

Objectives

- Control structures
 - Conditionals
 - Loops
 - Switches
- Arrays
- Command-line arguments

Review: Java's Library

- What is an API?
- What are some examples of Java classes?
- What is the syntax to create an object?
- What is the syntax to 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?

StringBuilders and 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

```
StringBuilder str = new StringBuilder( prevStr );  
str.append(" more!");
```
- Many `StringBuilder` methods
 - `toString()` to get the resultant string back

Effective Java: Code Inefficiency

- The example code was

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

instead of this

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

Why didn't the example ***construct*** a String?

Effective Java: Code Inefficiency

- The example code was

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

instead of this

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

Why didn't the example ***construct*** a String?

Because that would create *two* strings!

CONTROL STRUCTURES

Conditional Statements Syntax: **if**

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

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

```
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 { }

Logical Operators

Operation	Python	Java
AND		&&
OR		
NOT		!

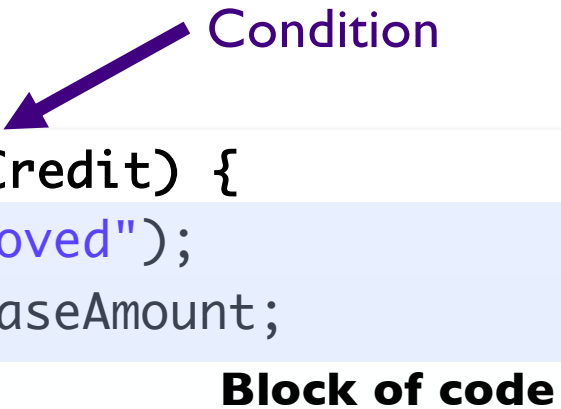
In Python, these are ...?

Logical Operators

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

Control Flow: Conditional Statements

- **if** statement



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

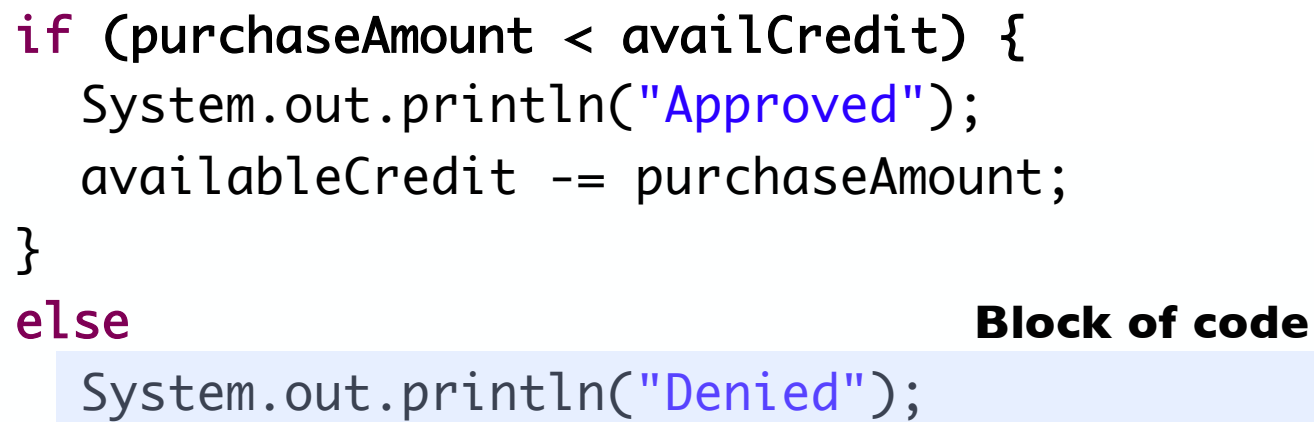
Condition

Block of code

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

Control Flow: Conditional Statements

- **if** statement



The diagram shows an if-else statement with two annotations. A purple arrow points from the word "Condition" to the expression `(purchaseAmount < availCredit)` in the if statement. A light blue rectangular highlight is placed under the code block `System.out.println("Denied");` in the else statement, with the text "Block of code" positioned to its right.

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

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?

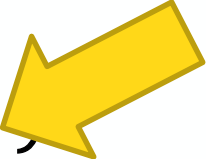
Not Fixed

```
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 declaration of `approved` outside of the `if` statement *and* initialize

```
boolean approved = false;  
if (purchaseAmount < availableCredit) {  
    availableCredit -= purchaseAmount;  
    approved = true;  
}  
  
if( ! approved )  
    System.out.println("Denied");
```

Alternative Fix

- Move declaration of **approved** outside of the **if** statement

```
boolean approved;  
if (purchaseAmount < availableCredit) {  
    availableCredit -= purchaseAmount;  
    approved = true;  
} else  
    approved = false;  
  
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 when x is 9, 13, or 6?

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;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

<https://cacm.acm.org/blogcacm/those-who-say-code-does-not-matter/>

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

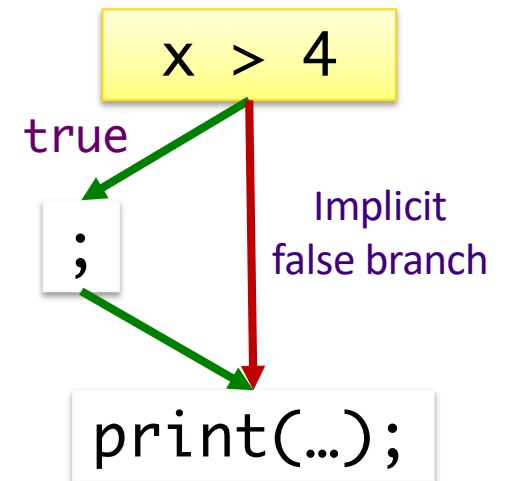
What does this code do?

```
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

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

(You probably guessed it!)

Control Flow: **while** Loops

● **while** loop

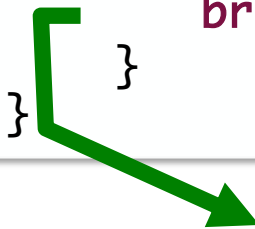
- Condition must be enclosed in parentheses
- Body of loop must be enclosed in **{}** if multiple statements

```
int counter = 0;
while (counter < 5) {
    System.out.println(counter);
    counter++; ← shortcut
}
System.out.println("Done: " + counter);
```

Changing control flow: **break**

- Exits the current loop

```
while ( <condition> ) {  
    ...  
    if( <something> ) { // now we're done!  
        break;  
    }  
}
```



Control Flow: for Loop

```
for ( <init value>; <condition>; <lastiteration> ) {  
    Body;  
    Body;  
    Body;  
}
```

<init value> : ~start

<condition> : ~stop

<lastiteration> : ~step

For Python's range(start, stop, step)

But, more customizable/flexible

Control Flow: **for** Loop Order

The diagram illustrates the execution order of a **for** loop. It features a code snippet with five numbered steps in green circles: 1 (above the opening parenthesis), 2 (above the semicolon), 5... (above the condition), 4 (above the closing parenthesis), and 3 (to the right of the loop body). Step 5 is highlighted with a green oval. To the right of the code, text explains that if the condition is true at Step 5, steps 3 and 4 are repeated, and the condition is checked again at Step 5.

```
for ( <init value>; <condition>; <lastiteration> ) {  
    Body;  
    Body;  
    Body;  
}
```

If condition is true at Step 5,
Repeat steps 3, 4, check 5, ...

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!");
```

- What is the output?
- How written as a while loop?
- How written in Python?
- You can't print out count with Blastoff. Why not?

for vs 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;      Counter variable declared before for loop
for (count=5; count >= 1; count--) {
    System.out.println(count);
}
System.out.println("Blastoff!" + count);
```

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

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];`
 - Assigned default values for data type
 - But don't rely on defaults
 - `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] = 1;`
 - ...
 - `fibNums[x] = fibNums[x-1] + fibNums[x-2];`
- Unlike in Python, **cannot** use negative numbers to index arrays
- Overstepping an array
 - Attempts to access or write to index < 0 or index $\geq \text{array.length}$ will generate array index out of bounds exception

Array Length

- All array variables have a *property* called **length**
 - Note: no parentheses because it's *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]);  
}
```

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 myFile.txt
```

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)
```

```
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);  
    }  
}
```

Contains the command-line arguments



Example Use:

```
java MyProgram myFile.txt
```

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>");
```

String Comparison: equals

- **boolean** equals(Object anObject)

- Compares this string to the specified object

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

- **test** is false because the Strings contain different values

- **==** is like Python's **is**

- Compares that the objects are the same (like Python's **is**)

Control Flow: for each Loop

- Sun called “enhanced for” loop
- Iterate over all elements in an array (or Collection)
 - Similar to Python’s **for** loop

```
int[] a;  
int result = 0;  
.  
.  
.  
for (int i : a) {  
    result += i;  
}
```

for each int element **i** *in* the array **a**,
the loop body is executed

<https://docs.oracle.com/javase/8/docs/technotes/guides/language/foreach.html>

Arrays

- Assigning one array variable to another → both variables refer to the same array
 - Similar to Python

Arrays

- Assigning one array variable to another → both variables refer to the same array

➤ Similar to Python

- Draw picture of 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

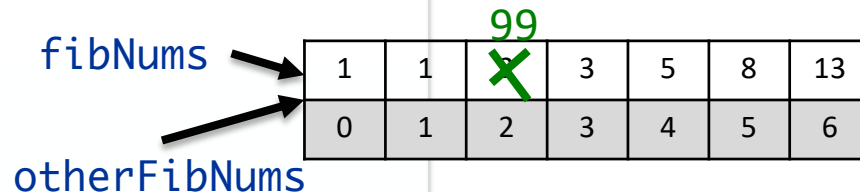
➤ Similar to Python

- Draw picture of 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** package
 - Methods for sorting, searching, `deepEquals`, filling arrays
 - To use class, need **import** statement
 - Goes at top of program, before class definition
- ```
import java.util.Arrays;
```
- Call methods on the *class* because the methods are static
    - More on this class and the distinction later

# Control Flow: **switch** statement

- Similar to an **if/else if** statement
- Works with variables with datatypes **byte**, **short**, **char**, **int**, and **String**

```
int x = 3;
switch(x) {
 case 1:
 System.out.println("It's a 1.");
 break;
 case 2:
 System.out.println("It's a 2.");
 break;
 default:
 System.out.println("Not a 1 or 2.");
}
```

→ Exits the block

# Control Flow: **switch** statement

```
switch(grade) {
 case 'a':
 case 'A':
 System.out.println("Congrats!");
 break;
 case 'b':
 case 'B':
 System.out.println("Not too shabby!");
 break;
 ... // Handle c, d, and f ...
 default:
 System.out.println("Error: not a grade");
}
```

As of version 3.10, Python has match/case statement:  
<https://docs.python.org/3/whatsnew/3.10.html>

Grades.java

# Control Flow: **switch** statement

```
switch(grade) {
 case 'a':
 case 'A':
 System.out.println("Congrats!");
 break;
 case 'b':
 case 'B':
 System.out.println("Not too shabby!");
 break;
 ... // Handle c, d, and f ...
 default:
 System.out.println("Error: not a grade");
}
```

When grade is 'a':

Congrats!

When grade is 'B':

Not too shabby!

Grades.java

# Control Flow: **switch** statement

(Not covered in class)

```
switch(grade) {
 case 'a':
 case 'A':
 System.out.println("Congrats!");
 case 'b':
 case 'B':
 System.out.println("Not too shabby!");
 break;
 ... // Handle c, d, and f ...
 default:
 System.out.println("Error: not a grade");
}
```

Grades.java

# Control Flow: **switch** statement

(Not covered in class)

```
switch(grade) {
 case 'a':
 case 'A':
 System.out.println("Congrats!");
 case 'b':
 case 'B':
 System.out.println("Not too shabby!");
 break;
 ... // Handle c, d, and f ...
 default:
 System.out.println("Error: not a grade");
}
```

When grade is 'a':

Congrats!  
Not too shabby!

When grade is 'B':

Not too shabby!

Grades.java

# Looking Ahead

- Assignment 1 due Wednesday before class