Shahidi YEAH Hours

Gaby Candes, Andrew Tierno, and Peter Maldonado

Outline For Today

- Classes
- Server/Client Review
- Shahidi Overview
- Tips and Tricks

Classes

106A, 106B, ...

Classes in Review

Sometimes we want to create our own custom variable types!

We already designed classes for NameSurfer (so look back to that for inspiration)

We need to define:

- What the class stores (instance variables)
- What are its behaviors (methods)
- How do we make one (constructor)

Museum.java

```java
public class Museum {
	//What the class stores
	private String museumName;
	private HashMap<String, GImage> paintings;

	//How do we make one
	public Museum(String museumName) {
		this.museumName = museumName;
		this.paintings = new HashMap<String, GImage>();
	}

	//What are its behaviors
	public void addPainting(String name, GImage image) {
		this.paintings.put(name, image);
	}

	public GImage getPainting(String name) {
		if (!this.paintings.containsKey(name)) {
			System.out.println("No record of " + name);
			return null;
		}
		return this.paintings.get(name);
	}

	public String getName() {
		return this.museumName;
	}
}
```

MainProgram.java

```java
public class MainProgram extends GraphicsProgram {
	public void run() {
		Museum sfMoma = new Museum("SF Moma");
		GImage p1 = new GImage("Abstraction.png");
		GImage p2 = new GImage("NationalVelvet.png");

		sfMoma.addPainting("Abstraction", p1);
		sfMoma.addPainting("National Velvet", p2);

		System.out.println("Showing " + sfMoma.getName());

		add(sfMoma.getPainting("Abstraction"));
		add(sfMoma.getPainting("National Velvet"));
	}
}
```

Server/Client Review

How does the internet work?

How Internet?

The internet is basically comprised of computers yelling at each other

The first computer that yells is called the client

The computer that yells back is called the server

How do they yell?

Computers yell by sending each other specially formatted Strings called Requests

These specially formatted Strings have two parts, the command and the params

Let's try it out!

Command	Params	Returns
`multiply`	`a, b`	Returns the product of `a` and `b`
`replicate`	`phrase, ntimes`	Returns `phrase` concatenated to itself `ntimes` times.

MyCoolServer.java

```
public String requestMade(Request request) {
    String command = request.getCommand();
    if (command.equals("multiply")) {
        return multiply(request);
    } else if (command.equals("replicate")) {
        return replicate(request);
    } else {
        println("Received unknown command " + command);
        return "Error: No such command called " + command;
    }
}
```

MyCoolServer.java (continued)

```
public String multiply(Request request) {
    double a = Double.parseDouble(request.getParam("a"));
    double b = Double.parseDouble(request.getParam("b"));
    println("Multiplying " + a + " and " + b);
    //Convert the double back to a String
    return "" + (a * b);
}
```

Command	Params	Returns
`multiply`	`a, b`	Returns the product of `a` and `b`
`replicate`	`phrase, ntimes`	Returns `phrase` concatenated to itself `ntimes` times.

MyCoolServer.java (continued)

```
public String replicate(Request request) {
    String phrase = request.getParam("phrase");
    int numTimes = Integer.parseInt(request.getParam("ntimes"));
    String output = "";
    println("Replicate " + phrase + "x" + numTimes);
    for (int i = 0; i < numTimes; i++) {
        output += phrase;
    }
    return output;
}
```

Command	Params	Returns
`multiply`	`a, b`	Returns the product of `a` and `b`
`replicate`	`phrase, ntimes`	Returns `phrase` concatenated to itself `ntimes` times.

Shahidi

```
Assignment[] cs106A = new Assignment[7];
//You got this!
Assignment shahidi = cs106A[cs106A.length - 1];
```

Milestone #1

Implementing the `Coordinate` class.

Should remind you ***a lot*** of implementing `NameSurferEntry`.

The constructor should take `String`s of the format `longitude, latitude` like

```
Coordinate c = new Coordinate("37.4, 122.1");
```

This will be made a lot easier if you choose the right pieces of information to store in the right variable types!

Do **NOT** call any of your variables `long` (since `long` already is a variable type in Java like `int` or `float` so trying to use it as a variable name will cause weird errors).

Milestone #1

Implementing the `Coordinate` class.

Coordinate strings will always be the format “`latitude, longitude`”

This applies to

Milestone #2

Implementing the `ping` request. Look back at the `requestMade` example for inspiration!

A great way to test if your server is working or not is by typing:

`localhost:8080/ping`

in your browser while you have the server running.

Command	Params	Returns
`ping`	*none*	`"Hello, internet"`

Milestone #3

Implementing the `distanceBetween` request.

Remember that the parameters you get from the server are strings, so you'll need to convert them into a different type if you want to use the `distanceTo` method you wrote in the `Coordinate` class (and you probably want to use this method).

Command	Params	Returns
`distanceBetween`	`coord1, coord2`	Returns the distance (in kilometers) between the two coordinates.

Milestone #4

Implementing the `Event` class.

The design is 100% up to you! We don't tell you what methods/instance variables to store (though looking ahead to Milestones 5 might help).

You *must* keep track of the text description of the `Event`, its location (*cough cough* how do we store locations *cough*), and its image.

Keep in mind that the `Event` may or may not have an image associated with it.

There is a "Tags" field in the client for uploading hashtags, this is an ***optional*** extension.

Milestone #5

This is a fairly large milestone!

You'll be implementing most of the server logic here. Servers are responsible both for acting as a database for information (think back to `NameSurferDatabase`), and yelling answers to the client in response to yelled requests.

Take some time to think about the best way to keep track of `Event`s.

Remember that each `Event` needs to be associated with an ID (you can get a new one by calling `getNextID()`).

Milestone #5

Implementing the `addEvent` request.

Adds the `Event` with the given `description`, `coordinates`, and `image`. The details of the implementation will depend on how you chose to keep track of `Event`s.

The coordinates parameter is formatted as `"50, 12"`. What does this look like?

Command	Params	Returns
addEvent	description, coordinates, image (optional)	Returns the eventID associated with this event.

Milestone #5

More on the `addEvent` request.

The `image` parameter is optional, you can check if it is present or not with:

```
request.hasParam("image"); //returns a boolean
```

The `image` parameter is a `String`, to reconstruct the `GImage` it represents, call:

```
GImage img = SimpleServer.stringToImage(request.getParam("image"));
```

Command	Params	Returns
addEvent	description, coordinates, image (optional)	Returns the eventID associated with this event.

Milestone #5

Implementing the `deleteEvent` request.

Deletes the `Event` with the given `eventID`. The details of the implementation will depend on how you chose to keep track of `Event`s.

Command	Params	Returns
`deleteEvent`	`eventID`	Returns “success” if the event was successfully deleted, or an `error` message if it doesn’t exist.

Milestone #5

Implementing the `getDescription` request.

Remember that an `eventID` is not an `Event`, and you'll want the latter if you want to use the methods you wrote in the `Event` class. Think about how you can get an `Event` from an `eventID`.

Command	Params	Returns
`getDescription`	`eventID`	Returns the description of the event with the given id, or an `error` message if the event doesn't exist.

Milestone #5

Implementing the `getCoordinates` request.

Remember that servers can only return strings, so you'll want to make sure that whatever you return is a string, like: `"35, 183"`. Where have you seen strings formatted like this before?

Command	Params	Returns
`getCoordinates`	`eventID`	Returns the coordinates of the event with the given id, or an `error` message if the event doesn't exist.

Milestone #5

Implementing the `getImage` request.

Remember that servers can only return strings. Convert a `GImage` into the corresponding `String` with:

```
String imgString = SimpleServer.imageToString(image);
```

If the `Event` doesn't contain an image you should return empty string (`""`)

Command	Params	Returns
getImage	eventID	Returns the image of the event with the given id, or an error message if the event doesn't exist.

Milestone #6

Implementing the `nearbyEvents` request.

The grand culmination of previous milestones. If it seems like you're missing information to complete this task, you might need to redesign your `Event` or how you store `Event`s in the `ShahidiServer`!

Make sure to run `ShahidiServerTests` to confirm that everything is working!

Command	Params	Returns
nearbyEvents	coordinates, radius	Finds all events within radius km of coordinates and returns their IDs as a list string.

Tips and Tricks

Questions?