Variable Usage,
Making HTTP requests,
Exceptions

How hard was second code review assignment?

A) Easy

B) Moderate

C) Challenging

D) Unreasonable

How long did second assignment take?

A) Less than 2 hours

B) 2 to 4 hours

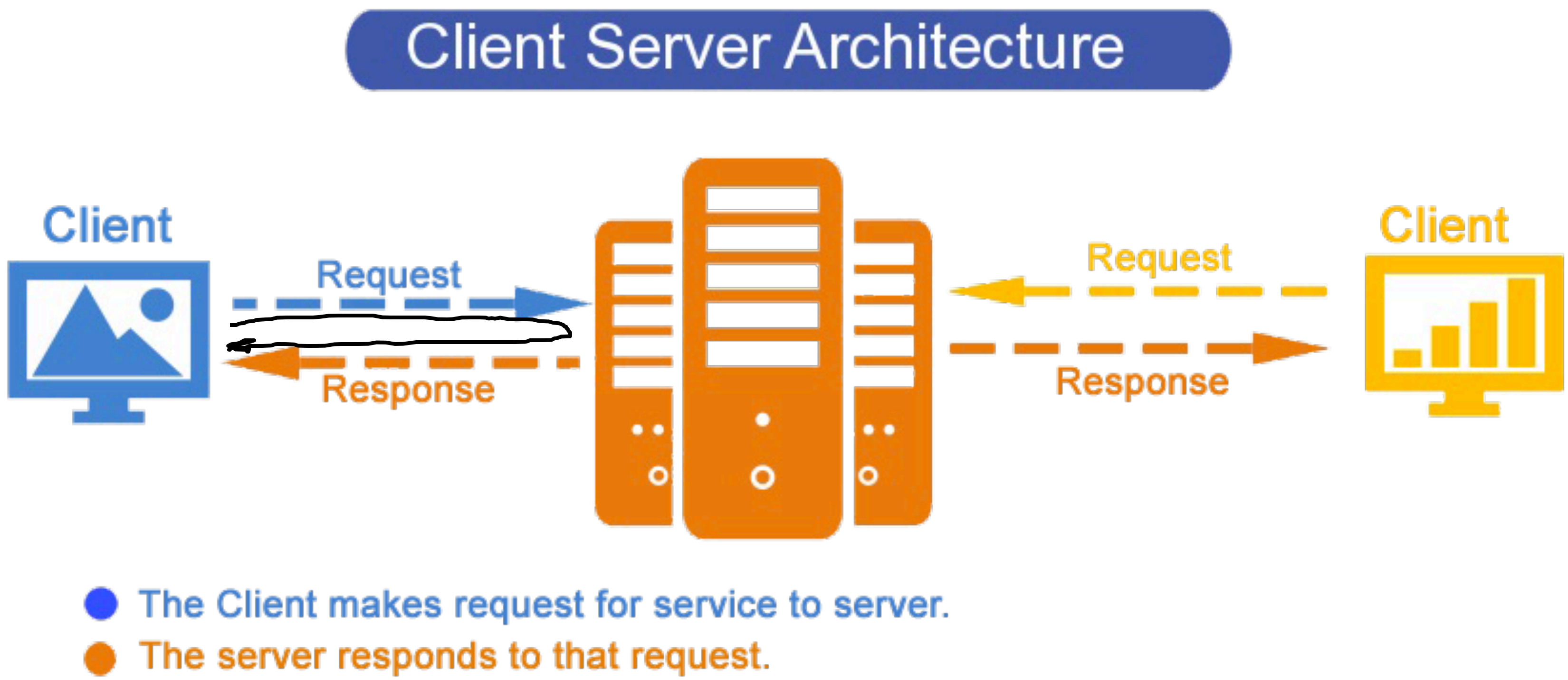
C) 4 to 6 hours

D) 6 to 8 hours

E) More than 8 hours

Client / Server Architecture

- **Server maintains durable state**
- **Clients connect to server to access/modify state**
  - Server supports multiple clients simultaneously



Server state typically kept as "tables"

- **Managed by databases**
  - E.g., Movies table at themoviedb.org

id	title	original_title	original_language	popularity	vote_count	vote_average
			~~Spanish~~ english			

- **We won't do server-side stuff in CS 126.**

Application Programming Interfaces (API)

- APIs are the interface between the client and server
- Like everything, there are good & bad APIs
- Some best practices to be aware of

RESTful APIs

- REST = Representational State Transfer
- Stateless, client-server, cacheable communications protocol
- Generally built on HTTP/HTTPS
- Supports Creating, Reading, Updating & Deleting (CRUD)
- Language/Platform independent
- REST operations should be self-contained

- Nice overview:
  - https://twincl.com/programming/*6af/rest-api-design

REST Concepts: Verbs

caching

- **Standard Verbs:**
  - GET: Read an existing resource (IDEMPOTENT, SAFE)
  - POST: Create a new resource ()
  - PUT: Update an existing resource (IDEMPOTENT)
  - DELETE: Delete an existing resource (IDEMPOTENT)
- **Other Standard Verbs:**
  - OPTIONS: Get list of what verbs are allowed(IDEM, SAFE)
  - HEAD: Get headers (e.g., metadata) of GET (IDEM, SAFE)
  - PATCH: Update part of existing resource ()
- **You can create your own, but some discourage it**

REST Concepts: Nouns (a.k.a. resources)

- **Mostly Describe Resources**
  - /movies, /movies/17, /movies/17/reviews
  - Prefer plurals, use numbers to index into collection
  - Narrow selection through use of query string
    - /movies?genre=13&year=2015

key value

- **Can also specify utility APIs**
  - /search?q=keyword&order=alphabetical
    - *Note: query parameters:* key=value pairs
    - separated from resource by a **?**
    - separated from each other by **&**

Interacting with a REST API

- **Hyper Text Transport Protocol (HTTP)**
  - Client makes requests, server responds
- **Uniform Resource Locators (URL) to specify server/resource**
  - http://host.name.here:port/path/to/resource (or https)
  - Port defaults to 80
- **Requests:**
  - A verb (e.g., GET), a URL, and an HTTP version under the covers.
  - Request headers and optional message body
- **Responses:**
  - Status code: 200 (OK), 401 (Unauthorized), 404 (Not Found)
  - Response headers and optional message body

In Java

- **Easy with the power of libraries**
  1. java.net.URL*

```
URL url = new URL("http://google.com");   // throws MalformedURLException
InputStream inStream = url.openStream();
InputStreamReader inStreamReader =
    new InputStreamReader(inStream, Charset.forName("UTF-8"));
```

*https://docs.oracle.com/javase/7/docs/api/java/net/URL.html

Even easier with better libraries

```
http://unirest.io/java.html

HttpResponse<String> response =
        Unirest.get(url)
        .header("user-key", Zomato.API_KEY)
        .asString();

if (response.getStatus() == 200) {
    String json = response.getBody();
    ...
}
```

Announcements

- Check your moderator assignment. It might have been updated earlier this week. Go to the right code review!

- Make sure that you are doing your own work! We look for plagiarism (sharing of code).

- Make sure that you protect your code. Don't publish it in any way. You are liable for facilitating plagiarism if you do.

If statements

- **What is wrong with:**

```
if (winner == true) {
    aFunctionCall(arg1, arg2);
}
```

Which is better?

A

```
if (count != 3) {
      return false;
}
return true;
```

B

```
if (count == 3) {
      return true;
}
return false;
```

C) Both are fine

return (count == 3);

D) Both are lacking

```
if (condition) {
   return true;
} else {
   return false;
}
```

How can this be improved?

```
if (func1(arg1, arg2)) {
    return true;
} else if(func2(arg1, arg2)) {
    return true;
}
```

```
}
if (func1(arg1, arg2) || func2(arg1, arg2)) {
    return true;
}
```

What is wrong with this?

```
if (c.equals("*") ||
    c.equals("/") ||
    c.equals("+") ||
    c.equals("-")) {
```

Are the following two things the same?

true

```
if (func1(arg1, arg2) ||
    func2(arg1, arg2)) {
    return true;
}
```

if side-effecting

```
bool isGood = func1(arg1, arg2);
bool isOkay = func2(arg1, arg2);
if (isGood || isOkay) {
    return true;
}
```

A) Same B) Different C) I don't know

Short-circuit evaluation

- Java stops evaluating conditionals as soon as knows the outcome:

false

- `if ((0 == 1) && willNeverBeCalled()) { …`
- `if ((1 == 1) || willNeverBeCalled()) {  …`

true

- In the above, the function will never be called.

- Short-circuiting can be useful
  - `if ((pointer != null) && (pointer->field == VAL)) {`
- Be judicious on when you rely on short circuiting

```
switch (type) {

    // checks if the type is an open parenthesis
    case "(" :

        // comment relating to the condition
        if (someCondition) {

            // comment explaining why this return value
            return false;
        }

        if (anotherCondition) {

            // another explanatory comment
            return aFunctionCall(arg1, arg2, arg3);
        }

        // comment relating to final return value possibility
        else return (anotherFunctionCall(anotherArg1));

    // if type is a closing parenthesis
    case ")":

        // comment relating to this different condition
        if (aDifferentCondition) {

            // explanation of this return value
            return true;
        }

        if (anotherConditionToConsider) {

            // comment relating to this return value
            return aDifferentFunctionCall(arg1, arg2, arg3);
        }

        // checks if the next value in the input is an operator and returns accordingly
        else return aValue;
```

```
switch (type) {

    // checks if the type is an open parenthesis
    case "(" :
        // comment relating to the condition
        if (someCondition) {
            // comment explaining why this return value
            return false;
        }
        if (anotherCondition) {
            // another explanatory comment
            return aFunctionCall(arg1, arg2, arg3);
        }
        // comment relating to final return value possibility
        else return (anotherFunctionCall(anotherArg1));


    // if type is a closing parenthesis
    case ")”:
        // comment relating to this different condition
        if (aDifferentCondition) {
            // explanation of this return value
            return true;
        }
        if (anotherConditionToConsider) {
            // comment relating to this return value
            return aDifferentFunctionCall(arg1, arg2, arg3);
        }
        // checks if the next value in the input is an operator and returns accordingly
        else return aValue;
```

8-16% blank lines is “optimal”

Finish the sentence

- **Initialize each variable ...**

A) ~~as early as possible~~.

B) as it is declared.

C) ~~if necessary~~.

D) ~~before every use~~.

Which is better?

A

```
public void foo(int [] A) {
     for (int i = 0 ; i < A.length ; i ++) {
          …
     }
     …
     for (int i = 0 ; i < A.length ; i ++) {
          …
     }
}
```

B

```
public void foo(int [] A) {
     int i;
     for (i = 0 ; i < A.length ; i ++) {
          …
     }
     …
     for (i = 0 ; i < A.length ; i ++) {
          …
     }
}
```

Which is better?

A

```
int xPos = getPositionX();
int yPos = getPositionY();

int xTranslated = translate(xPos, XMATRIX);
int yTranslated = translate(yPos, YMATRIX);

System.out.println(xPos + "->" + xTranslated);
System.out.println(yPos + "->" + yTranslated);
```

B

```
int xPos = getPositionX();
int xTranslated = translate(xPos, XMATRIX);
System.out.println(xPos + "->" + xTranslated);

int yPos = getPositionY();
int yTranslated = translate(yPos, YMATRIX);
System.out.println(yPos + "->" + yTranslated);
```

Which is better?

A

```
String nameString = getName();
System.out.println(nameString + ":" + lookup(nameString));

nameString = getAlias();
markUsed(nameString);
return nameString;
```

B

```
String nameString = getName();
System.out.println(nameString + ":" + lookup(nameString));

String aliasString = getAlias();
markUsed(aliasString);
return aliasString;
```

What (all) is wrong with this code?

```
public class Variables {  final
    private static String INVALID invalid = "INVALID";
    private static String [] args;
    public static final int MAX_LENGTH = 5;

    public static String handleIt(String argString) {
String [] args = argString.split(" ");
        return handleArgs(); args);
    }

?   public static String handleArgs() { String [] args) {
        int strLength;
        for (int i = 0; i < args.length; i++) {   <- for each
          int strLength = args[i].length();
            if (strLength > 5) { MAX_LENGTH) {
                return args[i] + ": " + strLength;
            }
        }
        return invalid; null;
    }
}
```

Read chapters 5 & 6 (Commenting) in book