Objectives

- Continuing Java Fundamentals
  - User Input
  - Control Structures
  - Arrays
  - Command-line Arguments

1

Review

- What are some of the primitive data types of Java?
- What is the syntax for declaring a variable in Java?
- What is the keyword for a constant value?
- What are some examples of Java classes?
- How do you call a method?
- How do we know what methods are available to call on a specific Java class?
- Object-oriented programming review: What is a constructor?

2

Assign 1

- Problems?
- Tips or tricks for others?
  - Read: what mistakes will you vow never to do again but probably will?

3

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

4

Getting user input

JAVA.UTIL.SCANNER

5

java.util.Scanner

- Create a Scanner object by calling the ***constructor***
  - new keyword means you're allocating memory for an object

```
Scanner sc = new Scanner(System.in);
```

What is this?

- Need to ***import*** the class because it's not part of java.lang package
  - Imports go at the top of the program, before class definition

```
import java.util.Scanner;
```


6

Scanner

- Makes reading/parsing input easier
- Breaks its input into tokens using a delimiter pattern, which matches whitespace

What is a "delimiter pattern"?
What is "whitespace"?

- Converts resulting tokens into values of different types using `nextXXX()`
- Can change token delimiter from default of whitespace
- Assumes numbers are input as decimal
  - Can specify a different radix

`java.util.Scanner`

- Many constructors
  - Read from file, input stream, string …
- Many methods
  - `nextXXXX` (int, long, line)
  - Skipping patterns, matching patterns, etc.
- Close the Scanner when you're done with it

Using Scanners

- Use *nextXXX()* to read from it…

```
long tempLong;

// create the scanner for the console
Scanner sc = new Scanner(System.in);

// read in an integer and a String
int i = sc.nextInt();
String restOfLine = sc.nextLine();

// read in a bunch of long integers
while (sc.hasNextLong()) {
        tempLong = sc.nextLong();
}

sc.close();
```


Using Scanner

```
public static void main(String[] args) {

  // open the Scanner on the console input, System.in
  Scanner scan = new Scanner(System.in);

  scan.useDelimiter("\n"); // breaks up by lines, useful for
              // console I/O

  System.out.print("Please enter the width of a rectangle: ");
  int width = scan.nextInt();

  System.out.print("Please enter the height of a rectangle: ");
  int length = scan.nextInt();
  scan.close();

  System.out.println("The area of your square is " + length * width + ".");
}
```

ConsoleUsingScannerDemo.java

Effective Java: Code Inefficiency

Why didn't we talk about *constructing* a String?

- I said to do this:

```
String s = "text";
```

- Instead of this

```
String s = new String("text"); // DON'T DO THIS
```


Effective Java: Code Inefficiency

Why didn't we talk about *constructing* a String?

- I said to do this:

```
String s = "text";
```

- Instead of this

```
String s = new String("text"); // DON'T DO THIS
```

Creates two strings

StringBuilders vs Strings

- Strings are read-only or *immutable*
  - Same as Python
- More efficient to use StringBuilder to manipulate a String
- Instead of creating a new String using
  - `String str = prevStr + " more!";`
- Use

new keyword:
allocate memory to a new object

```
StringBuilder str = new StringBuilder( prevStr );
str.append(" more!");
```

- Many StringBuilder methods
  - toString() to get the resultant string back

CONTROL STRUCTURES

Review

- What is the syntax of a *conditional statement* in Python?

Control Flow: Conditional Statements

- `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
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***

Logical Operators

Operation	Python	Java
AND		&&
OR		||
NOT		!

In Python, these are …?

Logical Operators

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

Scoping Issues: Python Gotcha

- 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?

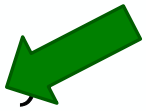
Not Fixed

```
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

Fixed

- 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");
```


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, and 6?

Apple's goto fail in SSL

(actually 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/

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

Review

- How do you write a `for` loop in Python for counting?

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?

ARRAYS

31

Python Lists → Java Arrays

- A Java **array** is like a ***fixed-length*** list
- To declare an array of integers:

```
int[] arrayOfInts;
```

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

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

new keyword:
allocate memory to a new object

32

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

```
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]);
}
```


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

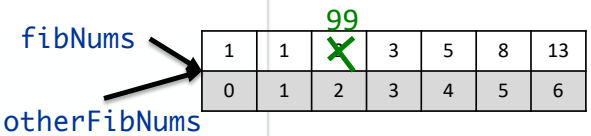
- Assigning one array variable to another ➔both variables refer to the same array
  - Similar to Python
- Draw picture of below code:

```
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

java.util.Arrays

- 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

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

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>");
```


TODO

- Assignment 2: Debugging
  - Due Sunday 11:59 p.m.
- Textbook: through Loops and Iteration