Objectives

- Continuing Java Fundamentals
  - Control Structures
  - Scope
  - Arrays
  - Command-line arguments

1

Review: Java's Library

- What are some examples of Java classes?
- How do you create an object?
- How do you call a method?
- How do we know what methods are available to call on a specific Java class?
- What classes are included by default in a Java program?
  - How can we use classes that aren't included?
- What are some helpful String methods?

2

Assign 1

- Problems?
- Tips or tricks for others?
  - Read: what mistakes will you vow never to do again but probably will?
- Debugging part – shows that you shouldn't write a lot of code before compiling!
- Goal: Comfort with API
  - Important that you understand how to read the API so that you can easily leverage the library
  - Library is well-tested and efficient!

Assignments Feedback

- Recall: Class comments are required
- High-level description first

```
/**
 * This program finds the file type when
 * the user inputs the name of the file.
 * @author Sara Sprenkle ⟵
 */
```

- Comment for author: `@author Dr. Seuss`
  - Syntax will make more sense when we talk more about JavaDocs
  - Needs to be ***last*** in the comment

CONTROL STRUCTURES

Conditional Statements Syntax

```
if (condition) {
    body
}
```

- `if` statement
  - **Condition** must be surrounded by ( )
  - Condition must evaluate to a `boolean`
  - If body includes *multiple* statements, **must** be enclosed by { }

```
if (purchaseAmount < availCredit) {
   System.out.println("Approved");
   availableCredit -= purchaseAmount;
}
else
   System.out.println("Denied");
```

Don't *need* { } if only one statement in the body, but **Best practice**: use { }

Control Flow: Conditional Statements

- `if` statement

Condition

```
if (purchaseAmount < availCredit) {
   System.out.println("Approved");
   availableCredit -= purchaseAmount;
}
else
   System.out.println("Denied");
```

Block of code

- Everything between { } is a block of code and has an associated ***scope***

7

Logical Operators

Operation	Python	Java
AND		&&
OR		||
NOT		!

In Python, these are …?

8

Logical Operators

Operation	Python	Java
AND	and	&&
OR	or	||
NOT	not	!

Python Gotcha: Scoping Issues

- Everything between { } is a block of code and has an associated ***scope***

```
if (purchaseAmount < availableCredit) {
      availableCredit -= purchaseAmount;
      boolean approved = true;
}

if( ! approved )
      System.out.println("Denied");
```

Out of scope
Will get a compiler error (cannot find symbol)

How do we fix this code?

Not Fixed

```
if (purchaseAmount < availableCredit) {
      availableCredit -= purchaseAmount;
      boolean approved = true;

      if( ! approved )  Will never execute
            System.out.println("Denied");
}
```

Almost Fixed

- Move `approved` outside of the `if` statement

```
boolean approved;
if (purchaseAmount < availableCredit) {
      availableCredit -= purchaseAmount;
      approved = true;
}

if( ! approved )
      System.out.println("Denied");
```

```
Compiler error: variable approved
might not have been initialized
```

Fixing Variable Scope Problem

- Move `approved` outside of the `if` statement *and* initialize

```
boolean approved = false;
if (purchaseAmount < availableCredit) {
        availableCredit -= purchaseAmount;
        approved = true;
}

if( ! approved )
        System.out.println("Denied");
```

13

String Comparison: `equals`

- `boolean equals(Object anObject)`
  - Compares this string to the specified object

```
String string1 = "Hello";
String string2 = "hello";
boolean test;
test = string1.equals(string2);
```

- `test` is false because the Strings contain different values
- Note that `==` does **not** do what you expect for Strings
  - Compares that the objects are the same (like Python's `is`)

`StringComparion.java`

14

Control Flow: `else if`

- Python Gotcha: in Python, was `elif`

```
if( x % 2 == 0 ) {
   System.out.println("Value is even.");
}
else if ( x % 3 == 0 ) {
   System.out.println("Value is divisible by 3.");
}
else {
   System.out.println("Value isn't divisible by 2 or 3.");
}
```

What output do we get if x is 9, 13, or 6?

15

Apple's goto fail in SSL

(C code but Java is similar)

```
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

https://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/

16

Apple's goto fail in SSL

```
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;  /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

Lesson: always use braces to mark the body of an if statement, even if it is just one line

What does this code do?

```
if ( x > 4 );
        System.out.println("x is " + x);
```

What does this code do?

```
if ( x > 4 );
      System.out.println("x is " + x);
```

- ; is a valid statement
- Print statement *always* executes
- Indentation doesn't matter

while loop

- You probably guessed it!

```
while (condition) {
    body
}
```

Control Flow: for Loop Example

```
System.out.println("Counting down…");

for (int count=5; count >= 1; count--) {
    System.out.println(count);
}
System.out.println("Blastoff!");
```

shortcut

- What is the counter variable?
- What is the condition?
- What is the output?
- How written in Python?

Can't print out count with Blastoff. Why not?

Countdown.java

Control Flow: for Loop

```
System.out.println("Counting down…");

for (int count=5; count >= 1; count--) {
    System.out.println(count);
}
System.out.println("Blastoff!");
```

```
for ( <init value>; <condition>; <lastiniteration> ) {

}
```

Initialize counter variable; not necessarily declared here

Body should be repeated until this condition isn't true

Executed at end of each iteration of the loop body

Control Flow: for Loop Order

1 2 5… 4

```
for ( <init value>; <condition>; <lastiniteration> ) {

      Body;
      Body;   3
      Body;

}
```

Translated to a while loop

```
System.out.println("Counting down…");

for (int count=5; count >= 1; count--) {
    System.out.println(count);
}
System.out.println("Blastoff!");
```

```
System.out.println("Counting down…");
int count=5;
while (count >= 1) {
    System.out.println(count);
    count--;
}
System.out.println("Blastoff!");
```

Control Flow: for Loop Compare

```
System.out.println("Counting down…");

for (int count=5; count >= 1; count--) {
    System.out.println(count);
}
System.out.println("Blastoff!");
```

In Python:

```
print("Counting down…");

for count in range(5, 0, -1):
    print(count);

print("Blastoff!")
```

More Examples

```
int count;
for (count=5; count >= 1; count--) {
    System.out.println(count);
}
System.out.println("Blastoff!" + count);
```

Counter variable declared before for loop

Empty initialization. Not recommended; set your counter variable!

```
int count=5;
for (; count >= 1; count--) {
    System.out.println(count);
}
System.out.println("Blastoff!" + count);
```

ARRAYS

Python Lists → Java Arrays

- A Java **array** is like a ***fixed-length*** list
- To ***declare*** an array:
  - `DataType[] myArray;`
- Example:

```
int[] arrayOfInts;
```

  - Declaration only makes a variable named `arrayOfInts`
  - Does not initialize array or allocate memory for the elements
  - To declare *and* ***allocate memory*** for array of integers:

```
int[] arrayOfInts = new int[100];
```

`new` keyword:
allocate memory to a new object

Array Initialization

- Initialize an array at its declaration:
  - `int[] fibNums = {1, 1, 2, 3, 5, 8, 13};`

Value	1	1	2	3	5	8	13
Position/index	0	1	2	3	4	5	6

  - Note that we do not use the `new` keyword when allocating and initializing an array in this manner
  - `fibNums` has length 7

Array Access

- Access a value in an array as in Python:
  - `fibNums[0] = 0`
  - …
  - `fibNums[x] = fibNums[x-1] + fibNums[x-2]`

- Unlike in Python, **cannot** use negative numbers to index arrays

Array Length

- All array variables have a ***field*** called `length`
  - Note: no parentheses because *not* a method because arrays are *not* objects

```
int[] array = new int[10];
for (int i = 0; i < array.length; i++) {
      array[i] = i * 2;
}

for (int i = array.length-1; i >= 0; i--) {
      System.out.println(array[i]);
}
```

`ArrayLength.java`

Overstepping Array Length

- Java safeguards against overstepping length of array
  - Runtime Exception: “Array index out of bounds”
  - More on exceptions later…
- Example:

```
int[] array = new int[100];
```

  - Attempts to access or write to index < 0 or index >= array.length (100) will generate exception

Arrays

- Assigning one array variable to another → both variables refer to the same array
  - Similar to Python
- Draw picture of below code:

```
int [] fibNums = {1, 1, 2, 3, 5, 8, 13};
int [] otherFibNums;

otherFibNums = fibNums;
otherFibNums[2] = 99;

System.out.println(otherFibNums[2]);
System.out.println(fibNums[2]);
```

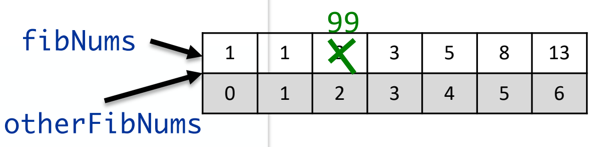
Arrays

- Assigning one array variable to another → both variables refer to the same array
  - Similar to Python
- Draw picture of below code:

```
int [] fibNums = {1, 1, 2, 3, 5, 8, 13};
int [] otherFibNums;

otherFibNums = fibNums;
otherFibNums[2] = 99;

System.out.println(otherFibNums[2]);
System.out.println(fibNums[2]);
```



Displays:
99
99

Command-Line Arguments

- We’ve seen use of an array (of Strings) since we started programming in Java

Contains the command-line arguments

```
public static void main(String[] args) {
   if( args.length < 1 ) {
       System.out.println("Error: invalid number of arguments");
       System.out.println("Usage: java MyProgram <filename>");
       System.exit(1);
   }
}
```

Example Use:
`java MyProgram filename`

35

Command-Line Arguments

- Similar to Python’s `sys` module

```
# Make sure there are sufficient arguments.
if len(sys.argv) < 2:
    print "Error: invalid number of command-line arguments"
    print "Usage: python", sys.argv[0], "<filename>"
    sys.exit(1)
```

Contains the command-line arguments

```
public static void main(String[] args) {
    if( args.length < 1 ) {
         System.out.println("Error: invalid number of arguments");
         System.out.println("Usage: java MyProgram <filename>");
         System.exit(1);
    }
}
```

Example Use:
`java MyProgram filename`

36

Command-Line Arguments

- In Python, `sys.argv[0]` represented the name of program
- Not same in Java
  - Command-line arguments do not include the classname

```
# Make sure there are sufficient arguments.
if len(sys.argv) < 2:
    print "Error: invalid number of command-line arguments"
    print "Usage: python", sys.argv[0], "<filename>"
    sys.exit(1)
```

Have to specify program name in Java, e.g.,

```
System.out.println("Usage: java MyProgram <filename>");
```

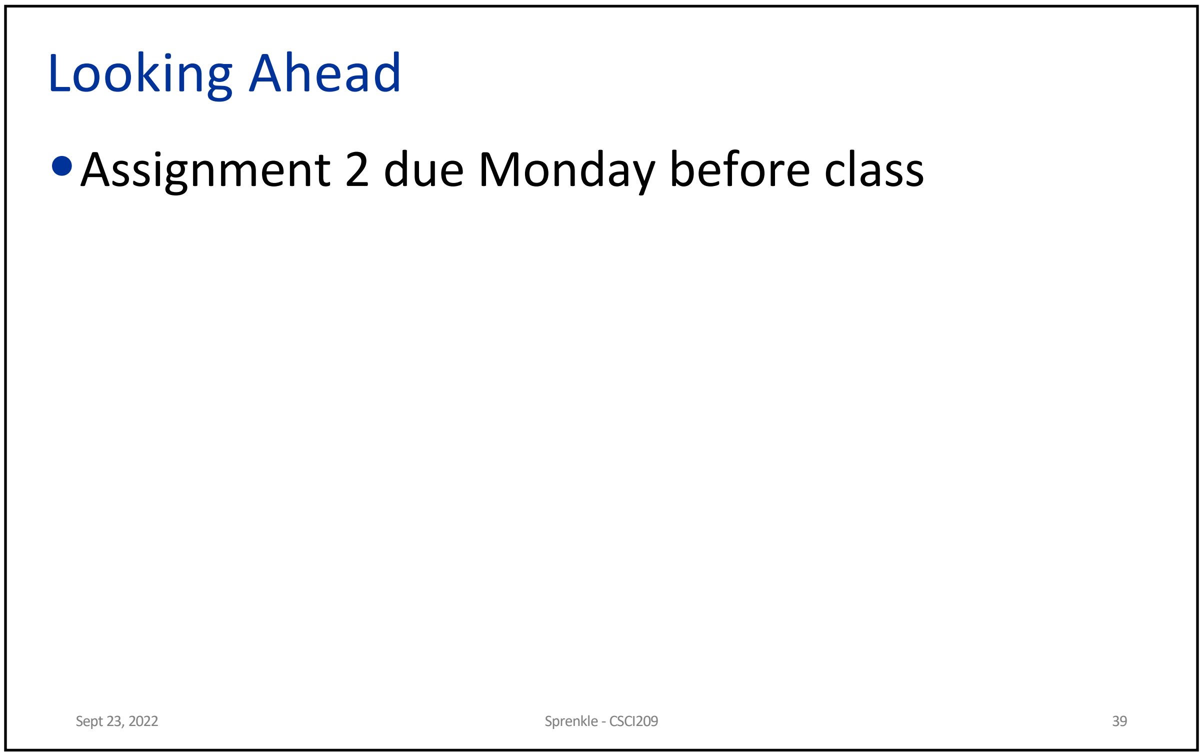
`java.util.Arrays`

Not quite ready for this yet, but leaving for future reference

- `Arrays` is a class in `java.util`
- Methods for sorting, searching, `deepEquals`, fill arrays
- To use class, need `import` statement
  - Goes at top of program, before class definition

```
import java.util.Arrays;
```

ArraysExample.java



Looking Ahead

- Assignment 2 due Monday before class

39