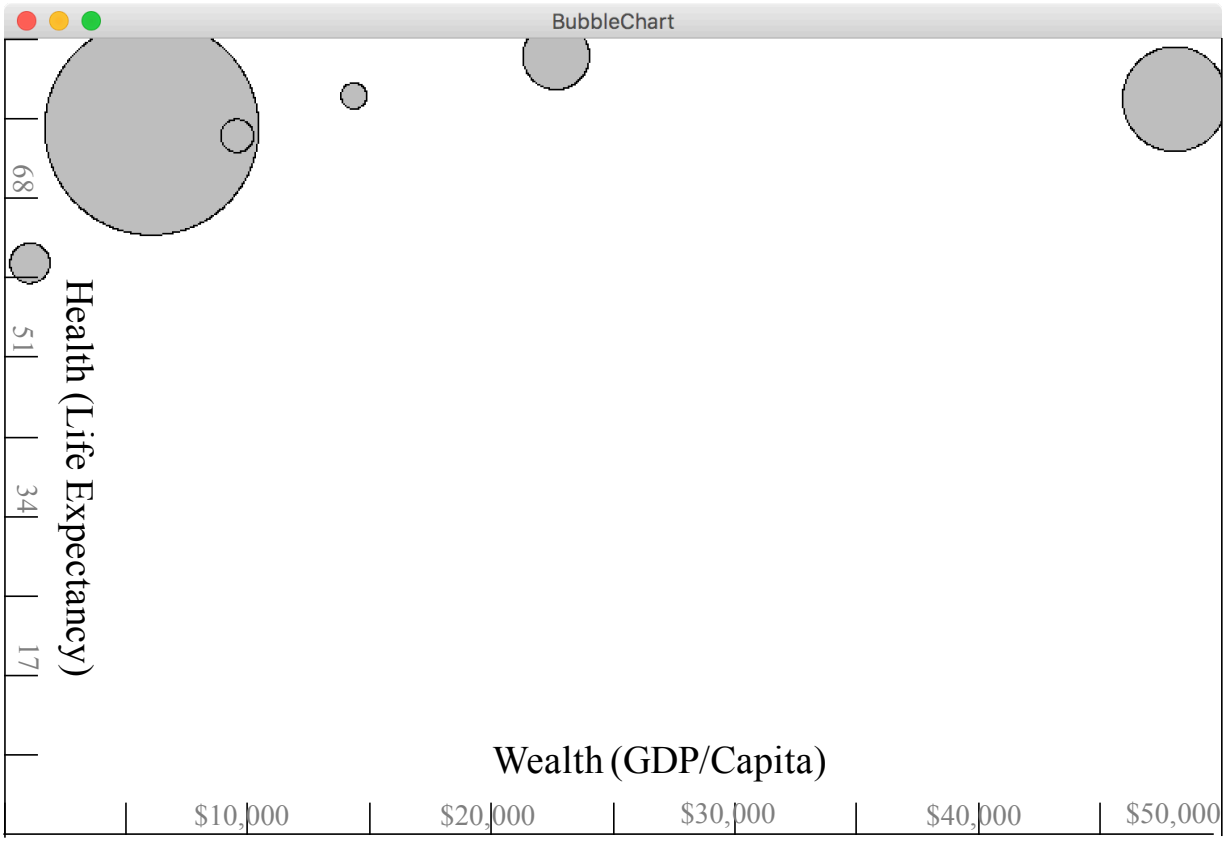
The user starts on the first page and as such you should display the contents of page1.txt when your program launches. The button "Previous Page" brings the user to the page before their current page. For example, if the user was on page 5, after pressing Previous Page the user would be on page 4. The button "Next Page" makes the user's current page number increase. When the page changes, print the contents of the new page.

Important: If the user is on the first page the Previous Page button should do nothing. Similarly if the user is on the last page (page 100) Next Page should do nothing.

```
public class Kindle extends ConsoleProgram {
```

Problem 3: Changing Mindsets (22 points)

Write a `GraphicsProgram` that can draw a "bubble plot" of Weath vs Health of countries. Similar visualizations have helped global health researchers communicate that in the last 50 years many countries have become relatively healthy, despite disparities in wealth.



Read in a file "`2015.txt`" that stores country data, one country per line. Each line has four fields: *CountryName, Wealth, Health, and Population (in millions of people)*

```
China  0.12  0.89  1357.0
Kenya  0.02  0.72  44.35
M'sia  0.19  0.88  29.72
U.S.A  0.96  0.93  318.9
Japan  0.45  0.98  127.3
Chile  0.27  0.93  17.6
```

There is one space between each field.

Both *Wealth* and *Health* are fractional values in the range (0.0 to 1.0). Zero means no health (or wealth) and 1.0 is the maximum health (or wealth) that can be displayed on this chart. While the data are all expresed in consistent units, you **do not** need to consider units in your solution.

Recall that each line in the file represents a single country. For each line in the file add a single circle (aka "bubble") to the screen with the following properties:

- The number of pixels from the **left** of the screen to the center of the circle is equal to the country's wealth multiplied by the width of the screen.
- The number of pixels from the **bottom** of the screen to the center of the circle is equal to the country's health multipled by the height of the screen.
- The circle's area in pixels is equal to the country's population value (in millions, as it appears in the file).
- Unlike in the picture, your circle does **not** need to be filled.

You do not need to draw either axis or axis-labels, you only need to draw the bubbles.

For each bubble, you are asked to draw a circle with a given area. Since `GOval`s are defined by width and height, not area, this is going to require some simple geometry. Recall that:

the area of a circle = $\pi \bullet \text{radius}^2$

If you know the area and would like to calculate the radius, you can use the inverse formula:

```
radius = Math.sqrt(area / Math.PI);
```

This space is left blank for notes. Please write your solution in the next page

Professor Hans Rosling popularized this visualization in his talk "Let My Dataset Change Your Mindset." Rosling argued that popular contemporary perception of global health reflects the realities of 1970 rather than the realities of 2017.

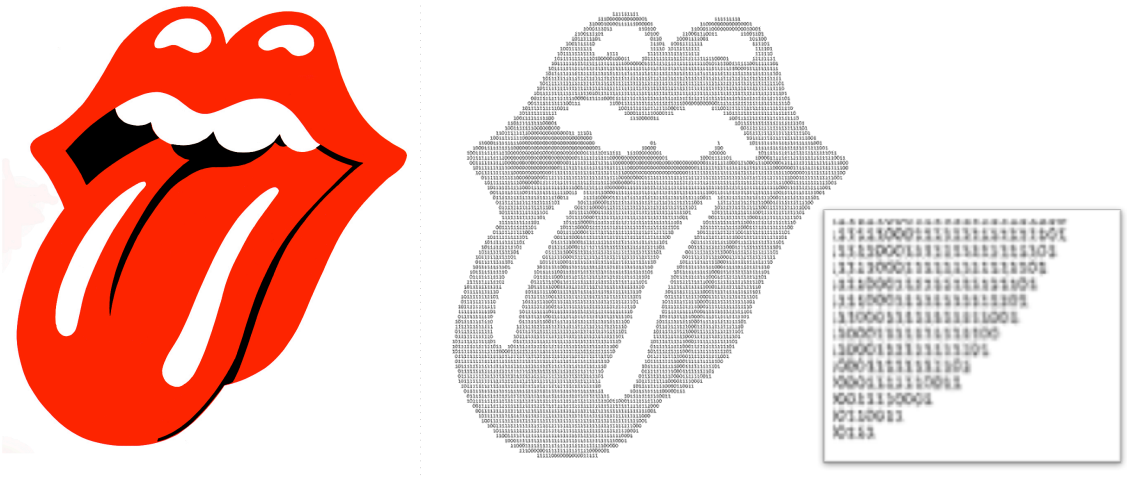
```
public class ChangingMindsets extends GraphicsProgram {
```

Problem 4: ASCII Art (24 points)

ASCII art is a graphic design technique that presents Strings that look like images. Write a method **`makeAscii`** that turns a **`GImage`** into an array of Strings that look like the image.

```
private String[] makeAscii(GImage img)
```

Original ASCII Art



The figure on the right looks like an image, but if you inspect it closely you will see that it is composed of many strings each of which use the characters `'1'` `'0'` and `' '`.

The parameter **`img`** is the input image. For simplicity, assume that there is a method:

```
double[][] brightness = img.getPixelBrightness();
```

that returns a matrix of doubles, one for each pixel. The brightness matrix has exactly the same number of rows and columns as the original image. Each value in the brightness matrix is in thee ange (0.0, 1.0) where 1.0 means that the pixel is white and 0.0 means this pixel is black. For each pixel you will chose a single char to represent it (based on the pixel's brightness).

You should return an array of Strings that has one String for each row in the image. The first String in the array will represent the top row of pixels, the second will correspond to the second row of pixels etc. Each String will have one character for each pixel in its row.

For each pixel chose one character to add to the corresponding row String.

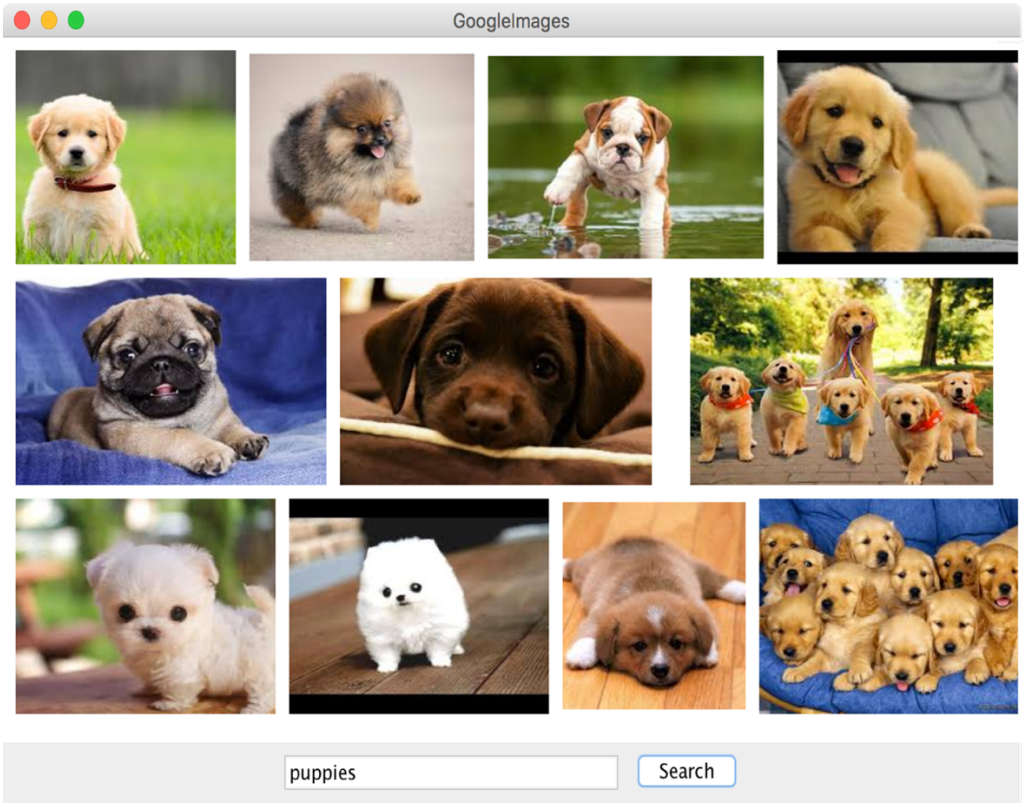
If the pixel has a brightness greater than 0.66, add the character `' '`.

Else, if the brightness is greater than 0.33, add the character `'1'`.

Else, the pixel is quite dark and you should add the character `'0'`.

```
private String[] makeAscii(GImage img) {
```

Problem 5: Google Images (30 points)



Write a `GraphicsProgram` that displays Google image search results. Your program should have both a text field where the user can enter search "queries" and a Search button. When the Search button is pressed, call the method `getSearchResults` that we provide to you:

```
ArrayList<GImage> results = getSearchResults(query);
```

The query parameter is the string the user entered in the textfield. The method is guaranteed to return exactly 100 search result images in ranked order (so the first image is the most important one to show to the user). You do not need to define this method!

You should display your images in three rows of fixed height `ROW_HEIGHT`. Layout the result images one at a time from left to right. When you run out of space on a row, move on to the next row. You can change the size of a `GImage` using it's **`setSize`**(*width*, *height*) method.

"a

Not all images have the same "aspect ratio," some are wider, some are taller. If you make an image's height equal to `ROW_HEIGHT` and make the width equal to:

`ROW_HEIGHT` * *imageWidth* / *imageHeight*,

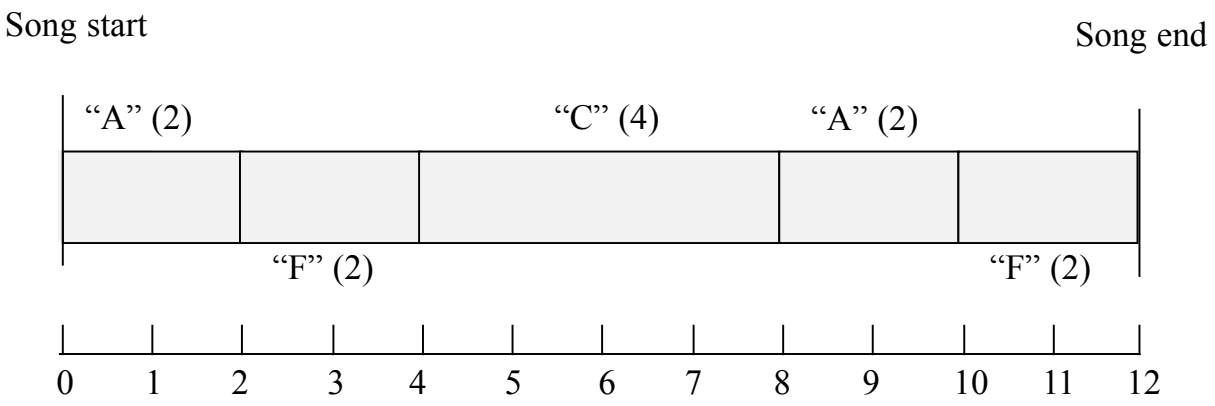
the image's aspect ratio will be preserved. There is a spacing of `GAP` pixels between each picture. You can optionally include the `GAP` between the pictures and the border of the window.

No image should go off the screen. You should not display all 100 returned images – only display the ones that fit into the three rows. ***Hint***: when laying out images it might help to keep track of your currentX, currentY, your current image index and your current row.

```
public class GoogleImages extends GraphicsProgram {
   private static final int ROW_HEIGHT = 300;
   private static final int GAP = 20;
   private static final int TEXT_FIELD_SIZE = 20;
```

Problem 6: Song Composer (30 points)

Let a song be defined to be a sequence of notes, where each note has a note name (for example "A", "C") and a duration, measured in beats[1]. The following is the start of Hold Up written as such a sequence of notes (with beat durations in parenthesis):



Hold Up starts with

- "A" for two beats
- "F" for two beats
- "C" for four beats
- "A" (again) for two beats
- "F" (again) for two beats

In the visualization, the numbers at the bottom represent how many "beats" into the song we are. The total number of beats in this song-in-progress is 12.

It is much easier to write a program that can compose a song if we first define a few new classes.

This space is left blank for notes. Please write your solution on the following pages

[1] A beat is a basic unit of time in a song. Often it is around 200 milli seconds.

```
// Write a class “Note” which has both a note name and a duration
public class Note {









   // the constructor
   public Note(String name, int duration) {









   }


   // returns the note’s name
   public String getName() {









   }

   // returns the note’s duration
   public int getDuration() {









   }
}
```

```
// Write a song class that keeps track of a sequence of
// notes. Specifically it should support the methods
// addNote, getNoteAtTime, and getSongLength.
public class Song {

   // the constructor
   public Song() {

   }
```

Continue on the next page…

```
// appends a new note to the song
public void addNote(Note newNote) {









}

// returns the total length of the song (in number of beats)
// note that number of beats does not equal number of notes.
public int getSongLength() {









}
```

Continue on the next page…

```
// returns the note name this many beats into the song. If
// the notes in the example had been added to a song instance
// song.getNoteAtTime(4) would return “C”
// song.getNoteAtTime(0) would return “A”
// song.getNoteAtTime(3) would return “F”
// if time is >= getSongLength() return the empty string.
public String getNoteAtTime(int time) {









}
```

Problem 7: Join (22 points)

Sometimes HashMaps get lonely. Write a method **`join`**:

```
private HashMap<String, double[]> join(
          HashMap<String, Double> a,
          HashMap<String, Double> b);
```

That takes in two HashMaps **`a`** and **`b`** (that both associate Strings with Doubles) and produces a third HashMap that associates any keys shared by both maps with an array that contains the value for the key in **`a`** and the value for the key in **`b`**.

For example: say you have the two HashMaps populationMap and lifeExpectancyMap, both of which have country names as keys. The former associates countires with their populations (in millions of people), the latter associates countries with average life expectancy (in years)

populationMap

Key	Value
China	1357.0
Kenya	44.35
Malaysia	29.72
U.S.A.	318.9
Chile	17.6

lifeExpectancyMap

Key	Value
China	75.2
Kenya	61.1
Malaysia	74.8
U.S.A.	78.7
Japan	83.1

Calling **`join`** on populationMap and lifeExpectancyMap would combine the data into a single HashMap:

join(populationMap, lifeExpectancyMap)

Key	Value
China	[1357.0, 75.2]
Kenya	[44.35, 61.1]
Malaysia	[29.72, 74.8]
U.S.A.	[318.9, 78.7]

The values in the new HashMap are arrays of length two where the first element is the value from **`a`** and the second element is the value from **`b`**. For example China was associated with 1357.0 in the first map and 75.2 in the second map. In the joined map China is associated with an array [1357.0, 75.2]. Note that neigher Chile nor Japan are present in the joined map (only keys that show up in ***both*** original maps are included).

```
private HashMap<String, double[]> join(
                      HashMap<String, Double> a,
                      HashMap<String, Double> b) {
```