Variable Usage,
Making HTTP requests,
Exceptions
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

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

B

```java
public void foo(int [] A) {
    int i;
    for (i = 0 ; i < A.length ; i ++) {
        if (!A[i].error()) { break; }
    }
    ...
    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

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

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

B

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

String aliasString = getAlias();
markUsed(aliasString);
return aliasString;
```
What (all) is wrong with this code?

```java
class Variables {
    private static String invalid = "INVALID";
    private static String[] args;

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

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

Downloading from the internet

- **Hyper Text Transport Protocol (HTTP)**
  - Client-Server Model
    - 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

- **Takes a “command”: GET is standard for getting data**
  - Assumes idempotency, can be cached

- **Returns:**
  - Status code: 200 (OK), 401 (Unauthorized), 404 (Not Found)
  - Response body
In Java

- Easy with the power of libraries

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

2. com.google.gson.stream.JsonReader
   ```java
   JsonReader jsonReader = new JsonReader(inStreamReader);
   Gson gson = new Gson();
   Thing thing = gson.fromJson(jsonReader, Thing.class);
   ```

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

- Events that occur during program execution
- Disrupt the normal flow of the program
  - (e.g. divide by zero, array access out of bound, etc.).

- In Java, an exception is an object that wraps an error event
  - contains information about the error including its type

- Typically handled through the use of try/catch
- Important piece of the interface of a method
  - Method signature includes what exceptions it might throw
Kinds of Exceptions

```
+----------+
| Throwable |
+----------+
  /      /  \
 /      /\   \
+----------+  +----------+
| Error    |  | Exception |
+----------+  +----------+
  /      /  \
 /      /\   \
+-----------+
| unchecked |  | checked |
+-----------+
  /      /  \
 /      /\   \
+-----------+
|            |
| unchecked  |
+-----------+
```
To Dos for Thursday

- Read Ch. 8 (Defensive Programming)
- Figure out what URL pulls pages other than page 1 from themoviedb.org (i.e., read API documentation)
- Extend your project to load a specified number of movies
  - The # of movies is a parameter
  - Make server requests until you have enough movies
  - If the user requests more movies than there are, return as many as there are.
- Your project should have a main function that:
  - Takes: <# movies> <query type> [query parameter]
  - Prints to the System.out movie titles, 1 per line
- Cleanly handle exceptions & invalid inputs