Objectives

- More Java fundamentals
  - `java.lang` classes: `Math` and `String` class
  - Control Structures
  - Arrays

Review: Python Transition **Warning**

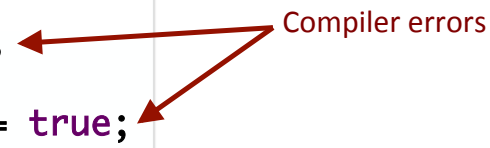
You cannot **redeclare** a variable name in the same scope

- OK:

```
int x = 3;
x = -3;
```

- Not OK:

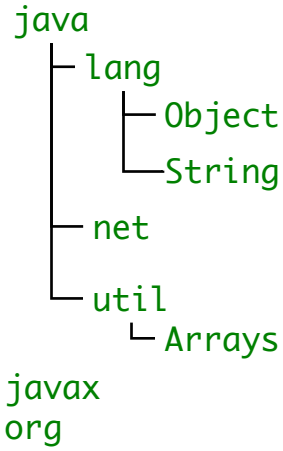
```
int x = 3;
int x = -3;
…
boolean x = true;
```



INTRO TO JAVA LIBRARIES

Java Libraries

- Organized into a hierarchy of ***packages***



Fully qualified name: `java.lang.String`

`java.lang.*` classes included by default in all Java programs

Many, many more classes and packages

Java API Documentation

- **API**: Application Programming Interface
  - What the class can do for YOU!
- Complete documentation of every class included with the JDK
  - Every method and variable contained in class

http://docs.oracle.com/javase/8/docs/api

- Bookmark it!
  - Too many classes, methods to remember them all
  - Refer to it often

`java.lang.Math` class

- Similar to Python's `math` module
- **Included by default** in every Java program
- Contains useful mathematical functions (methods) and constants (fields):

- Look at `java.lang.Math` API online
  - http://docs.oracle.com/javase/8/docs/api/
  - Note how API is specified

java.lang.Math class

- Example Uses:

method

constant

```
double y = Math.pow(x, a);
double z = Math.sin(y);
double d = Math.exp(4.59) * Math.PI;
```

Use Classname.methodname() to call Math's methods because they're static

static: for the class

MathExample.java

java.lang.String class

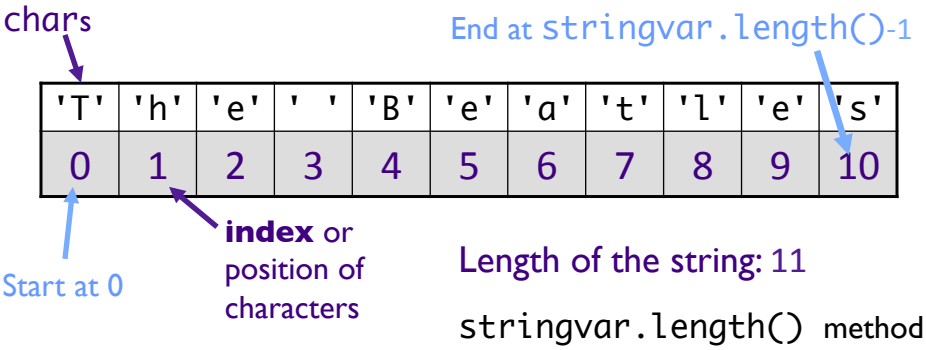
- Similar functionality to Python but different ways to use
- Strings are represented by **double** quotes: ""
  - Single quotes represent **chars** only
- Examples:

```
String emptyString = "";
String niceGreeting = "Hello there.";
String badGreeting = "What do you want?";
```

Strings

- A `char` at each position of `String`

```
stringvar = "The Beatles";
```



Length of the string: 11

`stringvar.length()` method

Use `charAt` method to access chars

String method: charAt

- A `String` is a collection of `chars`

```
String testString1 = "Demonstrate Strings";

char character1;
char character2 = testString1.charAt(3);
character1 = testString1.charAt(2);
```

String methods: substring

- Like *slicing* in Python
- `String substring(int beginIndex)`
  - Returns a new String that is a substring of this string, from `beginIndex` to end of this string
- `String substring(int beginIndex, int endIndex)`
  - Returns a new String that is a substring of this string, from `beginIndex` to `endIndex-1`

```
String language = "Java!";
String subStr = language.substring(1);
String subStr2 = language.substring(2, 4);
```

Python Gotcha: Can't use negative numbers for indices as in Python

String Concatenation

- Use `+` operator to concatenate Strings

```
String niceGreeting = "Hello";
String firstName = "Clark";
String lastName = "Kent";
String blankSpace = " ";

String greeting = niceGreeting + "," +
      blankSpace + firstName +
      blankSpace + lastName;

System.out.println(greeting);
```

Prints "Hello, Clark Kent"

String Concatenation

- If a `String` is concatenated with something that is not a `String`, the other variable is converted to a `String` automatically.

```
int totalPoints = 110;
int earnedPoints = 87;
float testScore = (float) earnedPoints/totalPoints;

System.out.println("Your score is " + testScore);
```

Converted to a `String`

`StringBuilders` vs `Strings`

- `Strings` are "read-only" or *immutable*
  - Same as Python
- 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

Effective Java: Code Inefficiency

- Avoid creating unnecessary objects:

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

- Do this instead:

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

Why?

String Comparison: `equals`

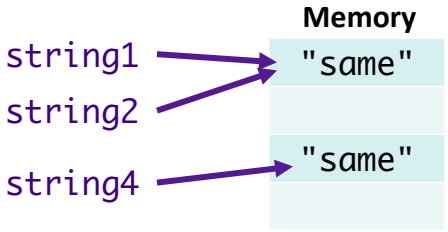
- `boolean equals(Object anObject)`
  - Compares this string to the specified object

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

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

Python Gotcha: String Comparisons

- `string1 == string4` will **not** yield the same result as `string1.equals(string4)`
  - == tests if the *objects* are the same
    - **not** if the *contents* of the objects are the same
  - Similar to `is` operator in Python



`string1 != string4`
BUT
`string1.equals(string4)`

String method: equalsIgnoreCase

- Does what it's named!

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

- `test` is true!

String methods: and many more!

- `boolean endsWith(String suffix)`
- `boolean startsWith(String prefix)`
- `int length()`
- `String toLowerCase()`
- `String trim()` : remove trailing and leading white space
- …
- See `java.lang.String` API for all

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`
  - Body is enclosed by {} if multiple statements

```
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
  - Has an associated scope

Scoping Issues: Python Gotcha

- Everything between { } is a block of code
  - 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?

Fixed

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

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

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


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: `else if`

- 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?

Control Flow: switch statement

- Like a big `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.");
}
```

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");
}
```

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++;
}
System.out.println("Done: " + counter);
```

shortcut

Changing control flow: `break`

- Exits the current loop

```
while ( <readingdata> ) {
   …
   if( <somethingbad> ) { // shouldn't happen
     break;
   }
}
```

Review

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

Control Flow: `for` Loop

- Very different syntax from Python
- Syntax:

```
for (<init>; <condition>; <execution_expr>)
```

Loop's counter variable, Usually used in condition

Executed at end of each iteration. Typically increments or decrements counter

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?

ARRAYS

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

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]`
  - `fibNums[x] = fibNums[x-1] + fibNums[x-2]`
- Unlike in Python, can**not** use negative numbers to index items

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]);
```

fibNums[2] and otherFibNums[2] are both equal to 99

Array Copying

- Copy an array (element-by-element) using the `arraycopy` method in the `System` class

```
System.arraycopy(from, fromIndex, to, toIndex, count);
```

- For example:

```
int [] fibNums = {1, 1, 2, 3, 5, 8, 13};
int [] otherNums = new int[fibNums.length];
System.arraycopy(fibNums,0,otherNums,0,fibNums.length);
otherNums[2] = 99;
System.out.println(otherNums[2]);
System.out.println(fibNums[2]);
```

`fibNums[2] = 2,`
`otherNums[2]= 99`

Control Flow: `foreach` Loop

- Introduced in Java 5
  - Sun calls "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;
}
```

"in"

http://docs.oracle.com/javase/1.5.0/docs/guide/language/foreach.html

for each int element `i` *in* the array `a` , the loop body is visited once for each element of `a`.

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`

Summary: Python to Java Gotchas

- Every variable needs to be declared before it is used
- Every variable needs a statically-declared data type
- Scope of variables
- Syntax
  - Semicolons at the end of **statements**
  - Braces around blocks of code
  - Keywords