YOUR EARLY ASSIGNMENT HELP | ASSIGNMENT SEVEN

YEAH! Hours: FacePamphlet

Brahm Capoor <brahm@stanford.edu>
Garrick Fernandez <gfaerr@stanford.edu>

Due Friday, Mar. 11 at 5PM PST (No late days!)

Garrick Fernandez
News Feed
Messenger
Watch
Marketplace

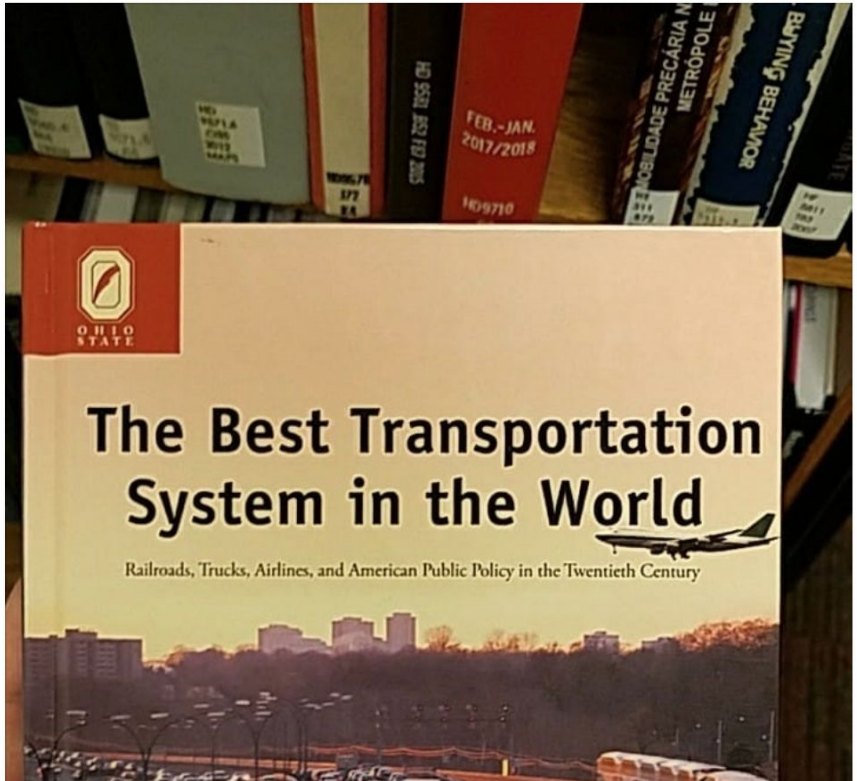
Shortcuts
VJA Alumni Extrava...
Test 11
Chatbot Study 4
Prior 14
Max 9
aestheti cs
FTT/YAC/YA Alumni 8
Free & For Sale 20+
See More...

Explore
Events 1
Groups 3
Pages

New Urbanist Memes for Transit-Oriented Teens

40 mins ·

Pack it up, everyone. Someone literally wrote the book on it. This is peak transit.



Stormy Daniels
Stormy Daniels sues Trump, says agreement invalid because he... - msn.com
Gary Cohn
Gary Cohn, WH chief economic adviser, announces resignation... - foxnews.com
See More

FacePamphletClient

Name Chris Add Delete Lookup

Chris

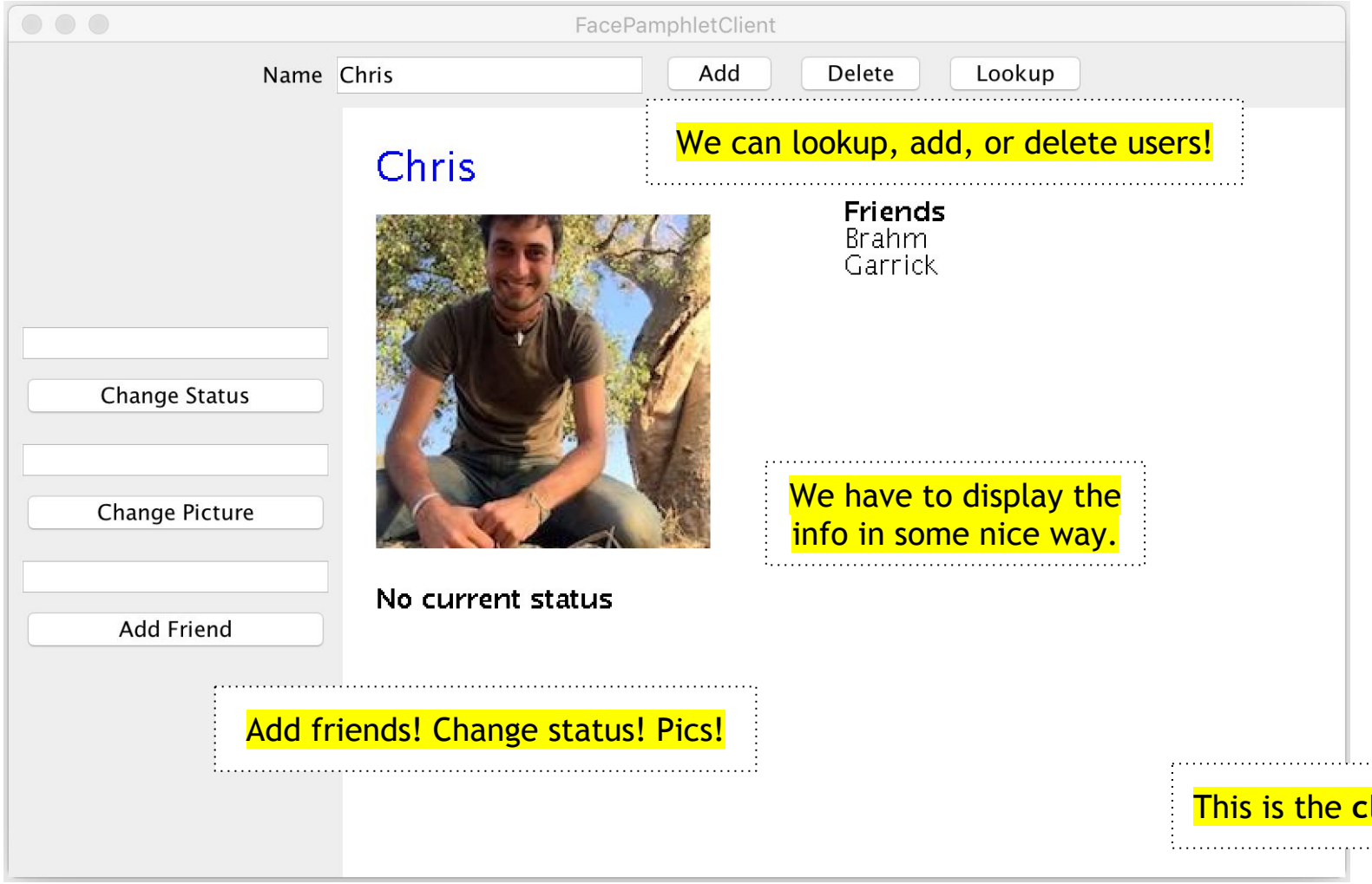
Friends
Brahm
Garrick

Change Status

Change Picture

Add Friend

No current status


FacePamphletClient
Name
Chris
Add
Delete
Lookup
We can lookup, add, or delete users!
Chris
Friends
Brahm
Garrick
Change Status
Change Picture
Add Friend
We have to display the info in some nice way.
No current status
Add friends! Change status! Pics!
This is the client.

Why FacePamphlet?

A fun sneak peek at what Java can do in the real world (the internet)!

Nailing down the concepts introduced in the latter half of the course: **data structures**, **classes**, and interactions between classes (client-server)

Further understanding large-scale programs as systems of Java classes (e.g. NameSurfer and FacePamphlet)

As always...Common Skills

Quickly build **intuition** for what a problem is asking.

Learn how to **draft and design** good code.

Pull bits and snippets from our **coding toolbox**.

Anticipate **edge cases** and **test** for errors.

But first, a lecture review

Classes

Most of FacePamphlet is very similar to NameSurfer

– – –

- Check out assignment 6 YEAH slides!

- Let's put everything you'll need to know on a single slide.

```
public class Comic {




}
```

```
public class Comic {




}
```

```
String title = “Detective Comics 27”;
String author = “Bill Finger”;

Comic comic = new Comic(title, author);

for (int i = 0; i < NUM_PAGES; i++) {
     GImage newPage = drawPage();
     comic.addPage(newPage);
}
```

```
public class Comic {




}
```

```
String title = “Detective Comics 27”;
String author = “Bill Finger”;

Comic comic = new Comic(title, author);

for (int i = 0; i < NUM_PAGES; i++) {
     GImage newPage = drawPage();
     comic.addPage(newPage);
}
```

```
Comic comic = getComicFromShelf();

int numPages = comic.getNumPages();

for (int i = 0; i < numPages; i++) {
     GImage page = comic.getPage(i);
     displayPage(page);
     waitForClick();
}
```

```
public class Comic {

     private String title;
     private String author;
     private ArrayList <GImage> pages;




}
```

```
String title = “Detective Comics 27”;
String author = “Bill Finger”;

Comic comic = new Comic(title, author);

for (int i = 0; i < NUM_PAGES; i++) {
     GImage newPage = drawPage();
     comic.addPage(newPage);
}
```

```
Comic comic = getComicFromShelf();

int numPages = comic.getNumPages();

for (int i = 0; i < numPages; i++) {
     GImage page = comic.getPage(i);
     displayPage(page);
     waitForClick();
}
```

```
public class Comic {

     private String title;
     private String author;
     private ArrayList <GImage> pages;

     public ComicBook(String title, String author) {
          this.title = title;
          this.author = author;
          pages = new ArrayList<GImage>();
     }




}
```

```
String title = “Detective Comics 27”;
String author = “Bill Finger”;

Comic comic = new Comic(title, author);

for (int i = 0; i < NUM_PAGES; i++) {
     GImage newPage = drawPage();
     comic.addPage(newPage);
}
```

```
Comic comic = getComicFromShelf();

int numPages = comic.getNumPages();

for (int i = 0; i < numPages; i++) {
     GImage page = comic.getPage(i);
     displayPage(page);
     waitForClick();
}
```

```
public class Comic {

     private String title;
     private String author;
     private ArrayList <GImage> pages;

     public ComicBook(String title, String author) {
          this.title = title;
          this.author = author;
          pages = new ArrayList<GImage>();
     }

     public void addPage(GImage page) {
          this.pages.add(page);
     }




}
```

```
String title = “Detective Comics 27”;
String author = “Bill Finger”;

Comic comic = new Comic(title, author);

for (int i = 0; i < NUM_PAGES; i++) {
     GImage newPage = drawPage();
     comic.addPage(newPage);
}
```

```
Comic comic = getComicFromShelf();

int numPages = comic.getNumPages();

for (int i = 0; i < numPages; i++) {
     GImage page = comic.getPage(i);
     displayPage(page);
     waitForClick();
}
```

```
public class Comic {

     private String title;
     private String author;
     private ArrayList <GImage> pages;

     public ComicBook(String title, String author) {
          this.title = title;
          this.author = author;
          pages = new ArrayList<GImage>();
     }

     public void addPage(GImage page) {
          this.pages.add(page);
     }

     public GImage getPage(int pageNum) {
          return this.pages.get(pageNum);
     }



}
```

```
String title = “Detective Comics 27”;
String author = “Bill Finger”;

Comic comic = new Comic(title, author);

for (int i = 0; i < NUM_PAGES; i++) {
     GImage newPage = drawPage();
     comic.addPage(newPage);
}
```

```
Comic comic = getComicFromShelf();

int numPages = comic.getNumPages();

for (int i = 0; i < numPages; i++) {
     GImage page = comic.getPage(i);
     displayPage(page);
     waitForClick();
}
```

```
public class Comic {

     private String title;
     private String author;
     private ArrayList <GImage> pages;

     public ComicBook(String title, String author) {
          this.title = title;
          this.author = author;
          pages = new ArrayList<GImage>();
     }

     public void addPage(GImage page) {
          this.pages.add(page);
     }

     public GImage getPage(int pageNum) {
          return this.pages.get(pageNum);
     }

     public String getTitle() {...}
     public String getAuthor() {...}
     public int getNumPages() {...}
}
```

```
String title = "Detective Comics 27";
String author = "Bill Finger";

Comic comic = new Comic(title, author);

for (int i = 0; i < NUM_PAGES; i++) {
     GImage newPage = drawPage();
     comic.addPage(newPage);
}
```

```
Comic comic = getComicFromShelf();

int numPages = comic.getNumPages();

for (int i = 0; i < numPages; i++) {
     GImage page = comic.getPage(i);
     displayPage(page);
     waitForClick();
}
```

Networking

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

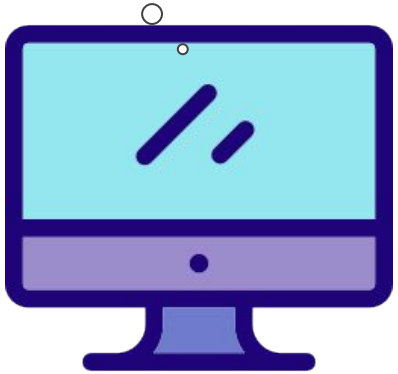
The computers that yell first are clients, and the computers that yell back are servers

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

Brahm's computer

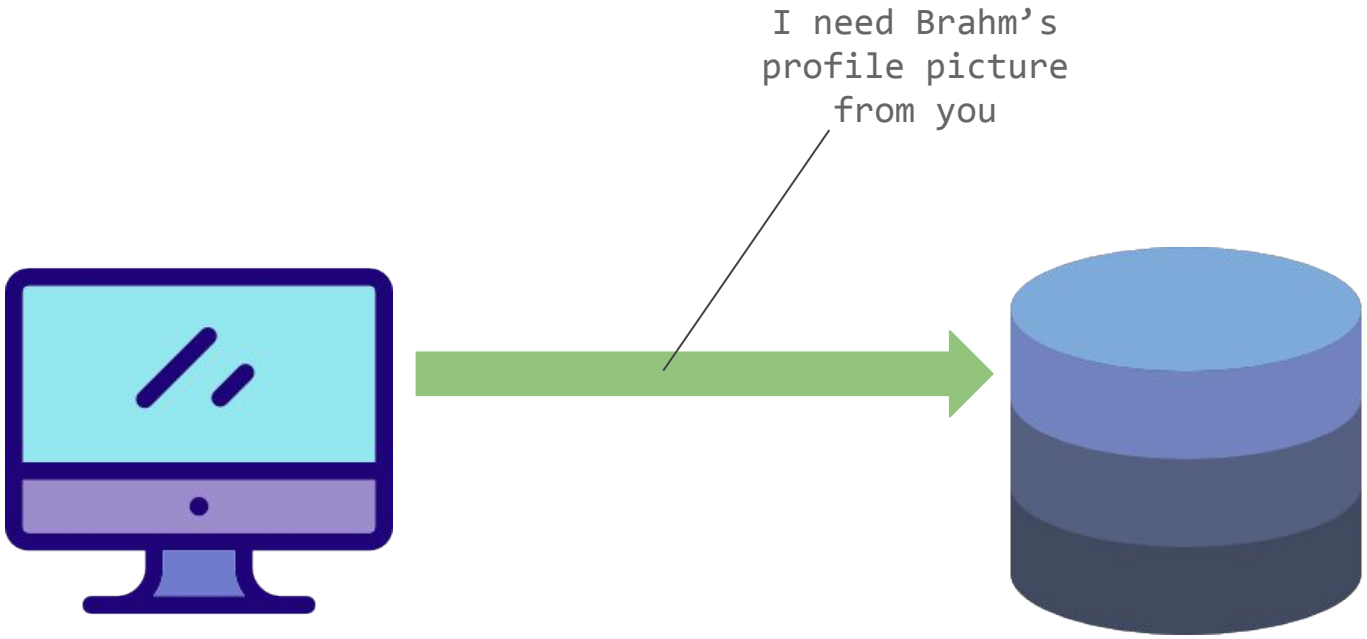
Facebook's servers

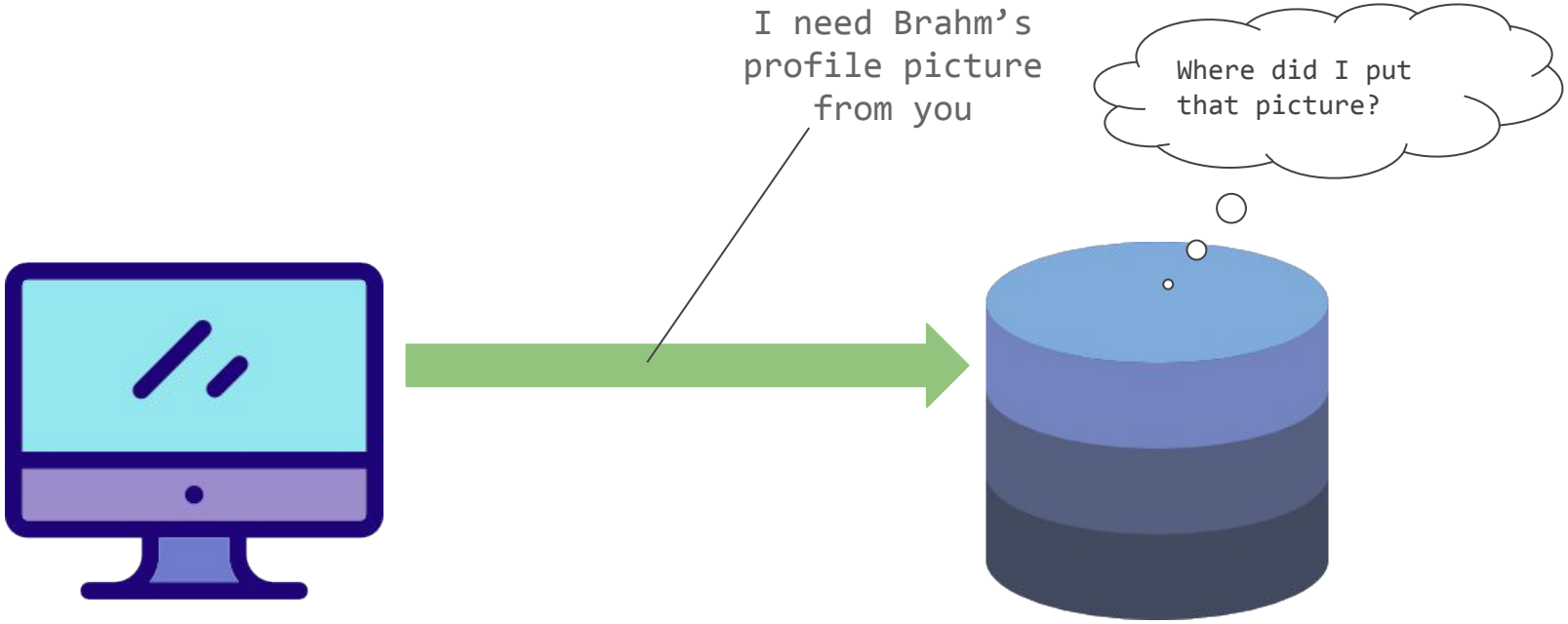
I need Brahm's profile picture

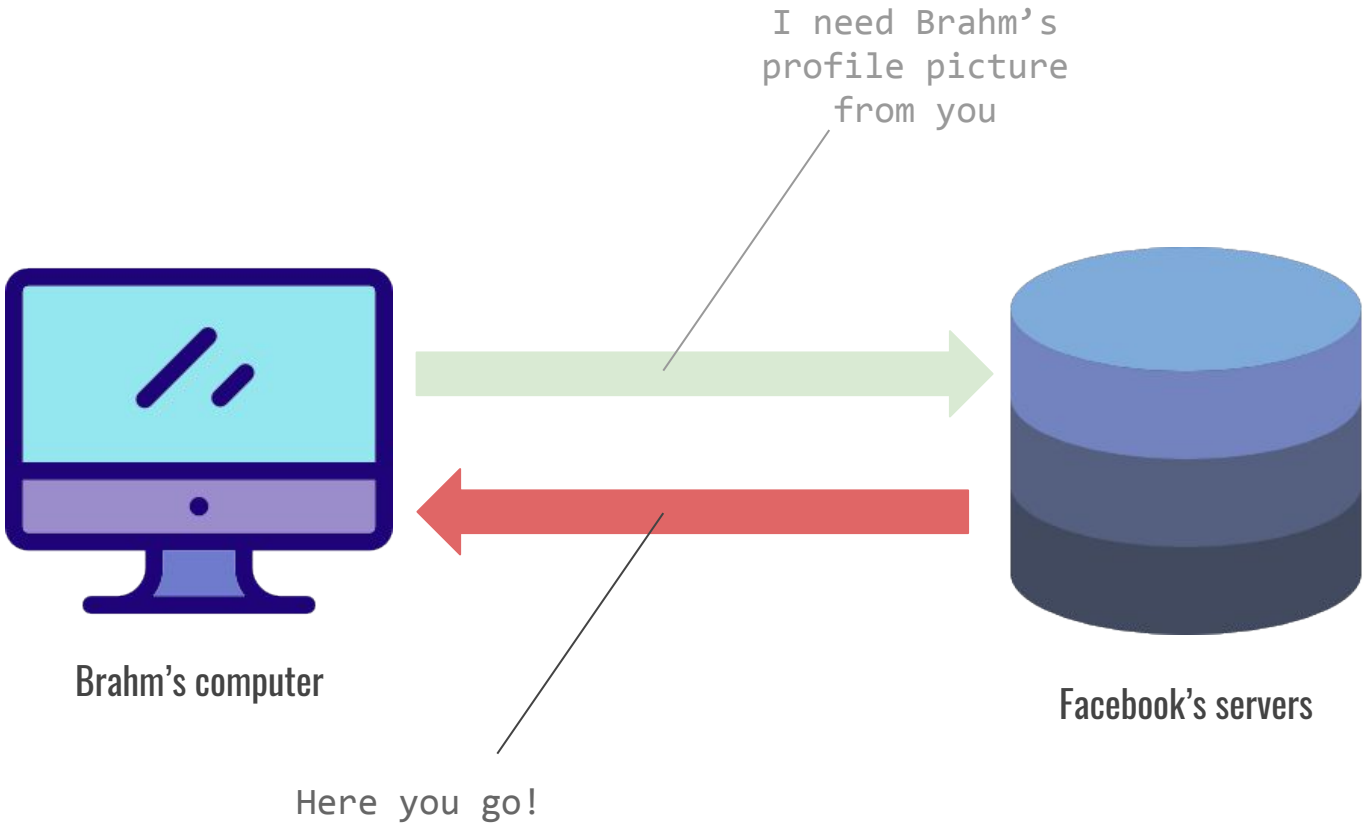


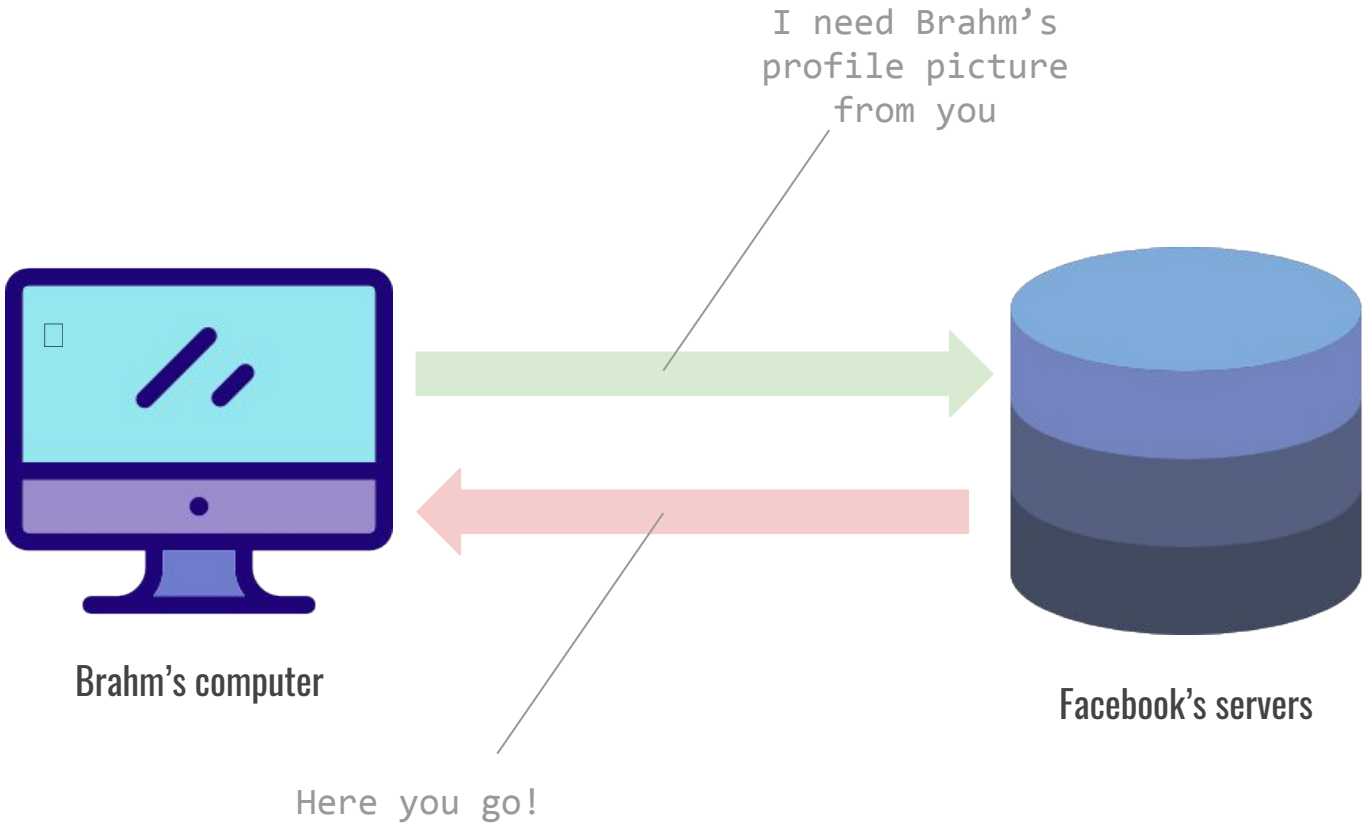
Brahm's computer

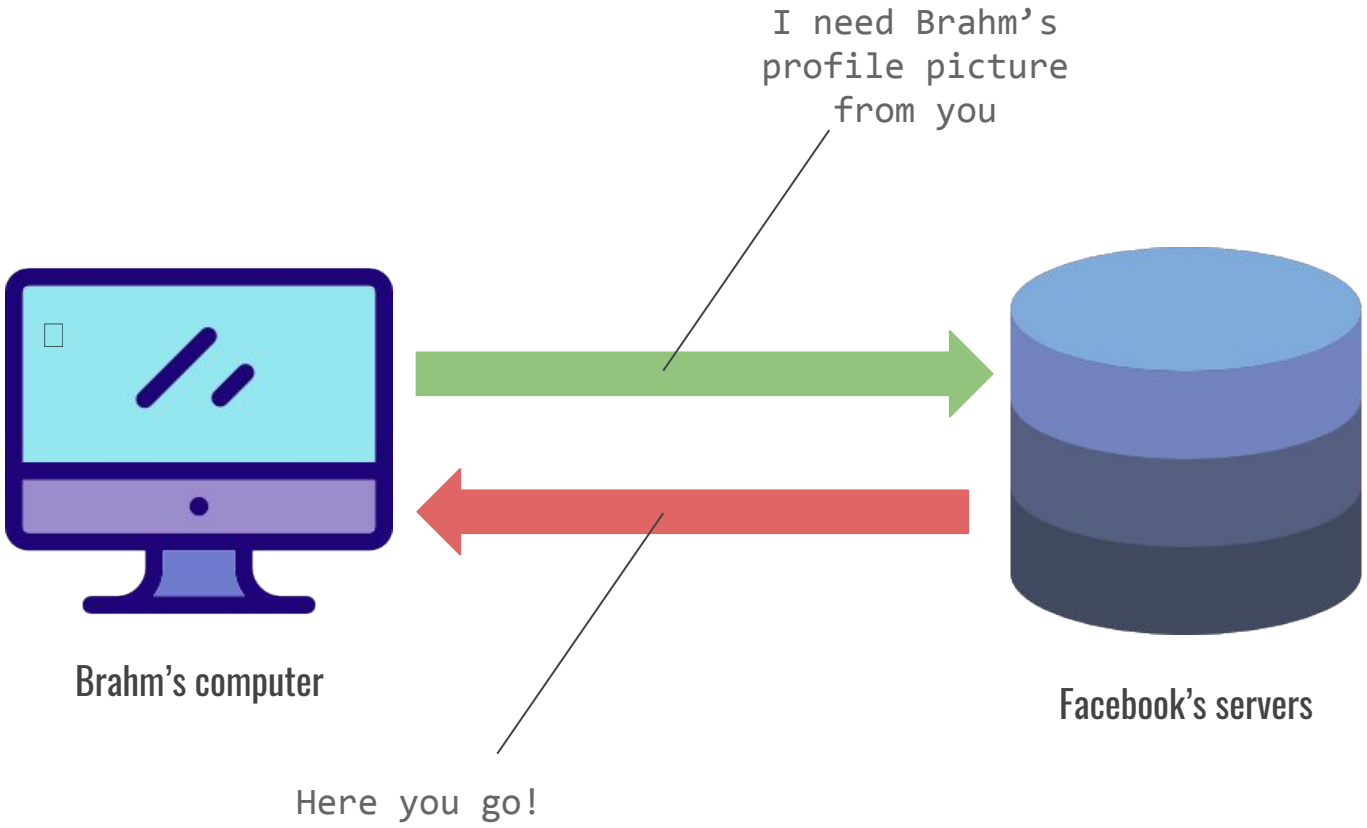
Facebook's servers

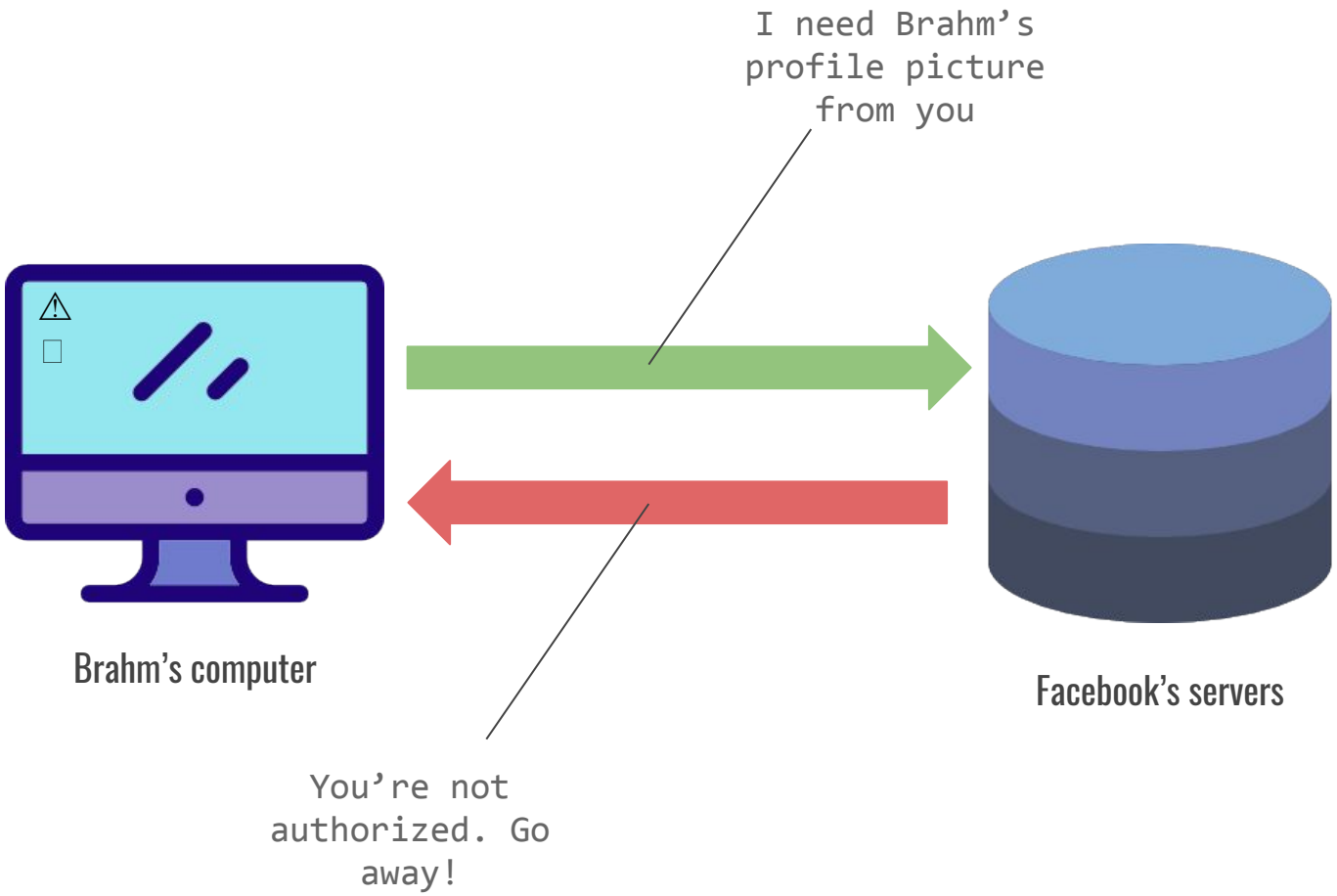

I need Brahm’s profile picture from you
Brahm’s computer
Facebook’s servers

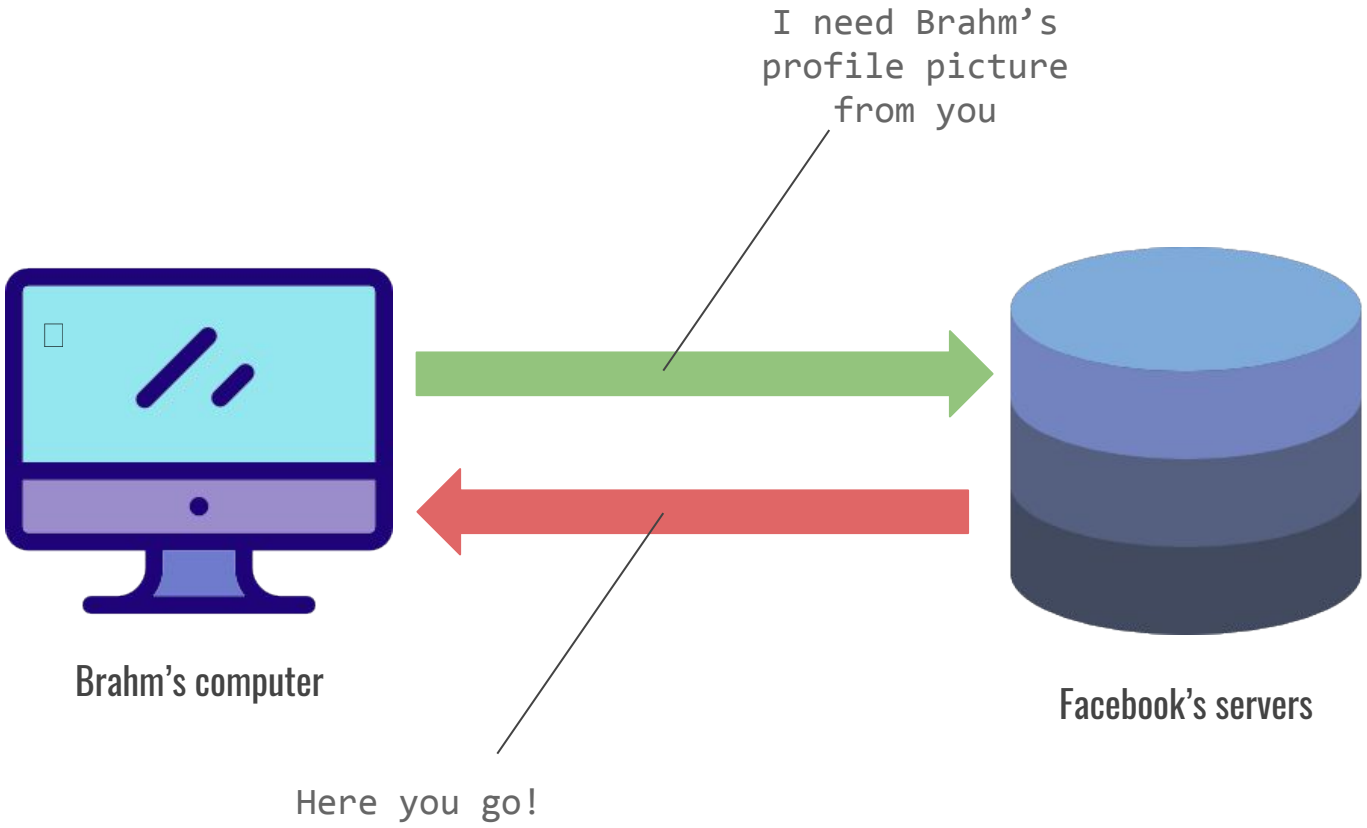

I need Brahm's profile picture from you
Where did I put that picture?
Brahm's computer
Facebook's servers

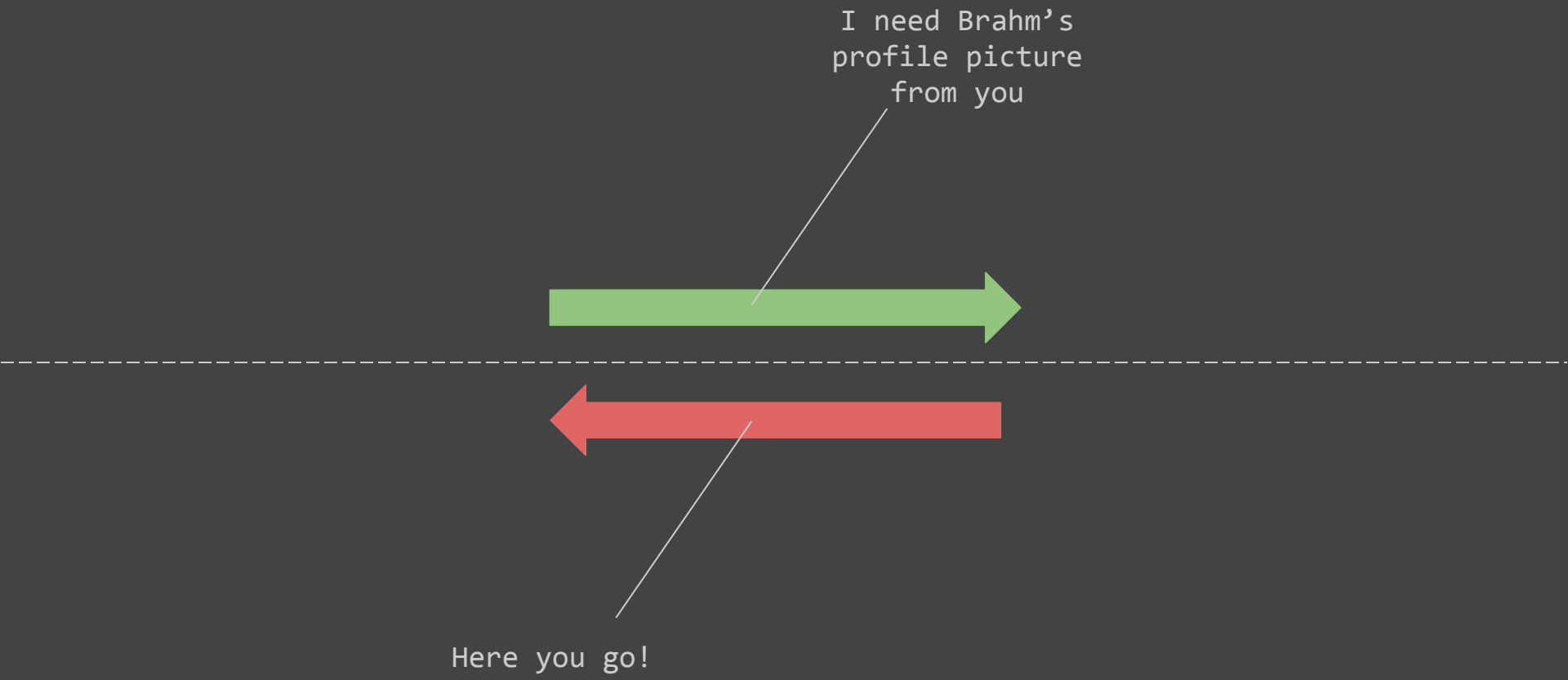

I need Brahm's profile picture from you
Brahm's computer
Facebook's servers
Here you go!


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
Brahm's computer
Facebook's servers
Here you go!

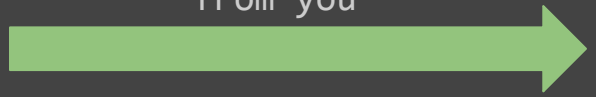

I need Brahm’s profile picture from you
Brahm’s computer
Facebook’s servers
You’re not authorized. Go away!


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
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. Choose something sensible/check the handout! */
```

Request

made by the client

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

private String 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

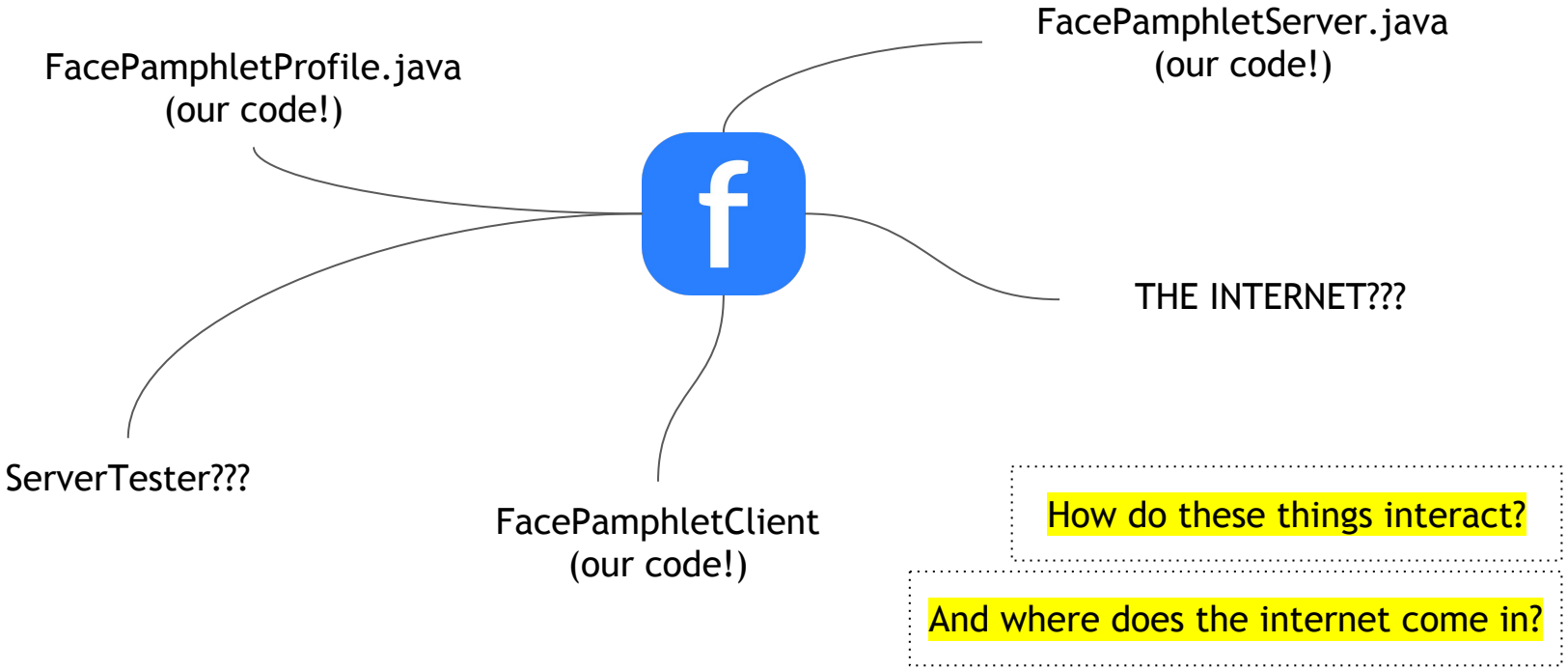
```
public String requestMade(Request req) {
     String cmd = req.getCommand();
     if (cmd.equals(“getStatus”)) {
          String username = req.getParam(“username”);
          String status = /* obfuscated for you to do */;
          return status;
     } // and so on...
```

```
Assignment[] CS106A = new Assignment[7];

Assignment facePamphlet = CS106A[CS106a.length - 1];

// You're so close! You've all crushed it so far!
```

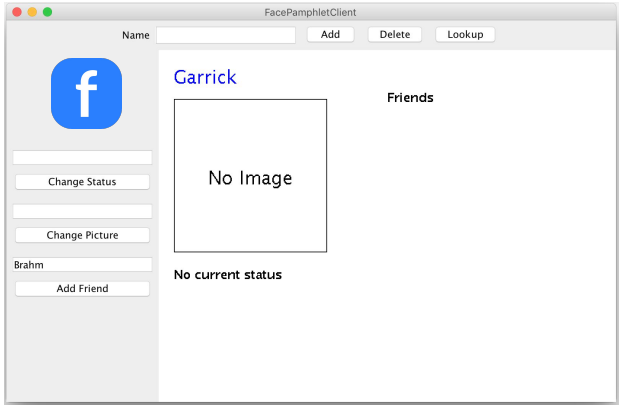
A Primer: FacePamphlet 



A Primer: FacePamphlet 

SERVER >

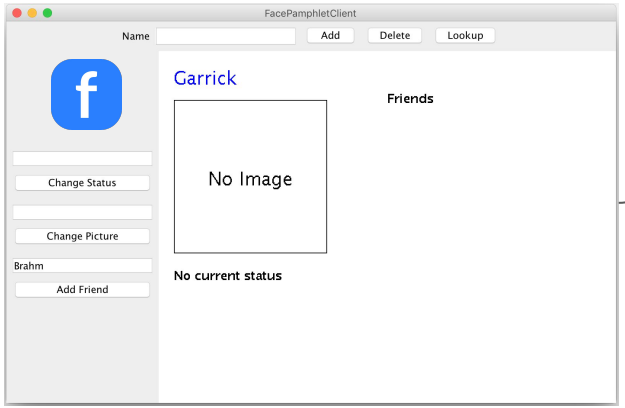
FacePamphletServer.java
(our code!)



FacePamphletClient
(our code!)

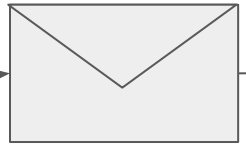
< CLIENT

A Primer: FacePamphlet 



FacePamphletClient
(our code!)

SERVER >



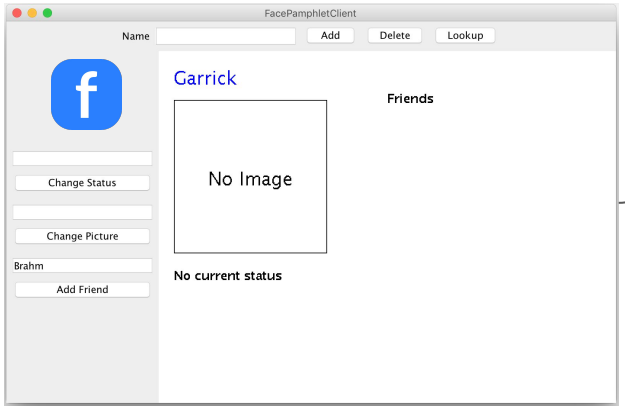
STORAGE?

FacePamphletServer.java
code!)

There's this idea of requests and responses over the internet.

< CLIENT

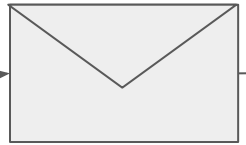
A Primer: FacePamphlet



FacePamphletClient
(our code!)

SERVER >

< CLIENT



STORAGE?

FacePamphletServer.java
code!)

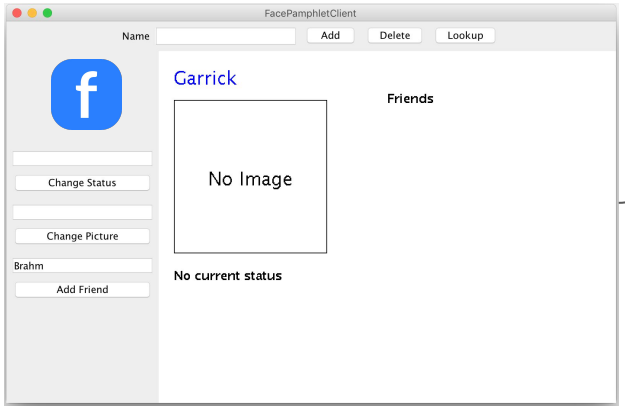
There's this idea of requests and responses over the internet.

The server stores all of the user profiles in some useful way!
(Think NameSurfer)

name:	Garrick
other things:	blah blah blah...

FacePamphletProfile.java
(our code!)

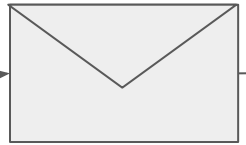
A Primer: FacePamphlet



FacePamphletClient
(our code!)

< CLIENT

SERVER >



FacePamphletServer.java
(our code!)

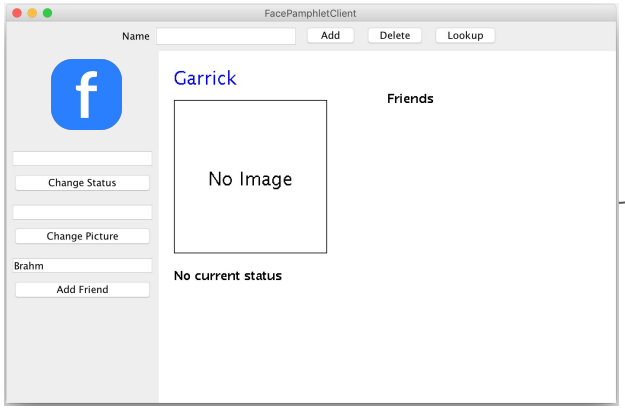
STORAGE?

Let's see how a request/response works.

name:	Garrick
other things:	blah blah blah...

FacePamphletProfile.java
(our code!)

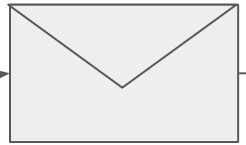
A Primer: FacePamphlet



FacePamphletClient
(our code!)

< CLIENT

SERVER >



FacePamphletServer.java
(our code!)

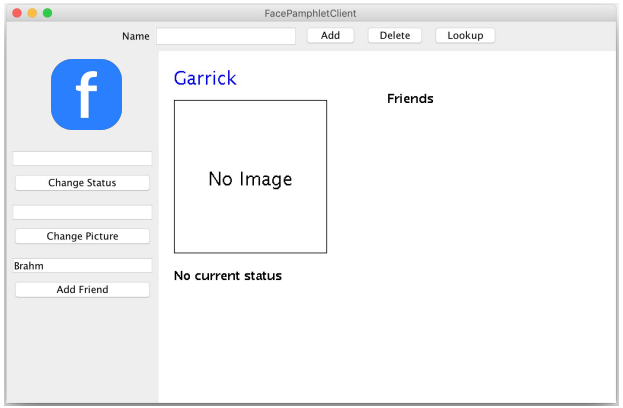
STORAGE?

First, a request is sent from the client to the server over the internet.

name:	Garrick
other things:	blah blah blah...

FacePamphletProfile.java
(our code!)

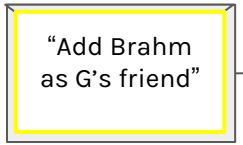
A Primer: FacePamphlet



FacePamphletClient
(our code!)

< CLIENT

SERVER >



STORAGE?

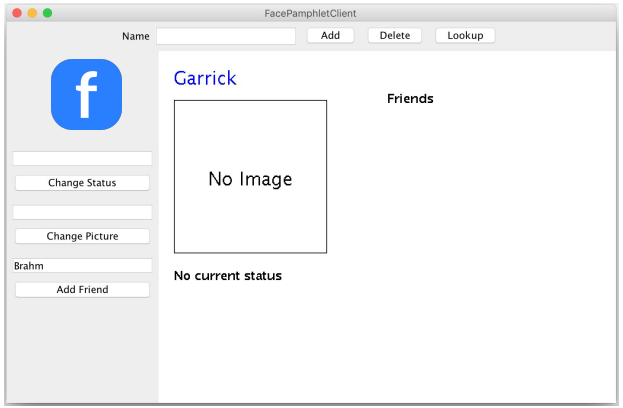
FacePamphletServer.java
(our code!)

The server receives the request and decides what to do with it.

name:	Garrick
other things:	blah blah blah...

FacePamphletProfile.java
(our code!)

A Primer: FacePamphlet

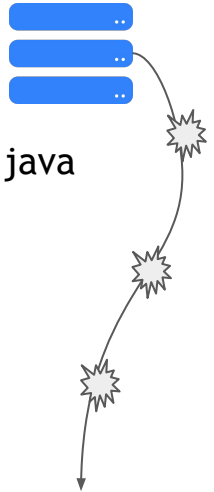


FacePamphletClient
(our code!)

< CLIENT

SERVER >

STORAGE?



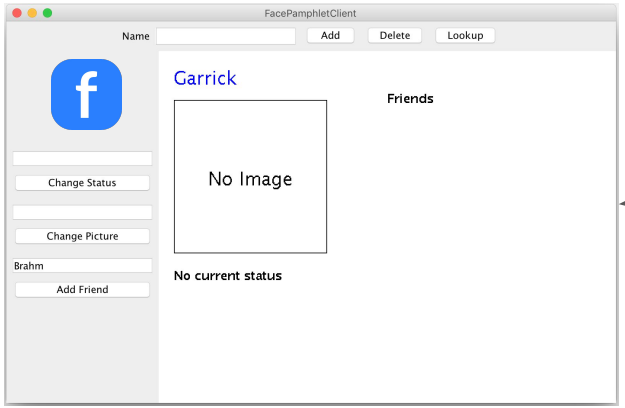
FacePamphletServer.java
(our code!)

The server *could* change internal data to satisfy the request.

name:	Garrick
other things:	Brahm is a friend!

FacePamphletProfile.java
(our code!)

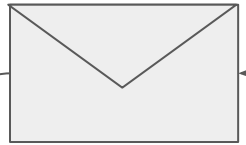
A Primer: FacePamphlet



FacePamphletClient
(our code!)

SERVER >

< CLIENT



FacePamphletServer.java
(our code!)

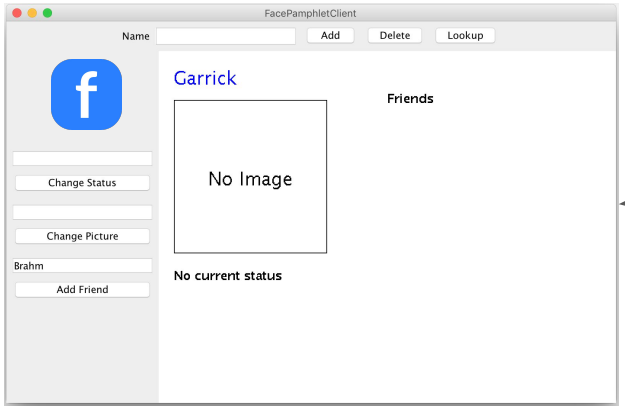
STORAGE?

Then, the server sends a response back to the user...

name:	Garrick
other things:	Brahm is a friend!

FacePamphletProfile.java
(our code!)

A Primer: FacePamphlet



FacePamphletClient
(our code!)

SERVER >

< CLIENT

FacePamphletServer.java
(our code!)

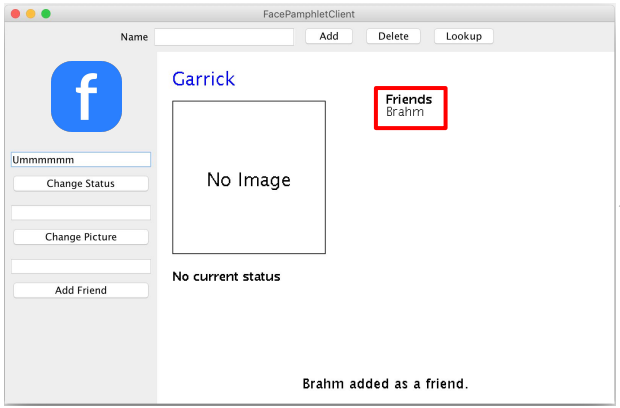
STORAGE?

The client receives the response...

name:	Garrick
other things:	Brahm is a friend!

FacePamphletProfile.java
(our code!)

A Primer: FacePamphlet



FacePamphletClient
(our code!)

SERVER >

STORAGE?

FacePamphletServer.java
(our code!)

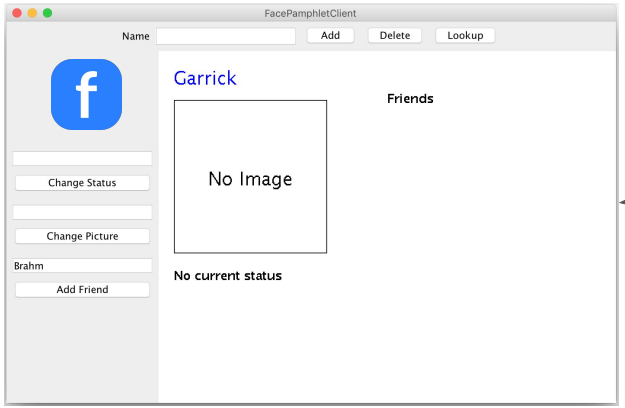
And updates appropriately!

< CLIENT

name:	Garrick
other things:	Brahm is a friend!

FacePamphletProfile.java
(our code!)

A Primer: FacePamphlet



FacePamphletClient
(our code!)

< CLIENT

SERVER >

Request/
Response

FacePamphletServer.java
(our code!)

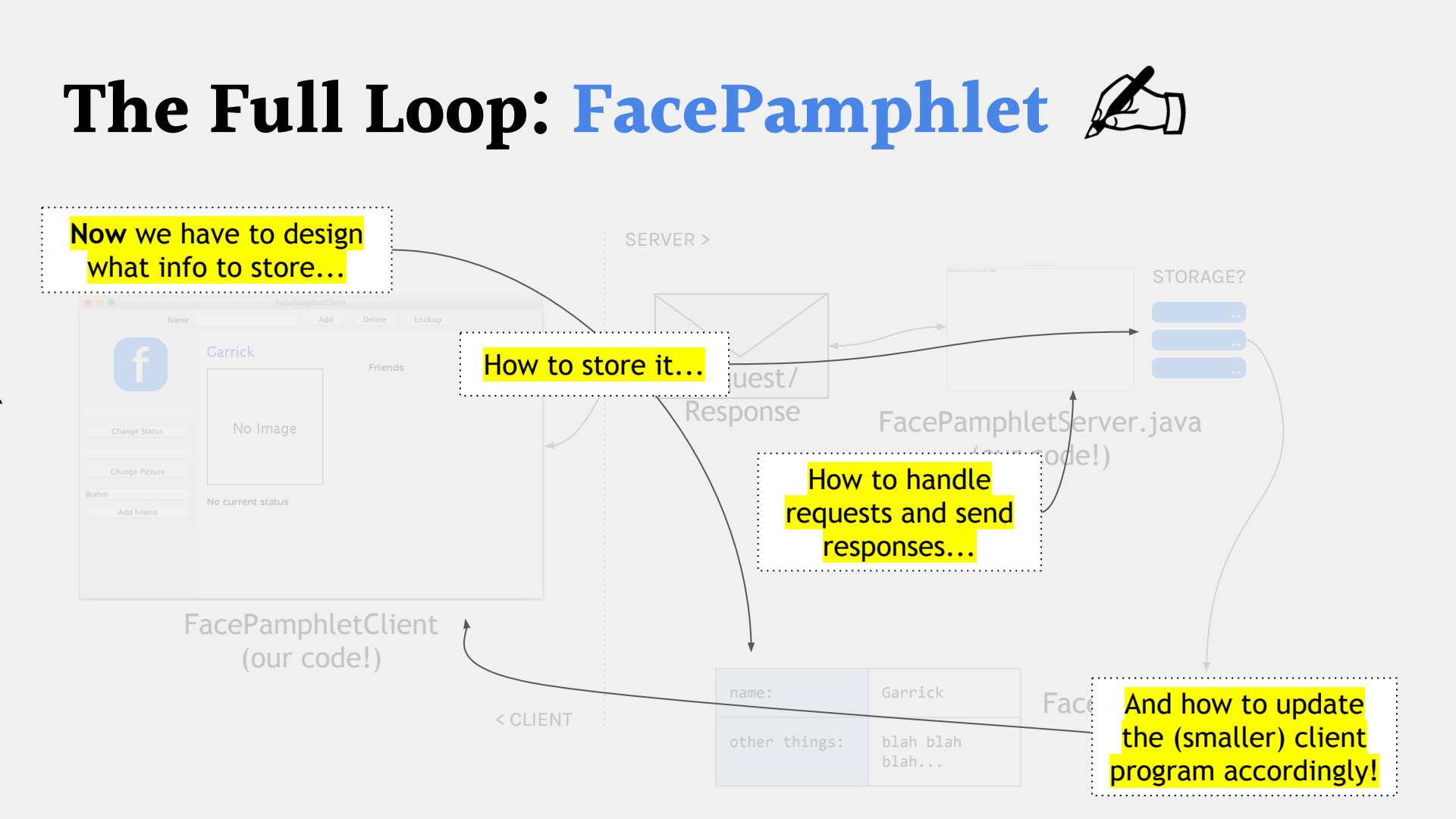
STORAGE?

This is a **request-response pattern** between client and server.

name:	Garrick
other things:	blah blah blah...

FacePamphletProfile.java
(our code!)

The Full Loop: FacePamphlet



Breaking Up the Problem

How do we break up the problem into approachable milestones?

Breaking Up the Problem

First, let's understand how classes would be useful for solving the problem of storing data in FacePamphlet.

Then, let's look at how requests are sent and how to respond to them!

Last, let's wrap back to the client and make a simple one!

Designing FacePamphletProfile

The class `FacePamphletProfile` is going to **abstract** the data we need to store for each user in a useful way.

We need to decide on a few things to make this happen:

- What data to store, and how?
- Choosing methods to interact with objects of our class

We tell you which methods need to be implemented. You need to decide on the structures and write the code!

Tips: FacePamphletProfile

Check out the handout! The four bits of info are:

1. The name of the person with this profile, such as "Chris Piech"
2. The status associated with this profile. This is just a String indicating what the person associated with the profile is currently doing. It should initially be the empty string.
3. The image associated with that profile. This is a GImage. It should initially be null.
4. The list of friends of this profile. The list of friends is simply a list of the names (Strings) that are friends with this profile. This list starts empty. The data structure you use to for this is left up to you.

Plus a high-level diagram!

Breaking Up the Problem

First, let’s understand how classes would be useful for solving the problem of storing data in FacePamphlet.

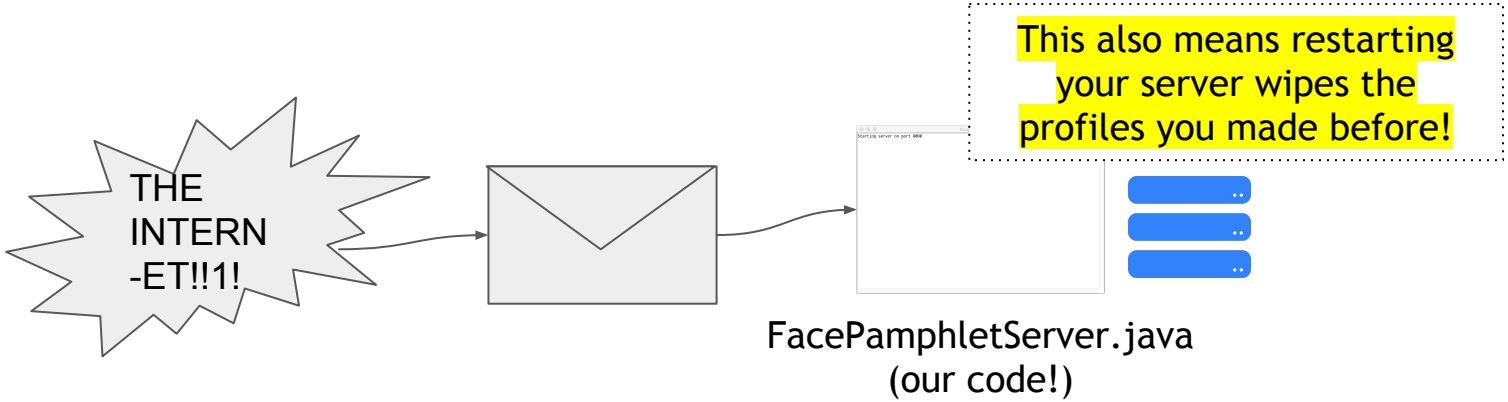
Then, let’s look at how requests are sent and how to respond to them!

Last, let’s wrap back to the client and make a simple one!

Earth to Mars

Requests are going to be sent to our FacePamphlet server over the internet.

For that to work, you have to be running the server. **Your computer** is receiving and responding to requests. *This means you have to run the server when you run the client!*



Step One: Our Program is a Server

Our class, `FacePamphletServer`, is going to **implement the interface** called `SimpleServerListener`. (Just think of this as a template, or **series of promises** we have to follow to make our program behave like a server.)

```
public class FacePamphletServer extends ConsoleProgram
    implements SimpleServerListener {

    // Our server code here

    public String requestMade(Request request) {}
}
```

This method is a promise of the **interface**, or template. Makes sense—all servers have to respond to requests!

Step Two: Housekeeping

We need to set up some things behind the scenes so our server knows how to work. The **port** tells the server where it's getting requests from.

```
public class FacePamphletServer extends ConsoleProgram
    implements SimpleServerListener {

    private static final int PORT = 8000;
    private SimpleServer server = new SimpleServer(this, PORT);

}
```

And we need to make a SimpleServer!

Step Three: Fire up!

The `SimpleServer` we made only needs one more thing to get started. We have to call `start()` from the `run()` method, no questions about it.

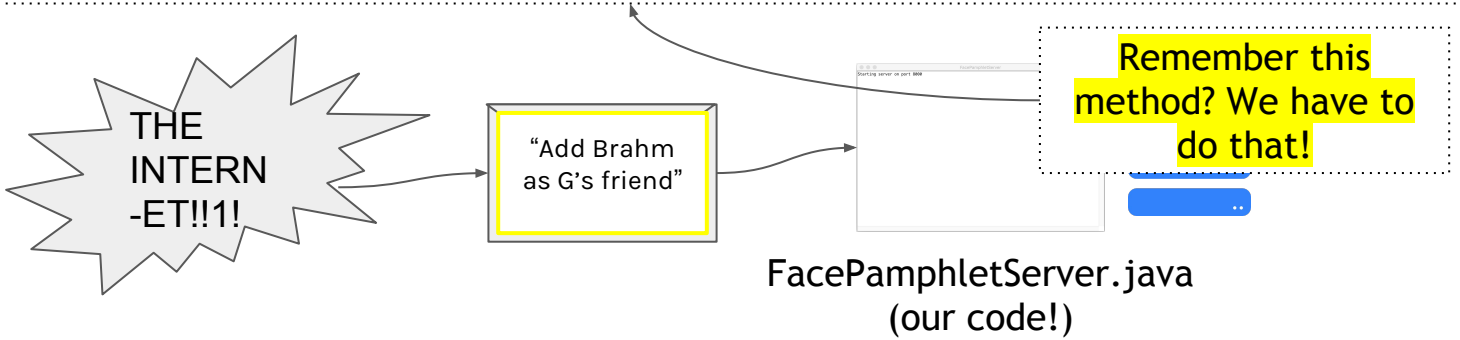
```
public void run() {
    println("Starting server on port " + PORT);
    server = new SimpleServer(this, PORT);
    server.start(); // Start the server
}
```

Great, what does that do?

Now we have the code that's running the server! It can receive requests.

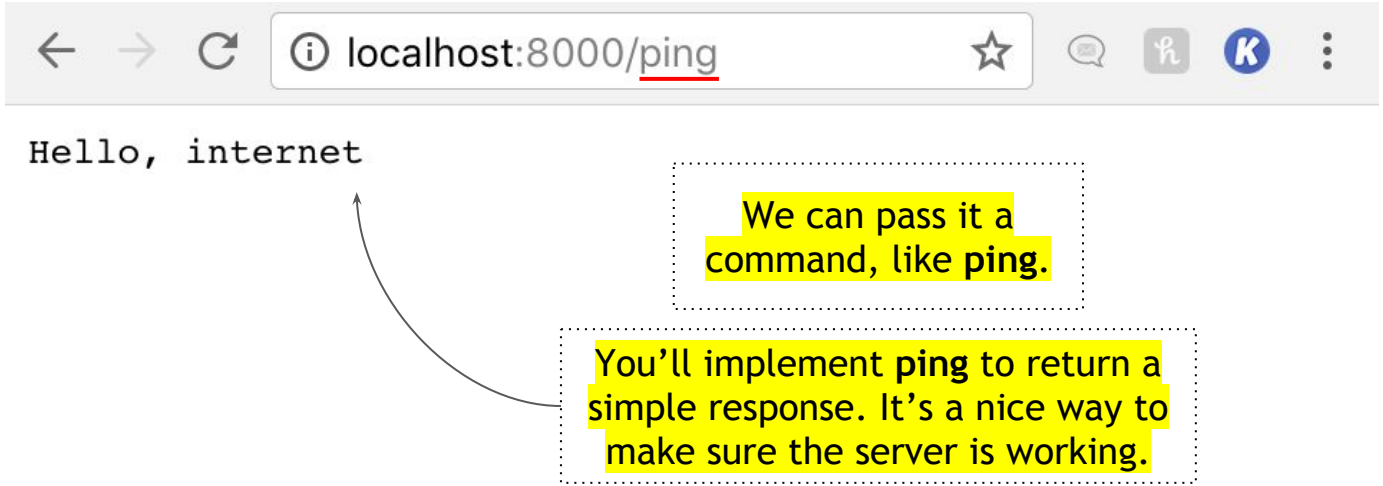
The only thing we have to do now is respond to requests.

```
public String requestMade(Request request) {} // TODO
```



Testing: Seeing the Server

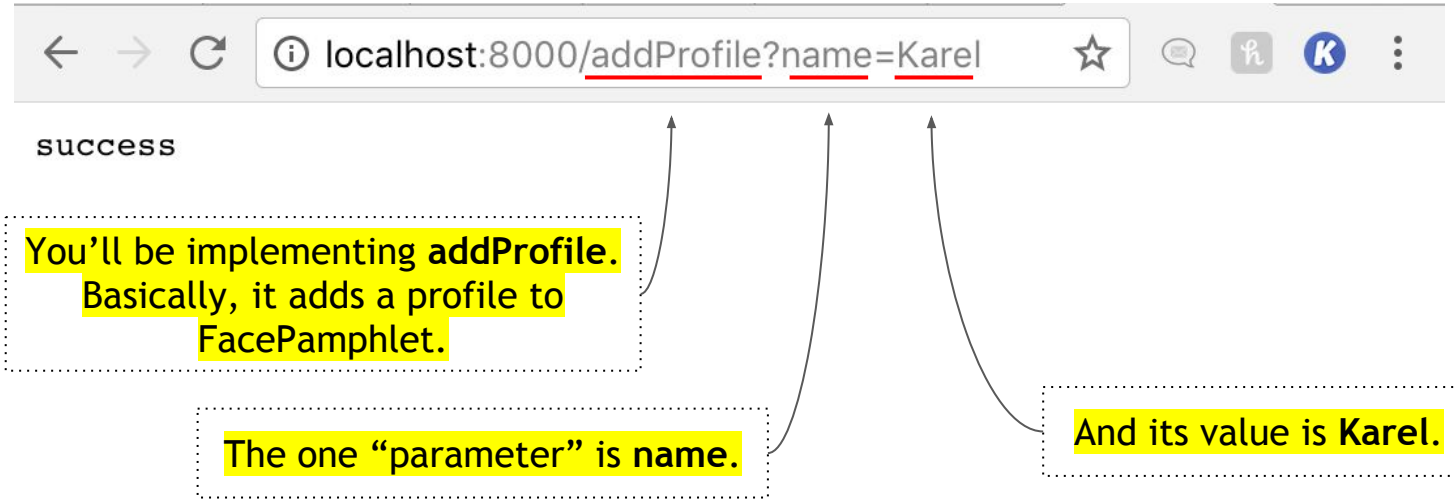
Say your **port** is 8000, you can go to `localhost:8000/` in a web browser to contact your server (remember, it has to be running!)



Testing: Making Commands

Everything following `localhost:8000/` is a **command**, and "**parameters**" of the command use this URL syntax: `?param=value`

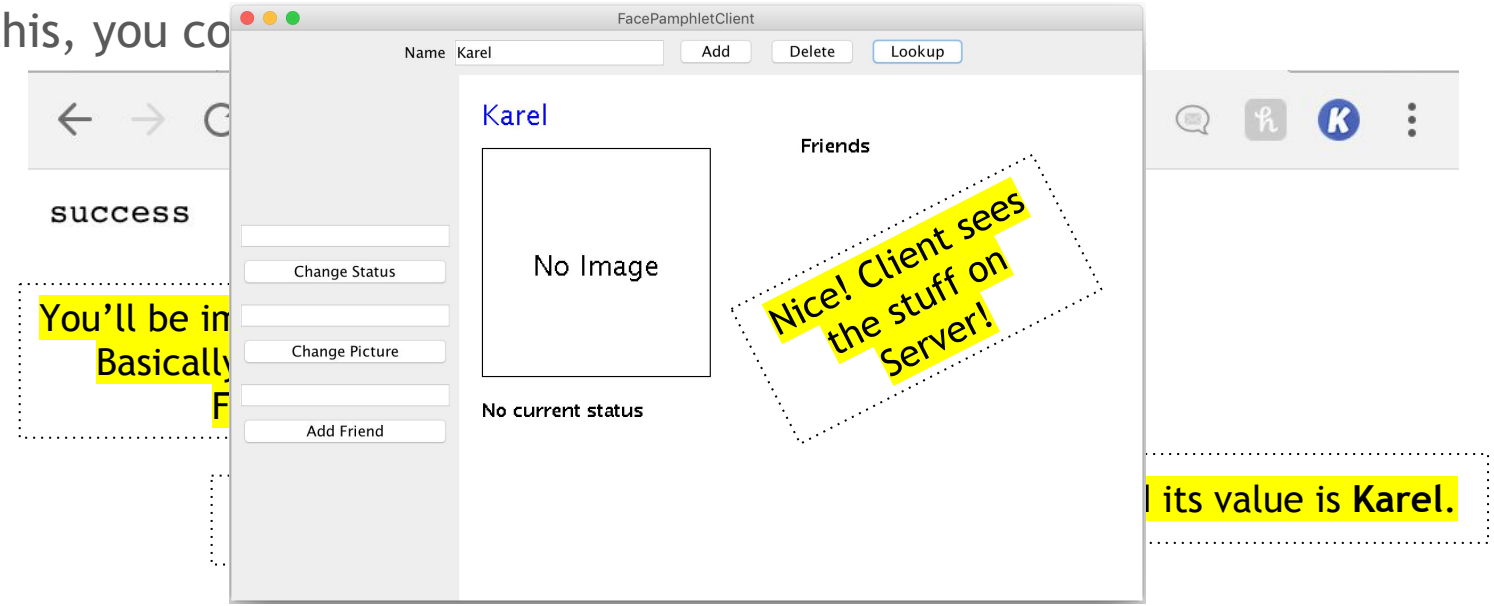
Using this, you could make manual commands to the server!



Testing: Making Commands

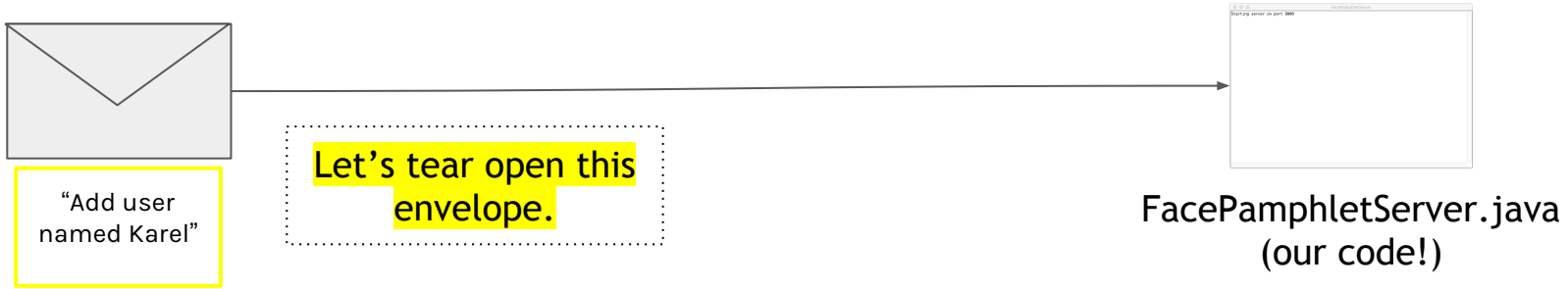
Everything following `localhost:8000/` is a **command**, and "**parameters**" of the command use this URL syntax: `?param=value`

Using this, you co



More about Requests

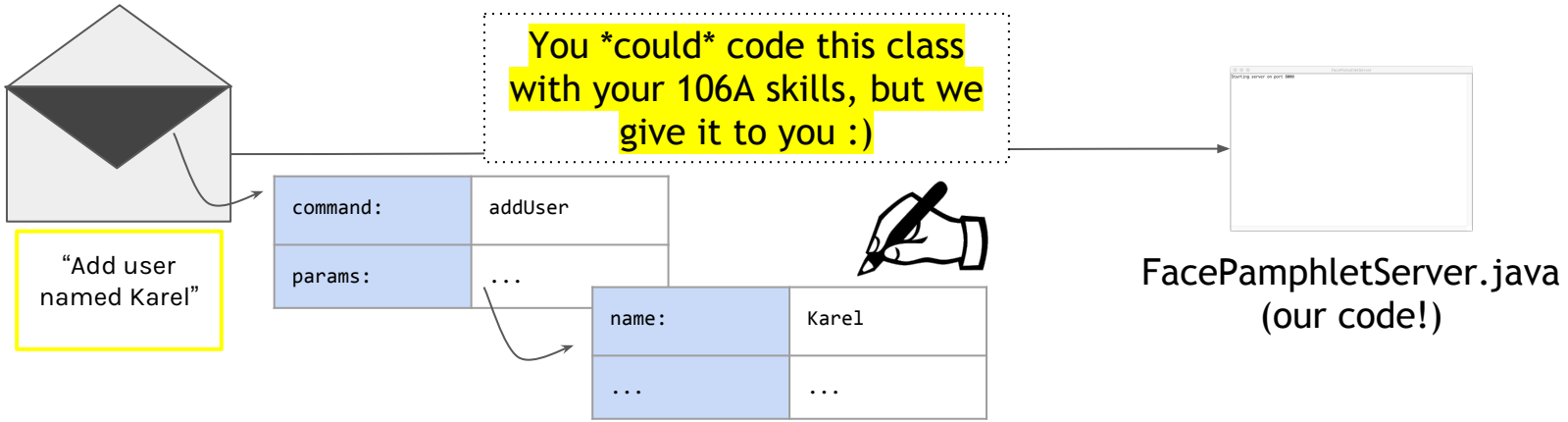
Cool! We know what a `Request` looks like in URLs and pictures. **What does it look like in code?**



More about Requests

Cool! We know what a `Request` looks like in URLs and pictures. **What does it look like in code?**

A `Request` is an object too! We don't need to know much about it, but it does store the **command** and the **parameters**. (What's a good way to store those parameters, you think??)



Syntax for Requests

We now know how requests look under the hood. **How do we get to that information?**

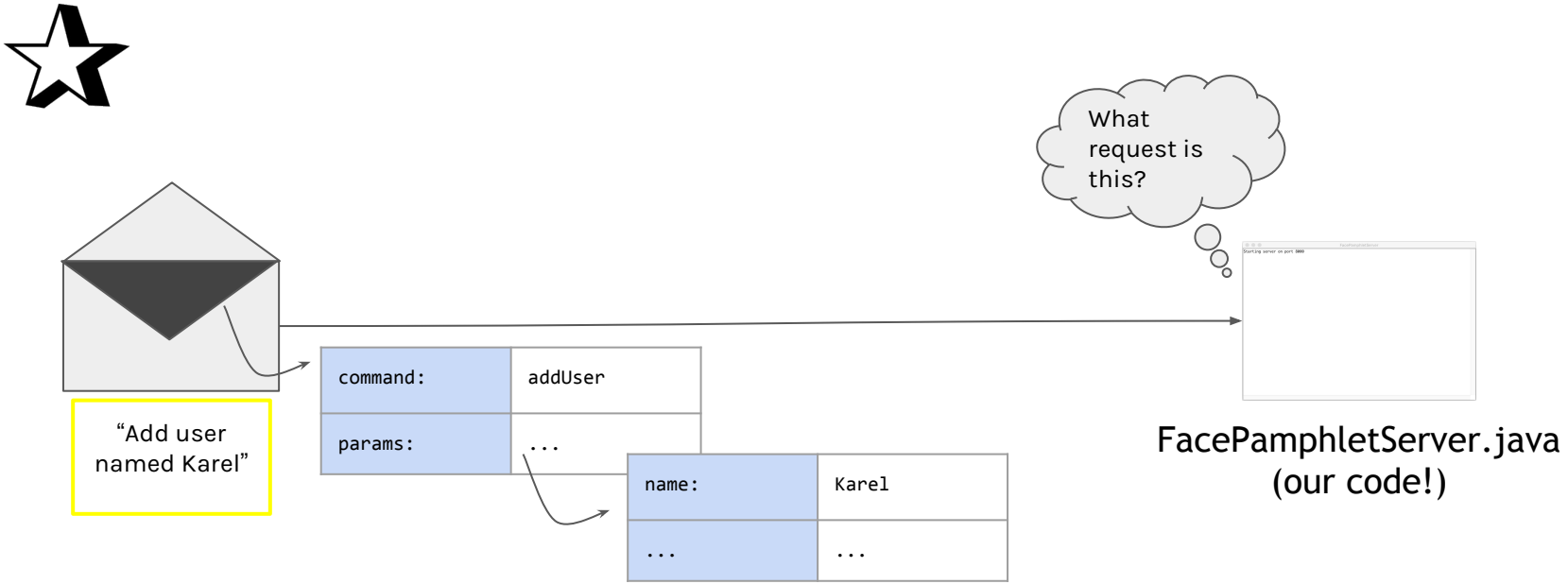
```
// Gets the command from a Request object
String cmd = request.getCommand();

// Gets a parameter from a Request object
// Note: You can only get parameters that exist.
//       How do we know which ones are available?
String userName = request.getParam("name");
```

These all return strings! Hm...

Responding to Requests

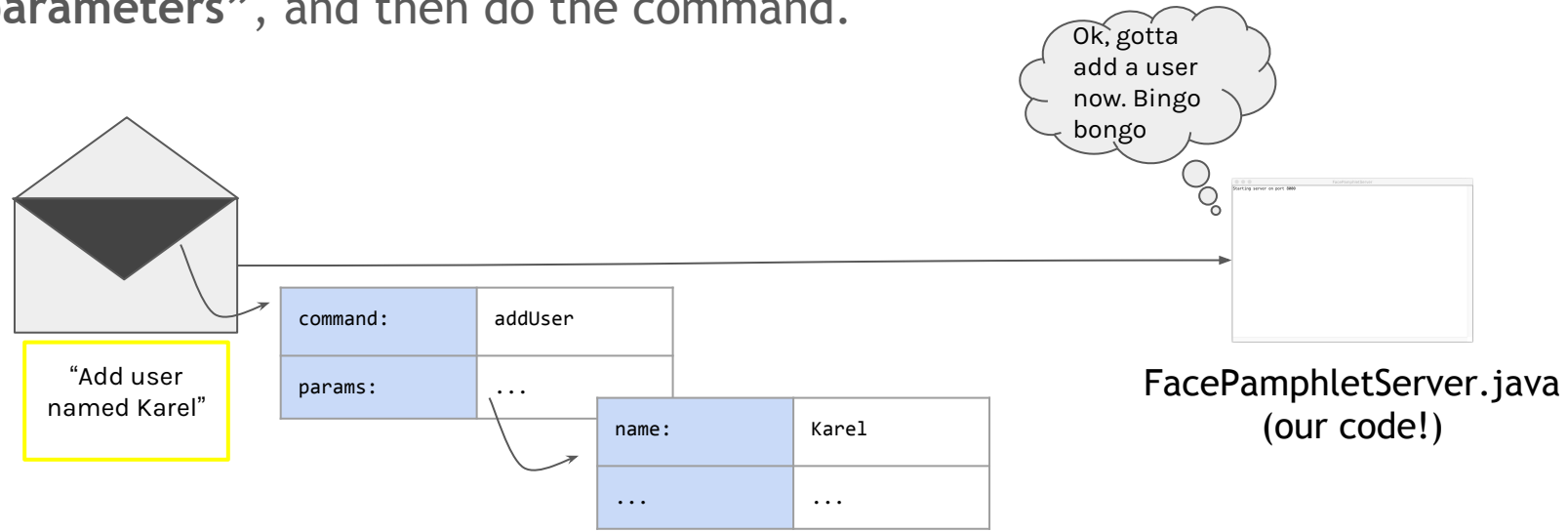
Now that we know how to access `Requests`, can we figure out how to respond to them?



Responding to Requests

Now that we know how to access `Requests`, can we figure out how to respond to them?

All we have to do is look at what the **command** is, extract relevant **"parameters"**, and then do the command.



Breaking Up the Problem

First, let’s review classes and other content brought up in lecture.

Then, let’s understand how classes would be useful for solving the problem of storing data in FacePamphlet.

Last, let’s look at how requests are sent and **how to respond to them!**

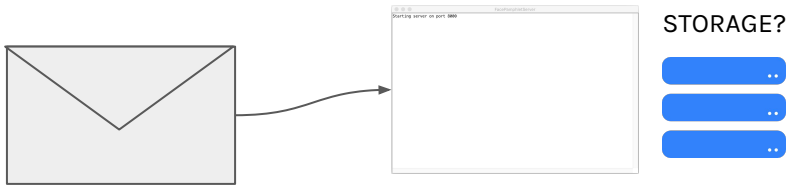
There are a Couple...

There are quite a few methods you have to implement for FacePamphlet.

We won't go over all of them, but we want you to understand how to get "parameters" out and operate on the data in `FacePamphletServer` at a high level.

- `ping`
- `addProfile, containsProfile`
- `deleteProfile`
- `setStatus, getStatus`
- `setImage, getImage`
- `addFriend, getFriends`

Remember, at a high level, these are the commands we're dealing with, and we're working in the server, receiving requests!



FacePamphletServer.java
(our code!)

Command	Parameters	Response
`addProfile`	`name`	Creates a profile with the given name. Returns "success" or, if the profile already exists, returns an error message.
`containsProfile`	`name`	Returns "true" if a profile with the given name exists, and "false" otherwise.
`deleteProfile`	`name`	Removes a profile from the database. Returns "success" or, if the profile doesn't exist, returns an error message.
`setStatus`	`name, status`	Sets the status of the user with the given name. Returns "success" or, if the profile doesn't exist, returns an error message.
`getStatus`	`name`	Returns the status of the user with the given name, or the empty string if the user exists but does not have a status. Returns an error message if the profile doesn't exist.
`setImage`	`name, imageString`	Sets the image for the user with the given name. Return "success" or, if the profile doesn't exist, returns an error message.
`getImage`	`name`	Returns the profile image of the user with the given name, or the empty string if the user exists but does not have an image. Returns an error message if
`addFriend`	`name1, name2`	Makes the user user with name "success", or a either user does not exist, b) if they are already friends, or c) if t
`getFriends`	`name`	Returns the list user with the g message if the p

The spec has the commands, their parameters, and their behavior mapped out!

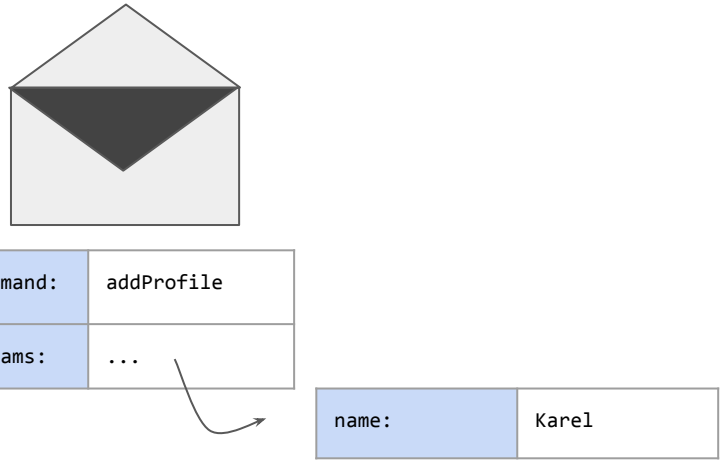
Commands have different numbers and names of parameters! Hmm...

Adding Profiles

Adding a profile means just adding it to the database.

There's only one **parameter**, the **name**!

What do we have to do?



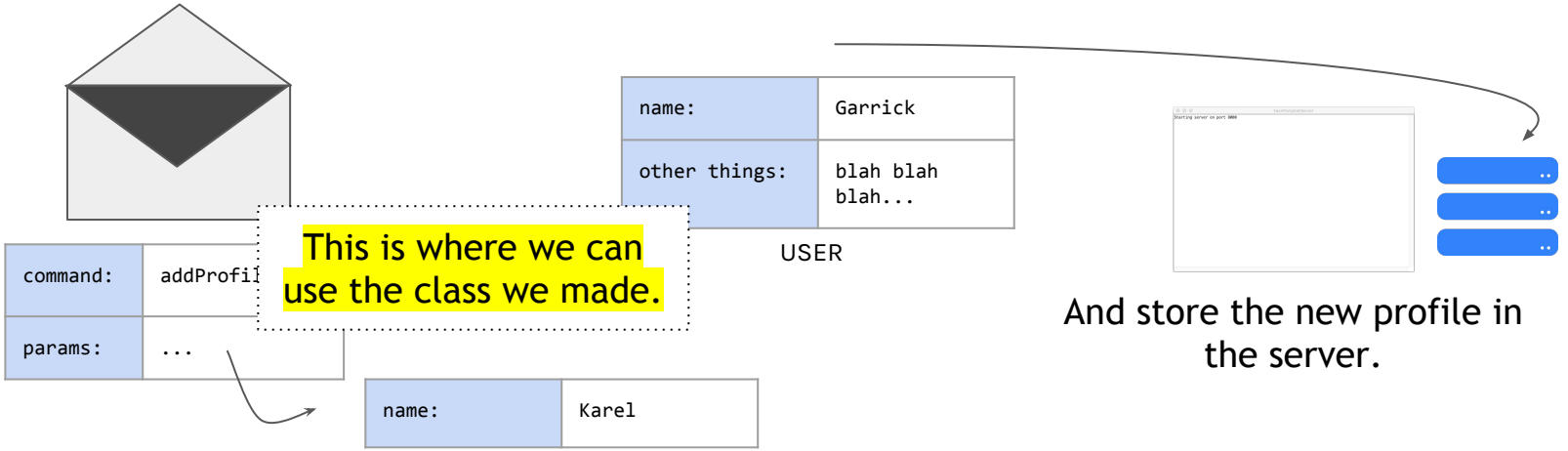
FacePamphletServer.java
(our code!)

Adding Profiles

Adding a profile means just adding it to the database.

There's only one **parameter**, the **name**!

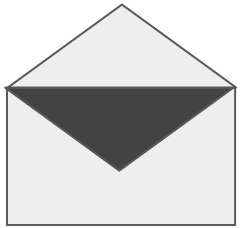
Idea: make a new **profile** corresponding to the user.



Fetching Profiles and Setting Info

Now say we want to get a profile (or check it exists). This is used by a good number of the commands!

If we have the **name parameter**, we can query the database for it.



command:	setStatus
params:	...

name:	Karel
status:	Turning right

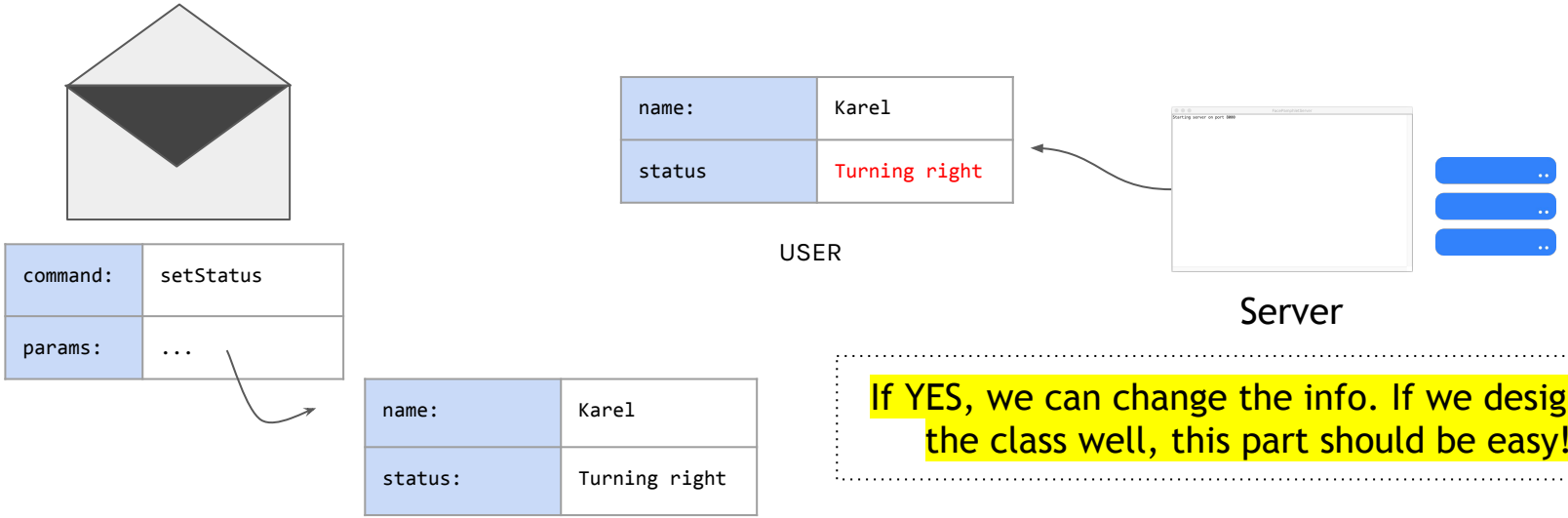
Does the server have an entry for Karel?

Server

Fetching Profiles and Setting Info

Now say we want to get a profile (or check it exists). This is used by a good number of the commands!

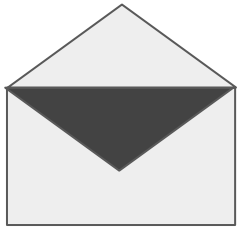
If we have the **name parameter**, we can query the database for it.



Fetching Profiles and Setting Info

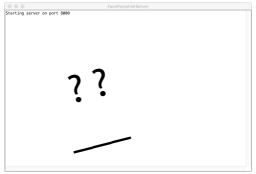
Now say we want to get a profile (or check it exists). This is used by a good number of the commands!

If we have the **name parameter**, we can query the database for it.



command:	setStatus
params:	...

name:	Karel
status:	Turning right



Server

What if NO: there is no user Karel in the database?

Giving Good Error Messages

Sometimes, we won't be able to find a profile in the database.

In that case, `requestMade()` still needs a response to send back. So let's send an error message! NOTE: Make sure error messages start with **"Error:"**

Make sure your error messages are informative! Some commands have multiple.

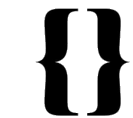
```
// requestMade() has to return a String
// If we can't find the profile, let's return an error
//    message instead, like so:
return "Error: Database does not contain a profile with name " +
    userName;
```

Setting Info: Images

One catch for setting info: sometimes, we need to deal with strings and GImages. **Requests and responses are made of Strings**, but **the server deals with and stores GImages**. Here are some methods to help translate:

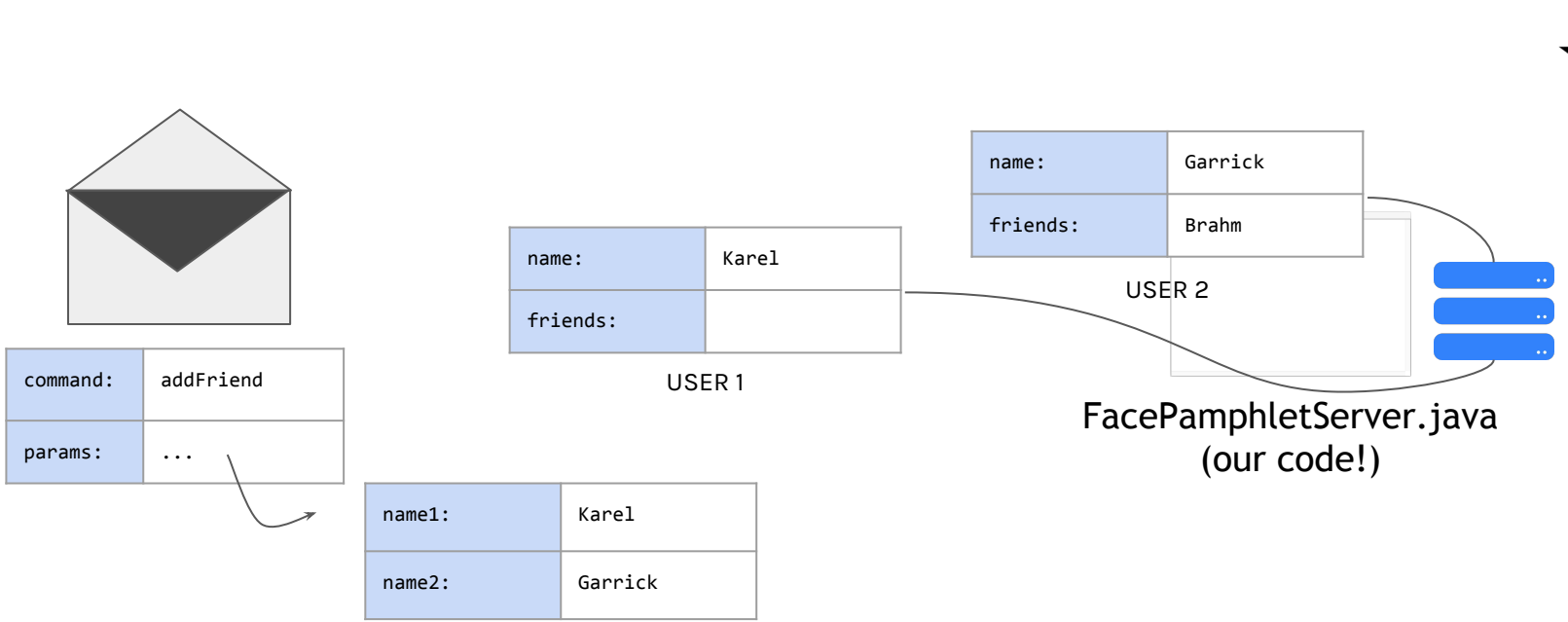
```
// Converts a GImage to its string representation
String SimpleServer.imageToString(GImage image)

// Converts a string representation of an image to a GImage
GImage SimpleServer.stringToImage(String str)
```



Add My Friends

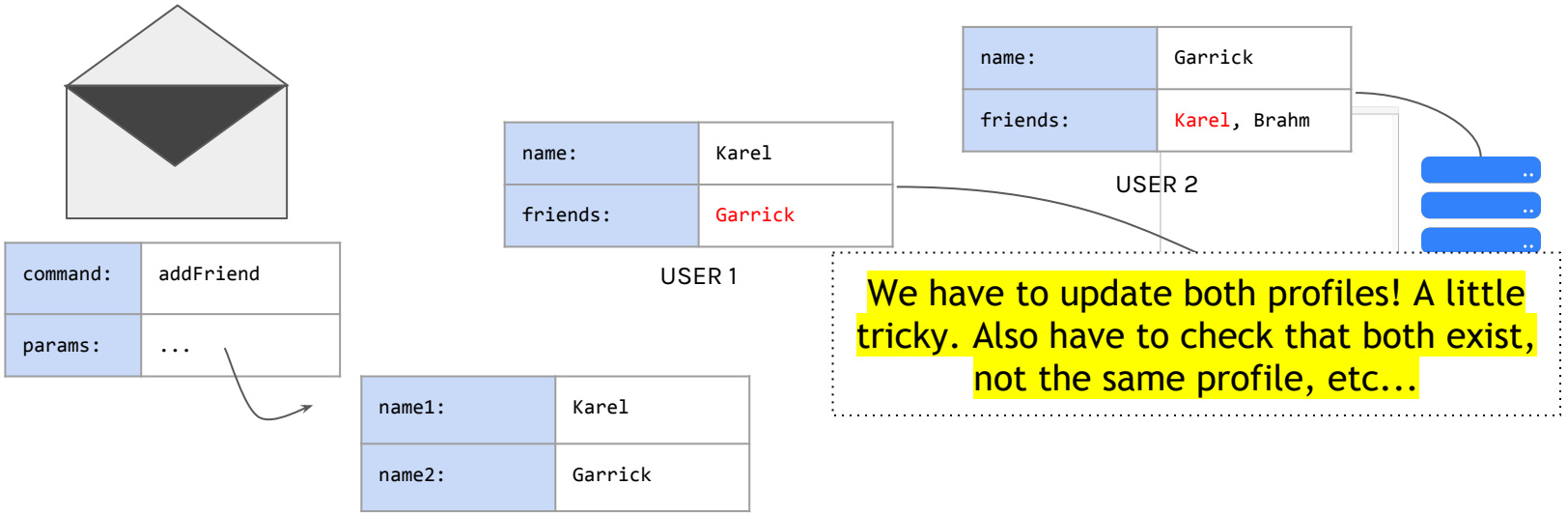
How do we add friends? Say we have two names, corresponding to users that want to be friends. What now?



Add My Friends

How do we add friends? Say we have two names, corresponding to users that want to be friends.

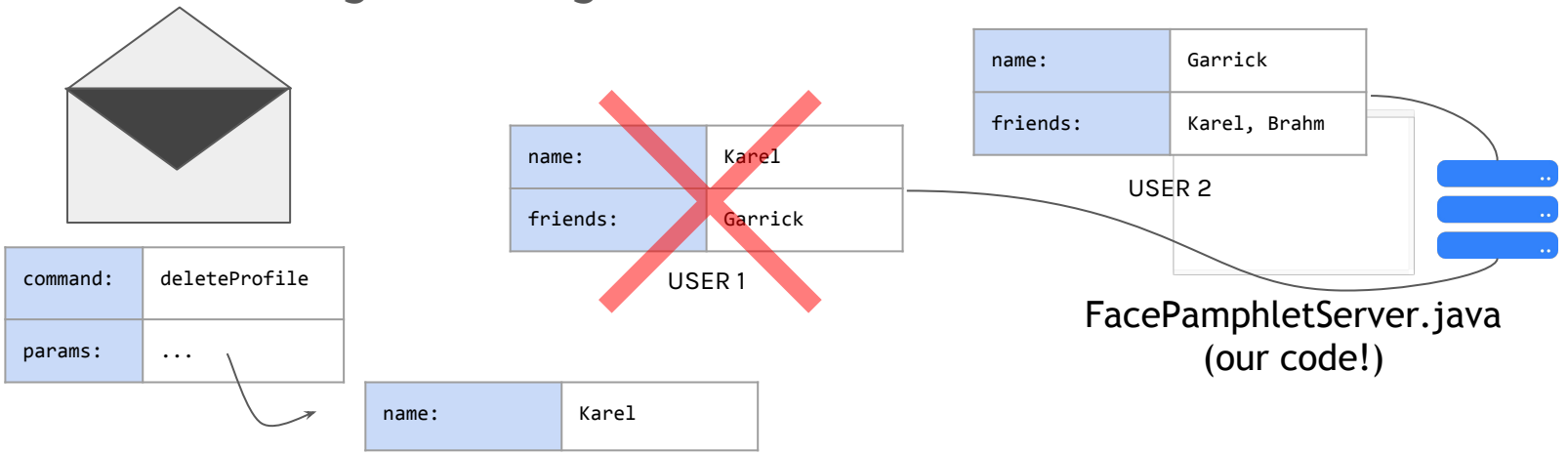
To add a friend, both users must have each other in their friends list.



Delete My Friends

What does it take to delete a user? If you look at Java documentation, you could find a good method to delete a user from the data structure you're using...

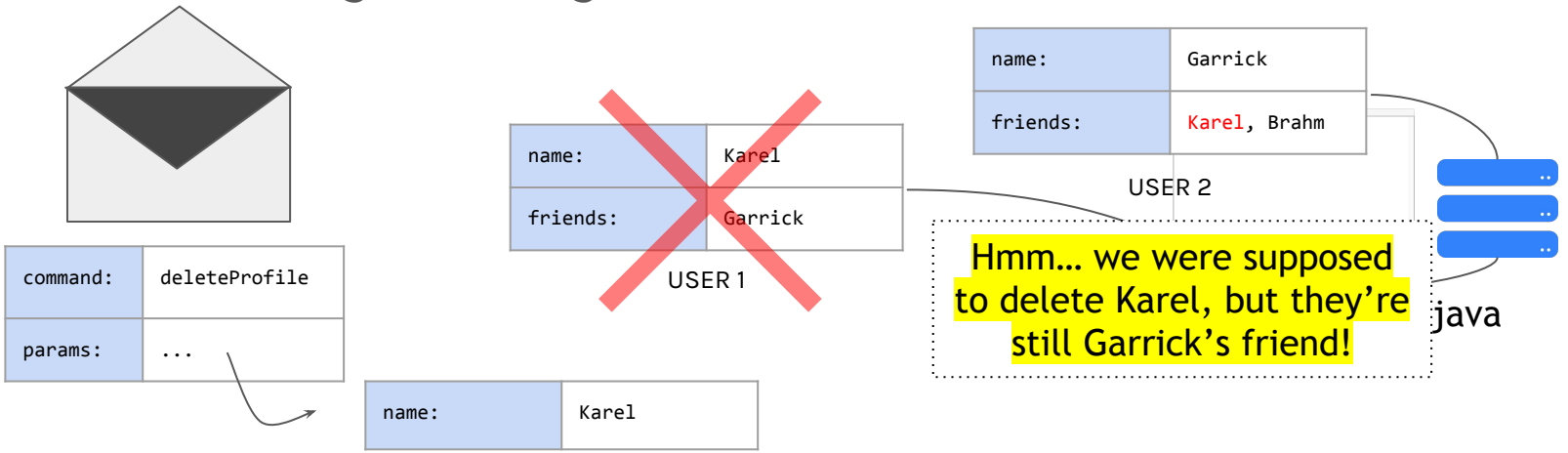
But are we missing something?



Delete My Friends

What does it take to delete a user? If you look at Java documentation, you could find a good method to delete a user from the data structure you're using...

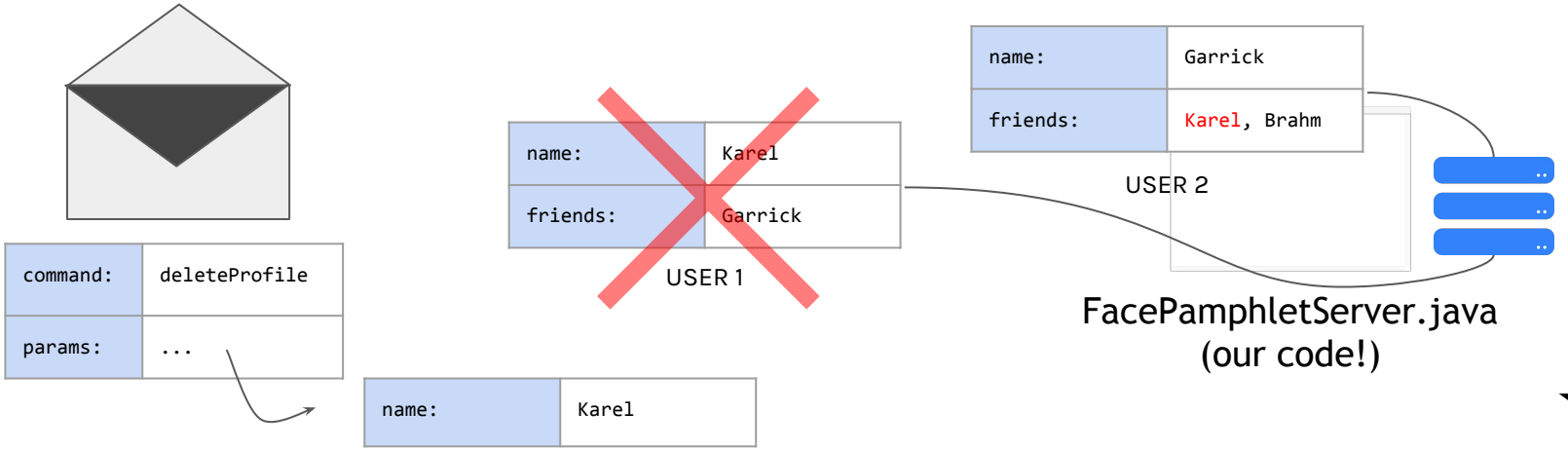
But are we missing something?



Delete My Friends

Ok, we know the problem with deleting a user is that **we also have to take them off other users' friend lists.**

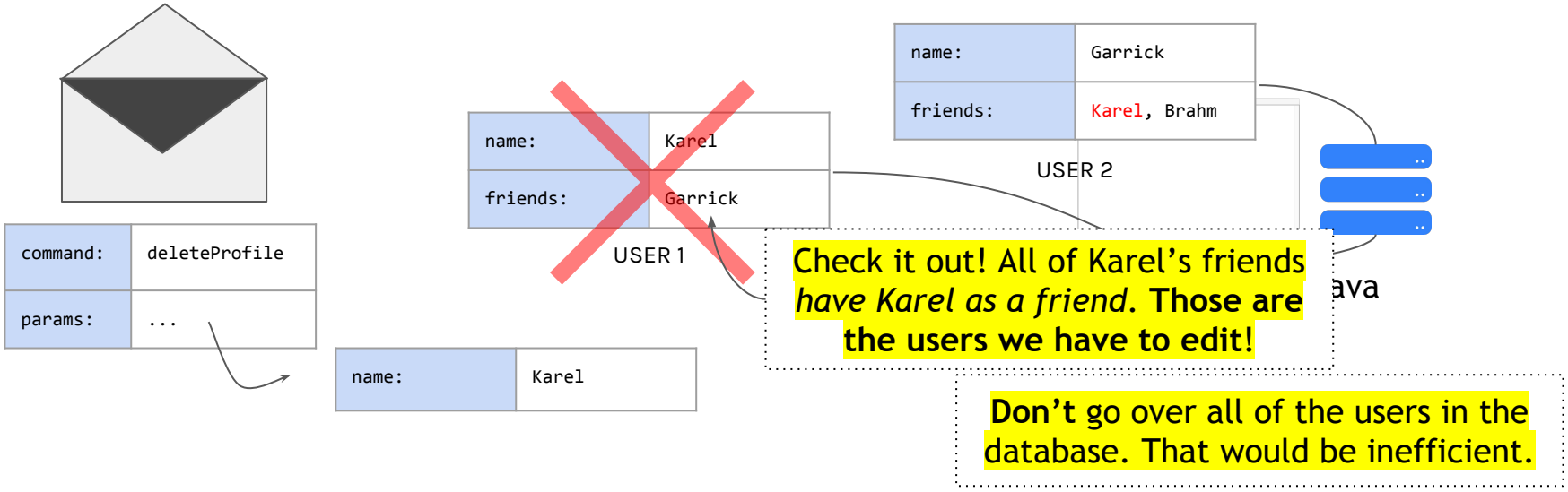
But, *there could be tens or hundreds of users in FacePamphlet*. How can we get just the ones we need to edit?



Delete My Friends

Ok, we know the problem with deleting a user is that **we also have to take them off other users' friend lists.**

But, there could be tens or hundreds of users in FacePamphlet. How can we get just the ones we need to edit?



Breaking Up the Problem

First, let’s review classes and other content brought up in lecture.

Then, let’s understand how classes would be useful for solving the problem of storing data in FacePamphlet.

Then, let’s look at how requests are sent and how to respond to them!

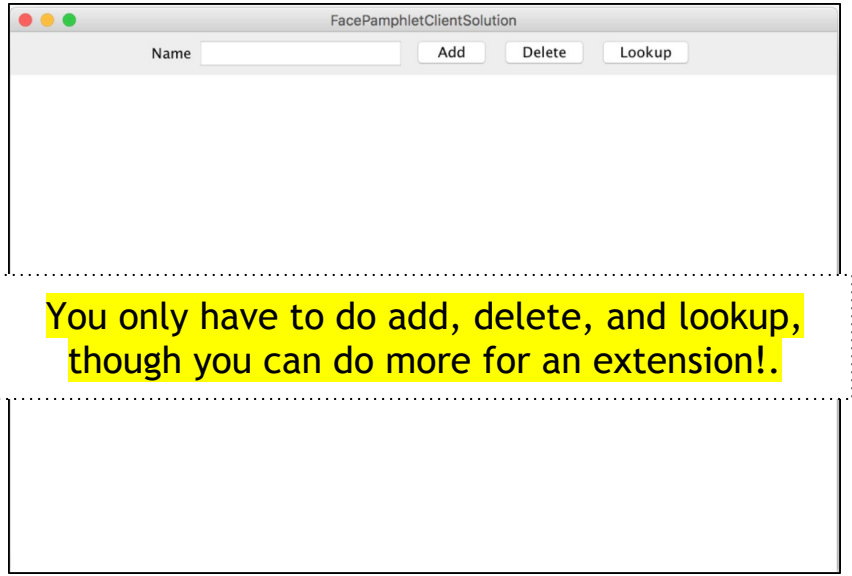
Last, let’s wrap back to the client and make a simple one!

Making the Client

The client we want you to make is a simplified version of the client we provide.

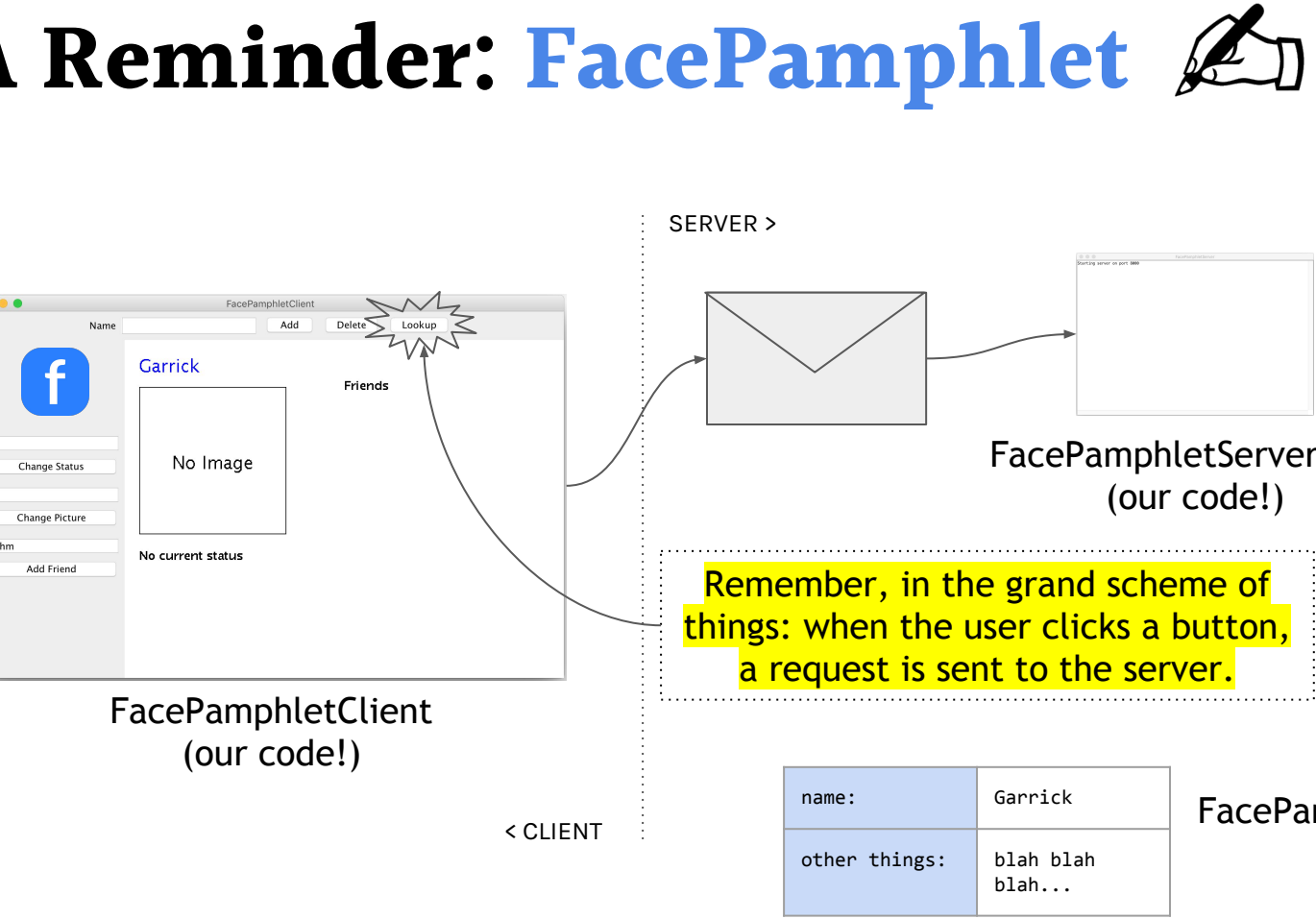
Old concepts: interactors, graphics/canvas

New concepts: **generating and sending requests**!



You only have to do add, delete, and lookup, though you can do more for an extension!.

A Reminder: FacePamphlet



Client: Making Requests

We provide methods for making and sending requests. Take a look at the `pingTheServer` method for clues on how to do it (`HOST` is a provided constant).

```
// Let's prepare ourselves a new request with command "ping".
Request example = new Request("ping"); // This is in the spec!

// (If we wanted to add a parameter, this is how we'd do it)
example.addParam("key_name", "value_name");

// We are now ready to send our request
String result = SimpleClient.makeRequest(HOST, example);
```

Client: Making Requests

Some requests return errors—remember how we did the code in server? To react to an error, we need a **try-catch block** for `makeRequest()`.

```
try {
  Request myRequest = // ...make request & add params
  String response = SimpleClient.makeRequest(HOST, myRequest);
  // if we get here, the request was successful – continue on...
} catch (IOException e) {
  // if we get here, there was an error
  String errorMessage = e.getMessage();
  // Do something with errorMessage
}
```

Tips and Tricks

Read the documentation! It's on the course website.

You should decompose in the server. Not everything should be in `requestMade()`.

When testing, you can run our versions of the server and client to help isolate bugs.

Good Luck!