Final Review Session #2

Brahm Capoor

First, a quick review of last time

A problem: The Stanford Carriage Pact ☜(゚ヮ゚☜) (☞゚ヮ゚)☞

Suppose we have a bunch of Stanford Students who want to go to a Masquerade Ball, and a bunch of carriages of variable size that can take them there. How can we assign the students to these carriages?

```
ArrayList<String> students = // {“Brahm”, “Kate”, “Zach”, “Jade”, “Mellany”, “Andrew”}
ArrayList<Integer> capacities = // {1, 3, 2}
printAssignments(students, capacities);
```

```
outputs:
Brahm is in carriage 0, which has Brahm
Kate is in carriage 1, which has Kate, Zach, Jade
Zach is in carriage 1, which has Kate, Zach, Jade
Jade is in carriage 1, which has Kate, Zach, Jade
Mellany is in carriage 2, which has Mellany, Andrew
Andrew is in carriage 2, which has Mellany, Andrew
```

```java
private void printAssignments(ArrayList<String> students, ArrayList<Integer> capacities) {
    HashMap<String, Integer> studentsToCarriages = new HashMap<String, Integer>();
    ArrayList<ArrayList<String>> carriages = new ArrayList<ArrayList<String>>();

    ArrayList<String> currentCarriage = new ArrayList<String>(); // represents current carriage
    int currCarriageIdx = 0;                                     // represents current carriage number

    for (int i = 0; i < students.size(); i++) {                  // go through each student
        String currStudent = students.get(i);
        studentsToCarriages.put(currStudent, currCarriageIdx); // student goes in current carriage
        currentCarriage.add(currStudent);                      // add the student to the carriage
        if (currentCarriage.size() == capacities.get(currCarriageIdx)) { // carriage is full
            carriages.add(currentCarriage);   // carriages is the list of all the full carriages
            currentCarriage = new ArrayList<String>(); // get a new carriage
            currCarriageIdx++;       // increment current carriage number
        }
    }

    for (int i = 0; i < students.size(); i++) { // for each student, print which carriage they're in
        String currStudent = students.get(i);
        int carriage = studentsToCarriages.get(currStudent);
        ArrayList<String> studentsInCarriage = carriages.get(carriage);
        println(currStudent + carriage + studentsInCarriage); // print all students in carriage
    }
}
```

Matrices

Number of rows

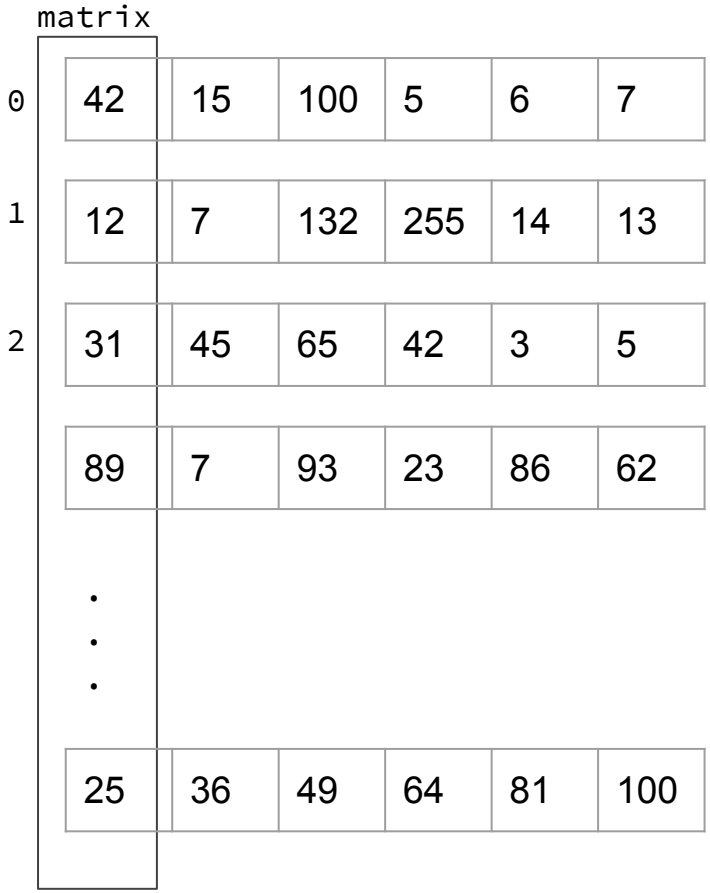
```
int[][] matrix = new int[10][6];
```

Number of columns

```java
int[][] matrix = new int[10][6];
```

42	15	100	5	6	7
12	7	132	255	14	13
31	45	65	42	3	5
89	7	93	23	86	62
64	3	38	32	79	50
161	80	27	82	81	84
228	106	107	103	109	221
140	110	227	144	105	101
27	64	125	4	9	16
25	36	49	64	81	100

```
int[][] matrix = new int[10][6];
```

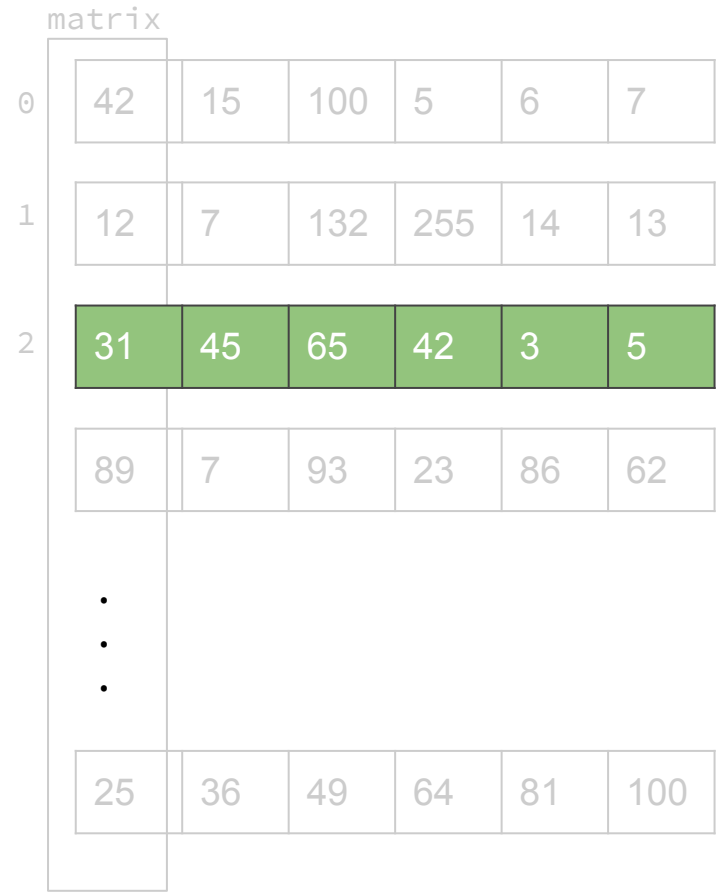


```
int[][] matrix = new int[10][6];
int n = matrix[2][3];
```

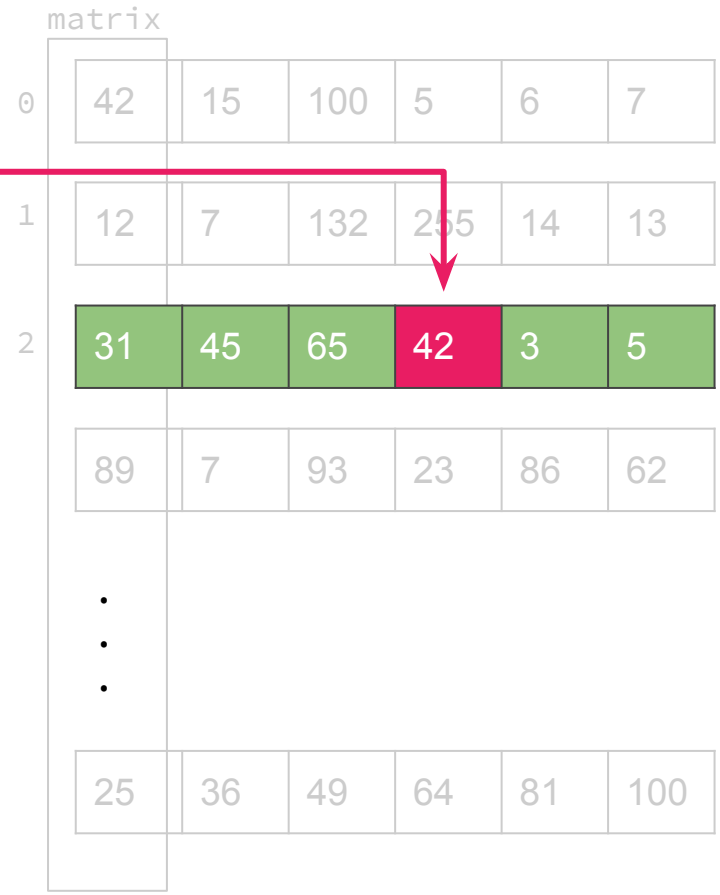
matrix

0	42	15	100	5	6	7
1	12	7	132	255	14	13
2	31	45	65	42	3	5
	89	7	93	23	86	62
	. . .					
	25	36	49	64	81	100

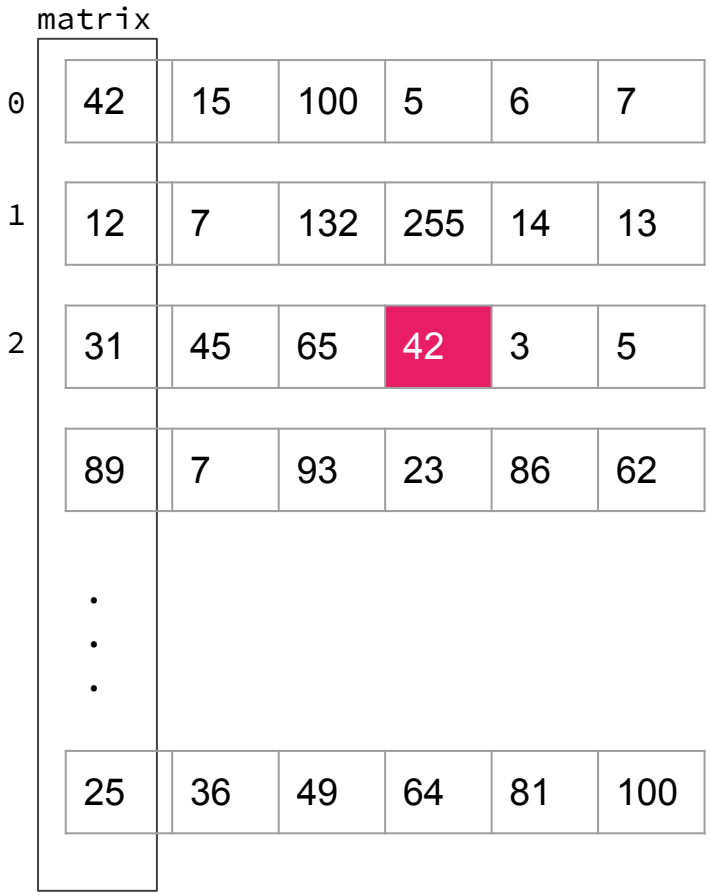
```
int[][] matrix = new int[10][6];
int n = matrix[2][3];
```



```
int[][] matrix = new int[10][6];
int n = matrix[2][3];
```



```
int[][] matrix = new int[10][6];
int n = matrix[2][3]; // 42
```



A common pattern in matrix problems

```java
String[][] matrix = /* a matrix of arbitrary size */

for (int r = 0; r < numRows(matrix); r++) {
    for (int c = 0; c < numCols(matrix); c++) {
        String elem = matrix[r][c];
        // process elem
    }
}
```

```java
private int numRows(int[][] m) {
    return m.length;
}

private int numCols(int[][] m) {
    return m[0].length;
}
```

A problem: Verifying a magic square

A magic square is an n x n grid containing integers whose rows, columns and diagonals all add up to the same number. Write the following method:

```
private boolean isMagicSquare(int[][] grid)
```

that takes in a matrix of ints (which is a square of arbitrary size) and returns whether or not it is a magic square.

8	11	14	1
13	2	7	12
3	16	9	6
10	5	4	15

```java
private boolean isMagicSquare(int[][] grid) {
    int total = rowSum(grid, 0);

    for (int i = 1; i < grid.length; i++) {
        if (total != rowSum(grid, i)) {
            return false;
        }
    }

    for (int i = 0; i < grid[0].length) {
        if (total != colSum(grid, i)) {
            return false;
        }
    }

    if (total != mainDiagonalSum(grid) || total != secondDiagonalSum(grid)) {
        return false;
    }

    return true;
}
```

```java
private int rowSum(int[][] grid, int rowNum) {
      int sum = 0;
      for (int col = 0; col < grid[rowNum].length; col++) {
            sum += grid[rowNum][col];
      }
      return sum;
}

private int colSum(int[][] grid, int rowNum) {
      int sum = 0;
      for (int row = 0; row < grid.length; row++) {
            sum += grid[row][colNum];
      }
      return sum;
}
```

```java
private int mainDiagonalSum(int[][] grid) {
      int sum = 0;
      for (int i = 0; i < grid.length; i++) {
            sum += grid[i][i];
      }
      return sum;
}

private int secondDiagonalSum(int[][] grid) {
      int sum = 0;
      for (int i = 0; i < grid.length; i++) {
            sum += grid[i][grid.length - 1 - i];
      }
      return sum;
}
```

Implementing Classes

I'm defining a thing called `Classname`

```
public class ClassName {

   // sick code here

}
```

```
public class Student {

   // sick code here

}
```

Student.java

```java
public class Student {

    // sick code here

}
```

Stanford.java

Creating objects of type Student

```java
public void run() {
    Student s1;
    Student s2;
    Student s3;
    // more sick code here
}
```

Instance variables

Defined as part of a class, but not within any particular method

s1, s2 and s3 all have their own independent properties, encoded as private instance variables

```
public class Student {

    private String studentName;
    private int studentId;
    private String email;
    private int numUnits;
    private boolean isInternational;
}
```

```
public void run() {

    Student s1;
    Student s2;
    Student s3;

}
```

Initializing your instance variables in the constructor

```
public class Student {

    /* instance variables go here */

    public Student(String name, int id, String email,
                   int numUnits, boolean isInternational) {
        studentName = name;
        studentId = id;
        this.email = email; // to disambiguate between variables
        this.numUnits = numUnits;
        this.isInternational = isInternational;
    }
}
```

Now we can make students!

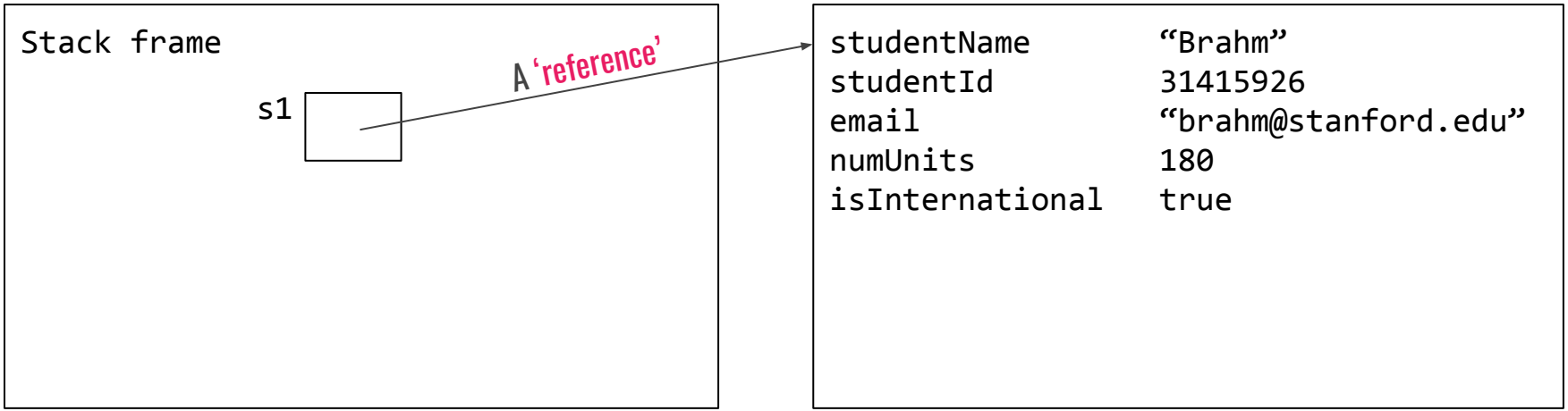
```
public Student(String name, int id, String email,
                    int numUnits, boolean isInternational) {...}
```

```
public void run() {

    Student s1 = new Student(“Brahm”, 31415926, “brahm@stanford.edu”,
                                180, true);
}
```

Under the hood

```
Student s1 = new Student("Brahm", 31415926, "brahm@stanford.edu", 180, true);
```



Getters and Setters

```java
public class Student {

    public Student(int unitCount) {
        numUnits = unitCount;
    }









    private int numUnits;

}
```

```java
public void run() {

    Student s1 = new Student(42);

}
```

Getters and Setters

```java
public class Student {

    public Student(int unitCount) {
        numUnits = unitCount;
    }

    public int getUnits() {
        return numUnits;
    }





    private int numUnits;

}
```

```java
public void run() {

    Student s1 = new Student(42);



    println("Curr:" + s1.getUnits());



}
```

Getters and Setters

```java
public class Student {

    public Student(int unitCount) {
        numUnits = unitCount;
    }

    public int getUnits() {
        return numUnits;
    }

    public void setUnits(int newUnits) {
        numUnits = newUnits;
    }

    private int numUnits;

}
```

```java
public void run() {

    Student s1 = new Student(42);



    println("Curr:" + s1.getUnits());



    s1.setUnits(60);



}
```

Getters and Setters

```
public class Student {

    public Student(int unitCount) {
        numUnits = unitCount;
    }

    public int getUnits() {
        return numUnits;
    }

    public void setUnits(int newUnits) {
        numUnits = newUnits;
    }

    private int numUnits;

}
```

Getter and Setter methods are public (exported) so we can call them in other classes and programs

Getters and Setters

```
public class Student {

    public Student(int unitCount) {
        numUnits = unitCount;
    }

    public int getUnits() {
        return numUnits;
    }

    public void setUnits(int newUnits) {
        numUnits = newUnits;
    }

    private int numUnits;

}
```

Getter and Setter methods are public (exported) so we can call them in other classes and programs

Define Getters and Setters whenever you want to grant a client access to or control over an instance variable

Getters and Setters

```
public class Student {

    public Student(int unitCount) {
        numUnits = unitCount;
    }

    public int getUnits() {
        return numUnits;
    }

    public void setUnits(int newUnits) {
        numUnits = newUnits;
    }

    private int numUnits;

}
```

Getter and Setter methods are public (exported) so we can call them in other classes and programs

Define Getters and Setters whenever you want to grant a client access to or control over an instance variable

These methods are typically very short

Why stop there?

Now that we know how to use instance variables, we can do even cooler things

```
public boolean canGraduate() {
    return numUnits >= 180;
}
```

```
public void dropClass (int classUnits) {
    if (classUnits <= 5) {
        numUnits -= classUnits;
    }
}
```

Methods allow us to define behaviours for our classes

Let's write a class called **Airplane** that implements functionality for boarding/unboarding passengers from a plane.

```
int capacity = readInt("Capacity? ");
Airplane plane = new Airplane(capacity);

// Board passengers
while (!plane.isFull()) {
    String passengerName = readLine("Name: ");
    boolean priority = readBoolean("Priority? (true/false) ");
    plane.boardPassenger(passengerName, priority);
}

// fly...

// Unboard passengers
while (!plane.isEmpty()) {
    String passengerName = plane.unboardPassenger();
    println("Unboarded " + passengerName);
}
```

Let's write a class called **Airplane** that implements the following functionality for boarding/unboarding passengers from a plane.

```
// Creates a new airplane with the given capacity
public Airplane(int capacity);

/* Boards 1 passenger, at front if they are priority, or
 * back otherwise */
public void boardPassenger(String name, boolean priority);

public boolean isFull();
public boolean isEmpty();

/* Unboards and returns next passenger, or null if there
 * are no more passengers. */
public String unboardPassenger();
```

Step #1: decide on instance variables

```
public class Airplane {
    private ArrayList<String> passengers;
    private int capacity;
```

Step #2: Using those instance variables, write public methods

```
public void boardPassenger(String name, boolean priority) {
    if (priority) {
        passengers.add(0, name);
    } else {
        passengers.add(name);
    }
}
...
```

Step #2: Using those instance variables, write public methods

```
public boolean isFull() {
    return capacity == passengers.size();
}
...
```

Step #2: Using those instance variables, write public methods

```
public String unboardPassenger() {
     return passengers.remove(0);
}
```

Step #3: Finish the constructor

```
// Private instance variables
private ArrayList<String> passengers;
private int capacity;

// Constructor
public Airplane(int numSeats) {
      capacity = numSeats;
      passengers = new ArrayList<String>();
}
```

Servers and Clients

The internet in 3 lines

The internet is a bunch of computers just yelling at each other

The internet in 3 lines

The internet is a bunch of computers just yelling at each other

The computers that yell first are clients, and the computers that yell back are servers

The internet in 3 lines

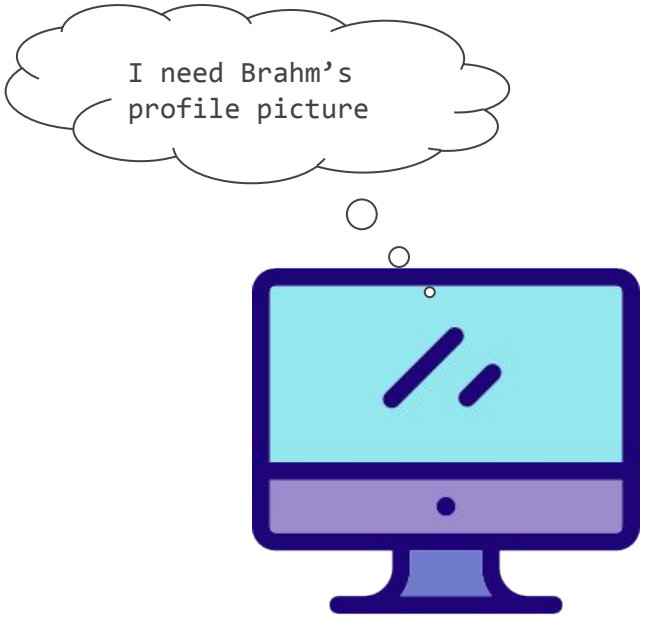
The internet is a bunch of computers just yelling at each other

The computers that yell first are clients, and the computers that yell back are servers

Every yell is made entirely of specially-formatted `Strings`

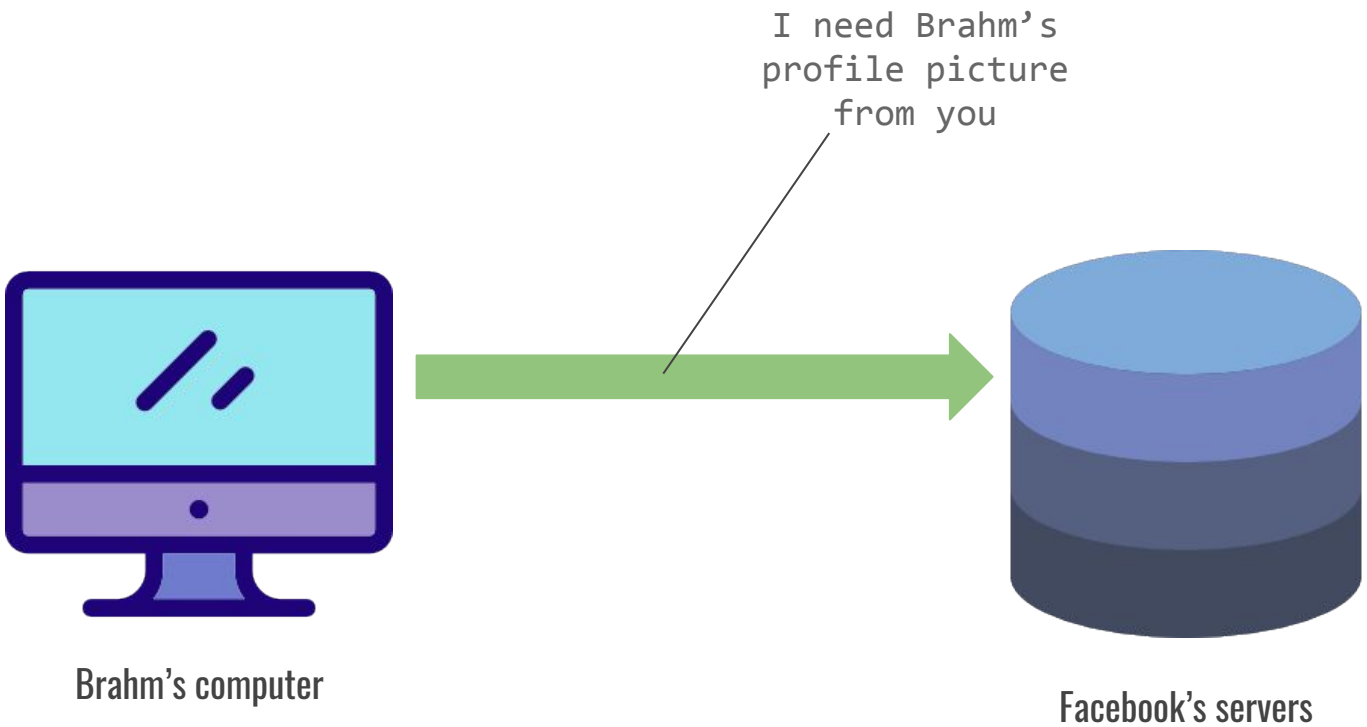
Brahm’s computer

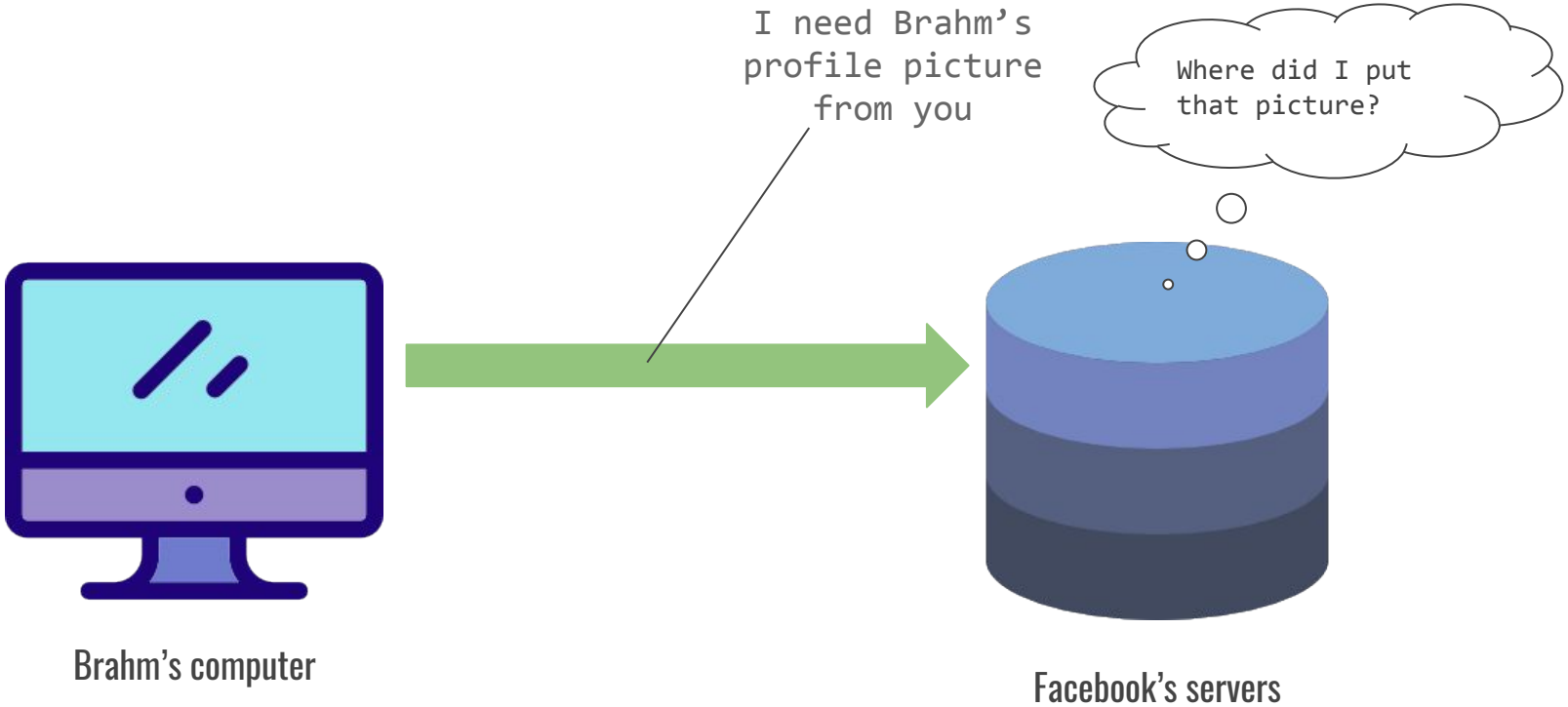
Facebook’s servers

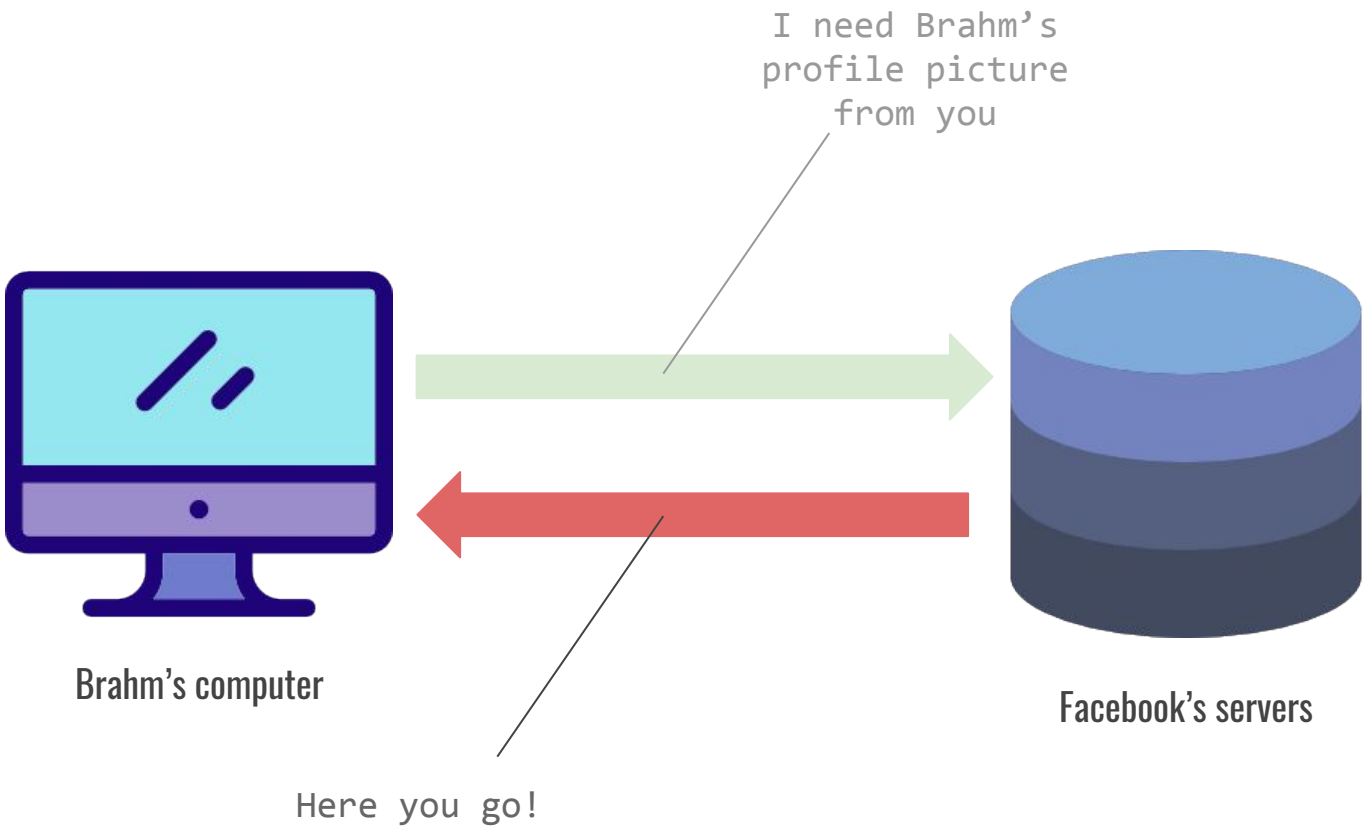


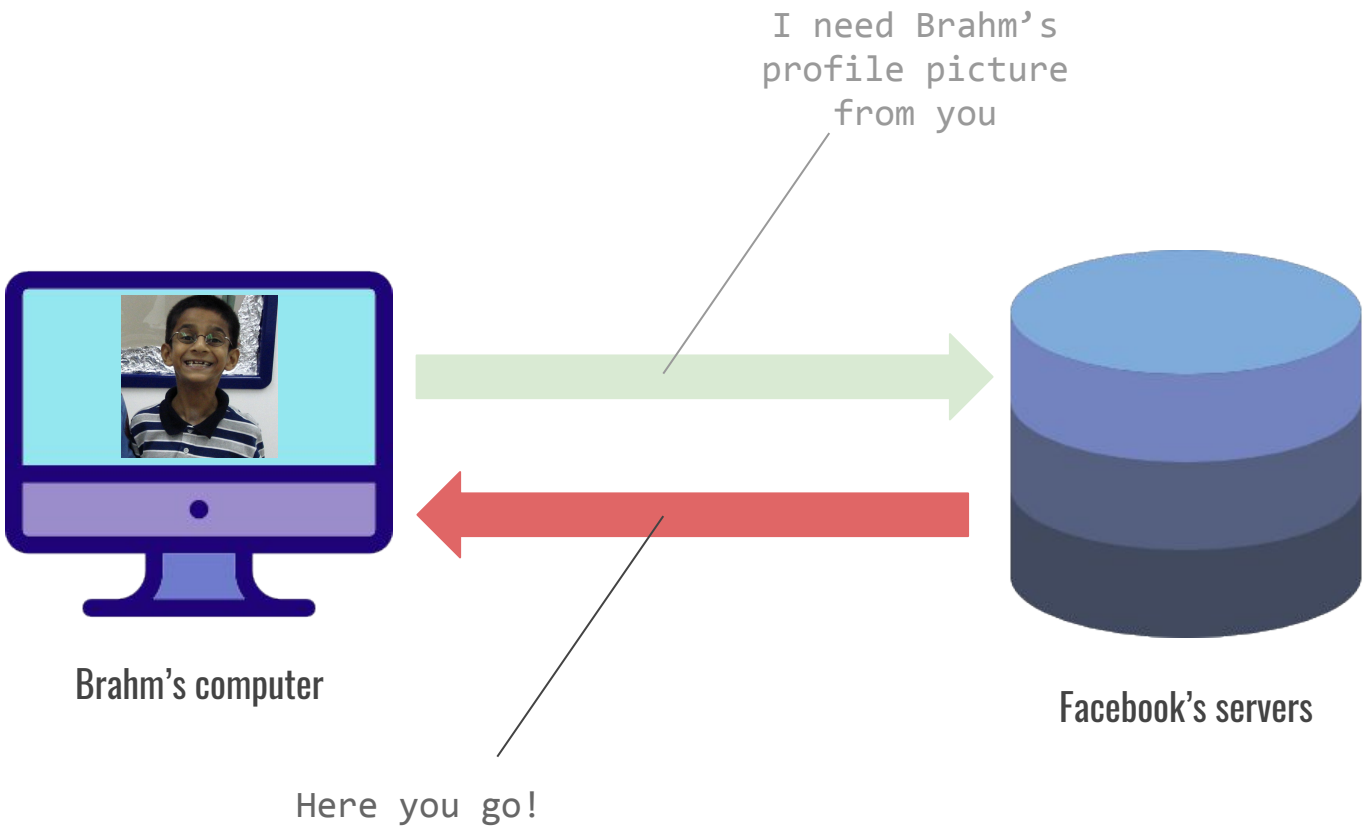
Brahm’s computer

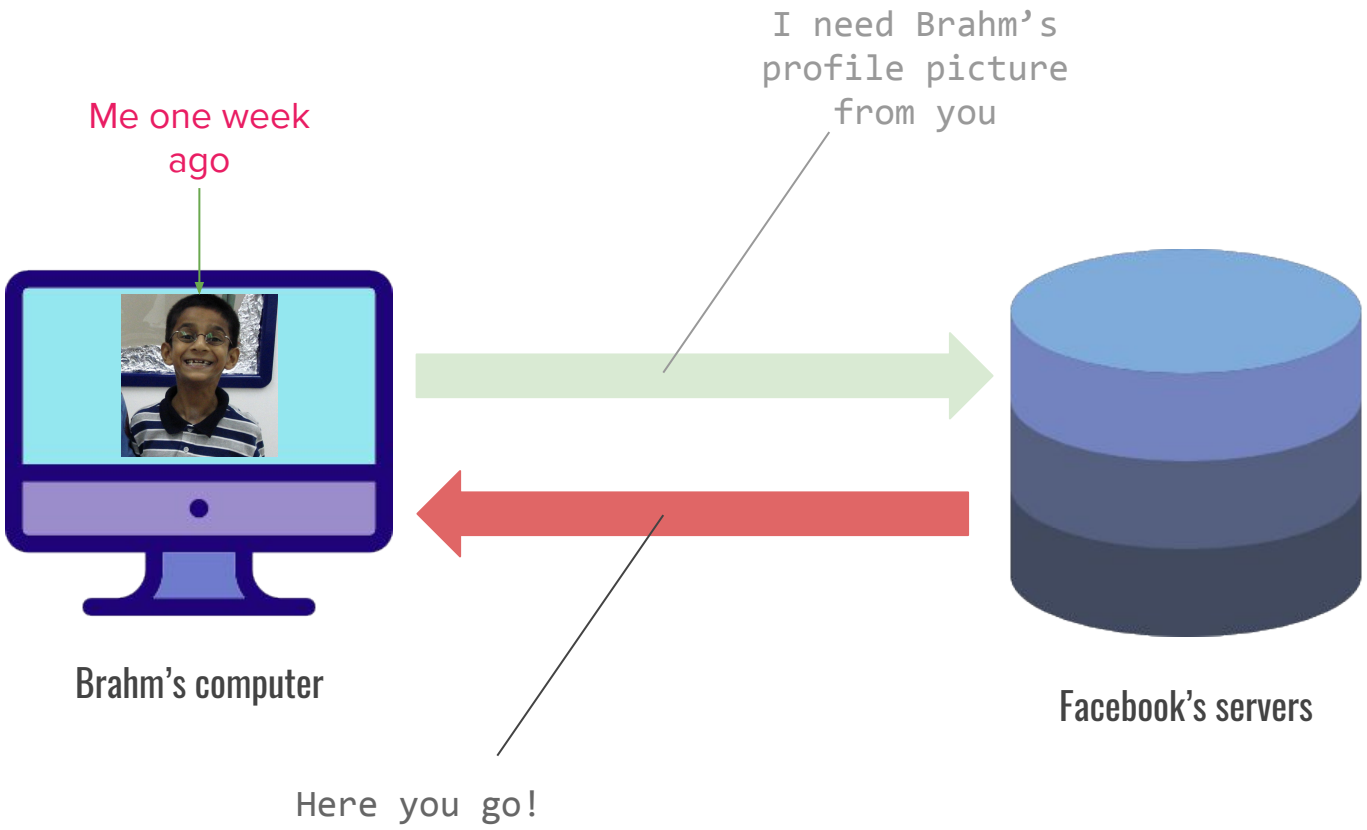
Facebook’s servers

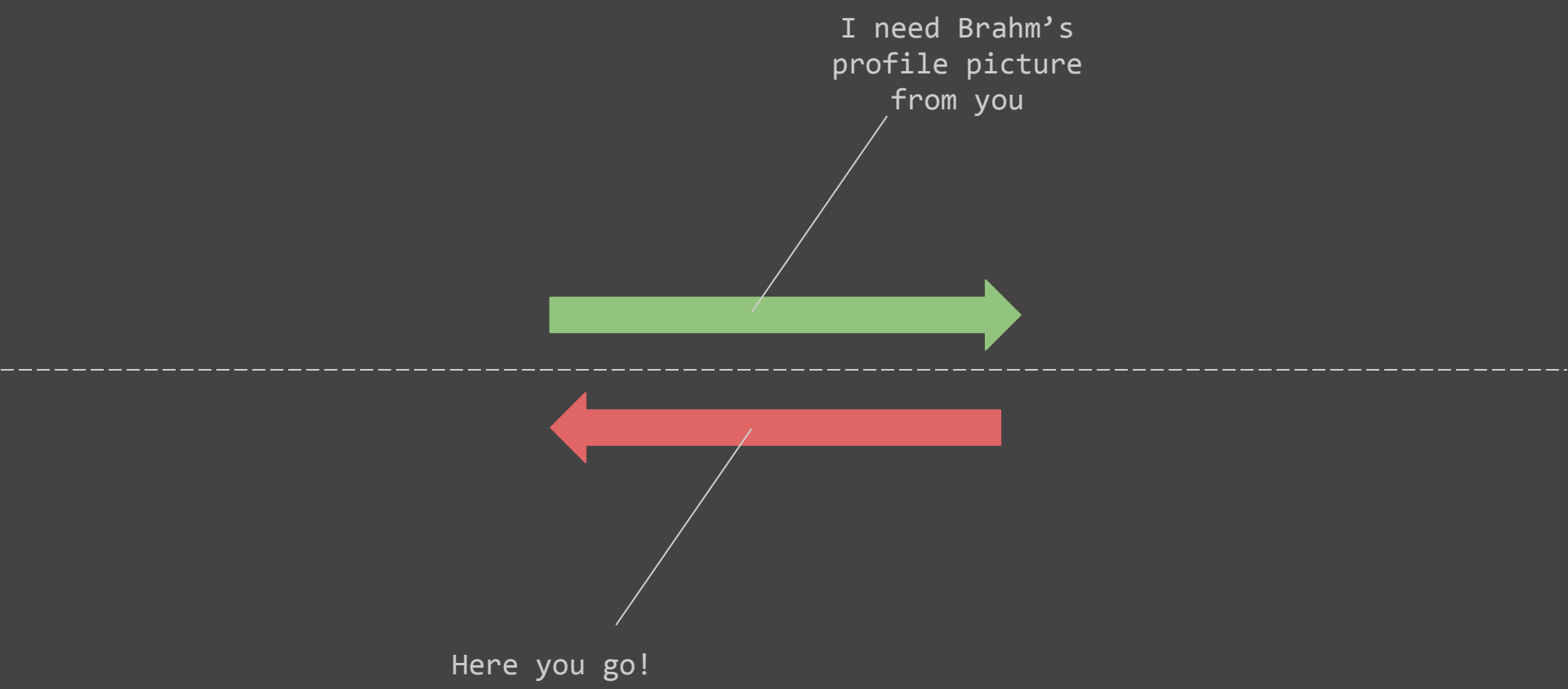

I need Brahm’s profile picture from you
Brahm’s computer
Facebook’s servers


I need Brahm's profile picture from you
Where did I put that picture?
Brahm's computer
Facebook's servers


I need Brahm's profile picture from you
Brahm's computer
Facebook's servers
Here you go!

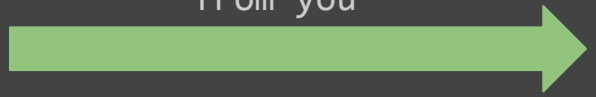

I need Brahm's profile picture from you
Brahm's computer
Facebook's servers
Here you go!


I need Brahm's profile picture from you
Me one week ago
Brahm's computer
Facebook's servers
Here you go!


I need Brahm’s profile picture from you
Here you go!

Request

“I need Brahm’s profile picture from you”



Response

“Here you go!”

Request

made by the client

```
public class Request {

    private String command;
    private HashMap <String, String> params;

    public Request(String command) { … } // constructor

    public void addParam(String name, String val) { … }

    public String getCommand() { … }

    public String getParam(String name) { … }

}
```

Response

by the server

```
/* It’s a string, but the contents of that String are up to
you. */
```

Request

made by the client

```
private static String HOST = “http://localhost:8080”;

private void makeRequest(String username) {
    try {
        Request r = new Request(“getStatus”);
        r.addParam(“username”, username);
        return SimpleClient.makeRequest(HOST, r);
    } catch (IOException e) {
        return null;
    }
}

public void run() {
    String status = makeRequest(“brahmcapoor”);
}
```

Response

by the server

```
public void requestMade(Request req) {
    String cmd = req.getCommand();
    if (cmd.equals(“getStatus”)) {
        String username = req.getParam(“username”);
        String status = “chillin’ like a villain”;
        return status;
    } // and so on...
```

Studying & Exam Strategy

Studying:

Optimize for understanding how everything fits together before how each part works individually

Become familiar with the textbook!

Don’t ask how, ask why a particular solution you see works

In the exam:

- Optimize for what’s easy for you at first
- Make sure a grader understands your thought processes
- Remain calm

After the exam:

You’re done! We’ll take it from here.

Good luck!

You can all do this!