# Objectives

- Arithmetic operators
- Using the Java Library
  - `java.lang` classes: `String` class
  - Getting user input with `java.util.Scanner`
  - Constructing objects
  - Using the Java API
  - Importing classes



1

# Review

- How do you compile and run Java programs?
- How do you display output in Java?
- What are the modifiers for the `main` method?
  - What are the parameter(s) to `main`?
  - How do you *call* the `main` method?
- What are some of the primitive data types of Java?
- What is the syntax for declaring a variable in Java?
  - What is the difference between *declaring* a variable and *defining* a variable?
- How does Java compare to Python (so far)?

You can and *should* review previous slides if you don't remember answers



2

# Review: Example Java Program

```
/**
 * Our first Java class: displays Hello!
 * @author Sara Sprenkle
 */
public class Hello {
   public static void main(String[] args) {
      //print a message
      System.out.println("Hello!");
   }
}
```

`main` method is automatically called when you run

```
java Hello
```





# Python Transition **Warning**

You can**not** redeclare a variable name in the same scope

- OK:

```
int x = 3;
x = -3;
… // more code
x = 7;
```

Declaration

Definition

- Not OK:

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

boolean x = true;
```

Compiler errors

## More Data Type-Related Information

- Result of integer division is an `int`
  - Same as Python **2**, **not** Python 3
  - Example: `4/3 = ??`
- Casting
  - Similar to Python for primitive types
  - Example: `4/(double) 3`

TestScore.java





## Floats in Java

- Decimal literals are considered *doubles*
- This code won't compile:

```
float f = 3.14;
```

Compiler reads 3.14 as a *double*

- Compiler error message:

```
Float.java:15: error: incompatible types: possible lossy
conversion from double to float
       float f = 3.14;
                 ^
1 error
```

- To fix code, add an `f` to specification of number or declare as `double`

## Reflection: Assign 0

- How did it go?
- How long did it take?
- What tips/tricks did you learn/would you recommend?
- Why is part of the assignment debugging given code?
  - Why is that approach beneficial to you?



7

## Reflection: Assignment 0

- I am "overcommenting" the Java programs
  - I explain things that you don't need to explain in your assignments (but you should certainly understand!)
- Review: Expectations for your assignments
  - Comments
    - High-level comment at top of program
    - Mark authorship with @author at bottom of top comment
    - Explain blocks of code or difficult code
    - Comments that help you
    - Expectations will change as we learn more
  - Name classes and output files as I request



8

## Arithmetic, Relational Operators

- Java has most of the same operators as Python:
  - Arithmetic operators: `+, -, *, /, %`
    - **No power operator:** `**`
  - Relational operators: `==, !=, <, >, <=, >=`
    - Evaluate to a `boolean` value
  - Increment and decrement
    - `+= x, -= y,` etc.
    - Additional shortcut for += 1, -=1: `++ , --`





## Escape Sequences

Same as Python:

| Meaning | Sequence |
|---|---|
| Newline character (carriage return) | \n |
| Tab | \t |
| Quote | \" |
| Backslash | \\ |

- Combination of characters to represent something else
- Escape character: \
- In Java, you can represent a ' without escaping
- What does the following display?

```
System.out.println("To print a \\, you must use
  \"\\\\\"");
```

EscapeCharacters.java

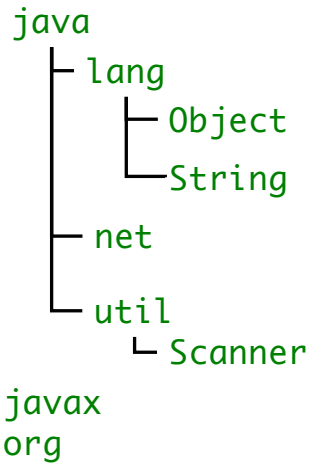
# INTRODUCTION TO JAVA LIBRARIES





## Java Libraries

- Organized into a hierarchy of ***packages***
- Similar to Python's ***packages*** (`__init__.py`)




Fully qualified name: `java.lang.String`

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

Many, many more classes and packages

## java.lang.String class

- Similar functionality to Python but accessed differently
  - Mostly through *methods*!
- **Included by default** in every Java program
- Strings are represented by **double** quotes: ""
  - Single quotes represent **chars** only
- Examples:

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

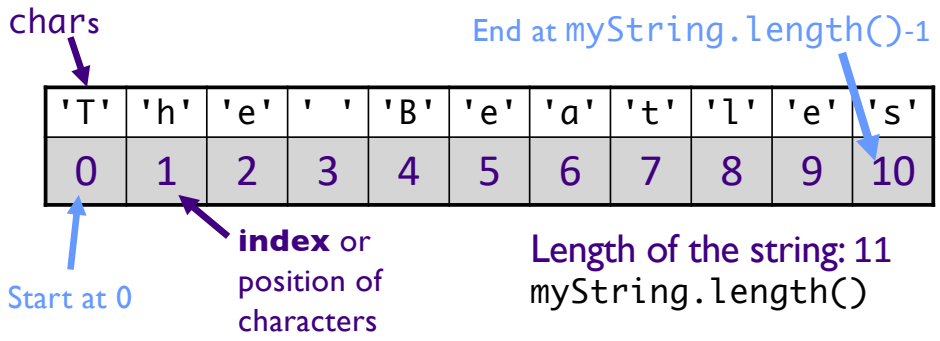




## Strings

- A char at each position of String

```
String myString = "The Beatles";
```




Calling a method is the same syntax as Python: `object.method()`

# Java API Documentation

https://docs.oracle.com/en/java/javase/18/docs/api/index.html

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



15

# Reading JavaDocs: String Overview

Data type of what method returns

| Modifier and Type | Method | Description |
|---|---|---|
| char | charAt(int index) | Returns the char value at the specified index. |
| IntStream | chars() | Returns a stream of int zero-extending the char values from this sequence. |
| | ePointAt(int index) | Returns the character (Unicode code point) at the specified index. |
| int | codePointBefore(int index) | Returns the character (Unicode code point) before the specified index. |
| int | codePointCount(int beginIndex, int endIndex) | Returns the number of Unicode code points in the specified text range of this String. |
| IntStream | codePoints() | Returns a stream of code point values from this sequence. |



16

# Reading JavaDocs: Detail

**charAt**

`public char charAt(int index)`

Returns the `char` value at the specified index. An index ranges from `0` to `length() - 1`. The first `char` value of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

If the `char` value specified by the index is a surrogate, the surrogate value is returned.

**Specified by:**

`charAt` in interface `CharSequence`

**Parameters:**

`index` - the index of the `char` value.

**Returns:**

the `char` value at the specified index of this string. The first `char` value is at index `0`.

**Throws:**

`IndexOutOfBoundsException` - if the `index` argument is negative or not less than the length of this string.





# String method: charAt

- Python equivalent: indexing with [<pos>]

```
String testString1 = "Demonstrate Strings";

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

System.out.println(character1 + " " + character2);
```

Displays “g o”

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

# String methods: substring

- Python equivalent: *slicing*
- `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);
```

subStr is "ava!"
subStr2 is "va"





# 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);
```

Note: statement doesn't end until ;

Displays "Hello, Clark Kent"

## 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)

Equals.java





## String methods: and many more!

- boolean endsWith(String suffix)
- boolean startsWith(String prefix)
- boolean equalsIgnoreCase(String other)
- int length()
- String toLowerCase()
- String trim() : remove trailing and leading white space
- …

See java.lang.String API for all available methods:
https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html

Getting user input

# JAVA.UTIL.SCANNER



23

## Getting User Input

- To get user input, we will use the Scanner class
- 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;
```



24

## Generalizing

- Constructing a new object:

```
DataType object = new DataType(arguments,…);
```

- Importing classes
  - If the class does not belong to `java.lang` package, need to import it





## 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()`
- Assumes numbers are input as decimal
  - Can specify a different radix

# Example Code Using Scanners

```
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();

tempLong = sc.hasNextLong();

sc.close();
```





# java.util.Scanner

Multiple constructors? Different from Python!

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

# 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

# To Do

- Textbook: Read “Java Data Types”, up to but not including List
- Assign 1
  - Part 0: Fix compiler and logic errors from program
  - Part 1: Find the file extension of a filename
  - Due before Friday’s class