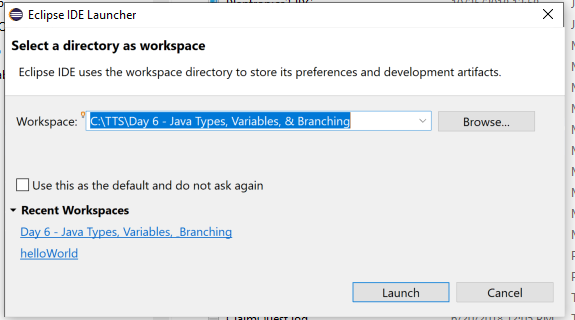
**Day 6 - Java Types, Variables, & Branching Notes**

Git / Git-hub questions?

**Java- Setup and Introduction Slides**

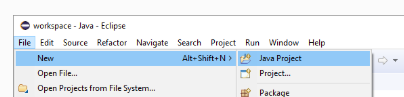
Create C:\TTS Notes\Code Immersion Java\Day 6 - Java Types, Variables, & Branching

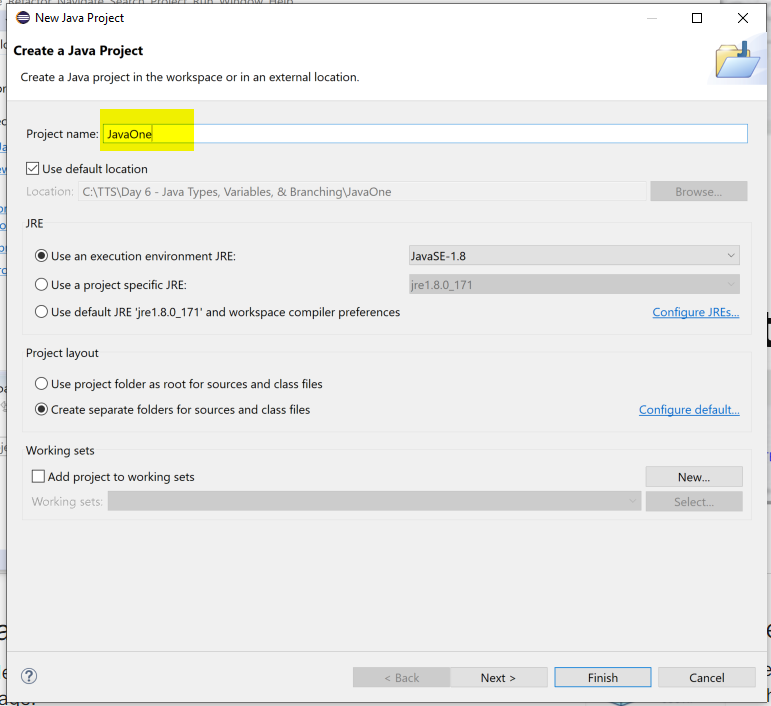
<https://happycoding.io/tutorials/java/eclipse>



Explain “Recent Workspaces” and how to remove

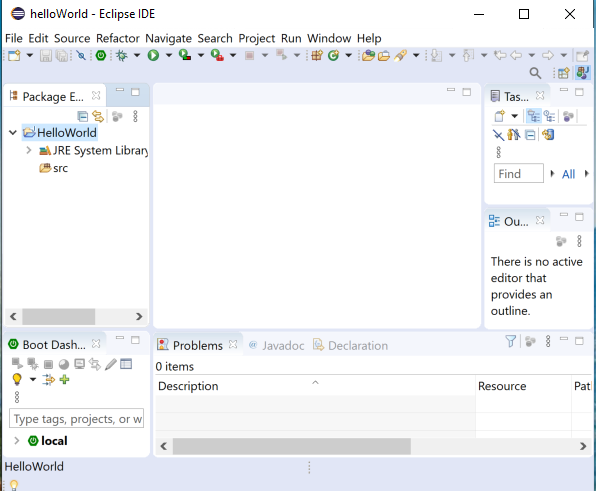
Show Help About





Note the JSE in use.

// This will be project JavaOne, not HelloWorld

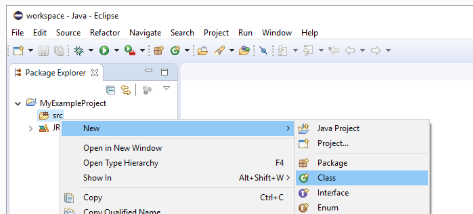


Discuss

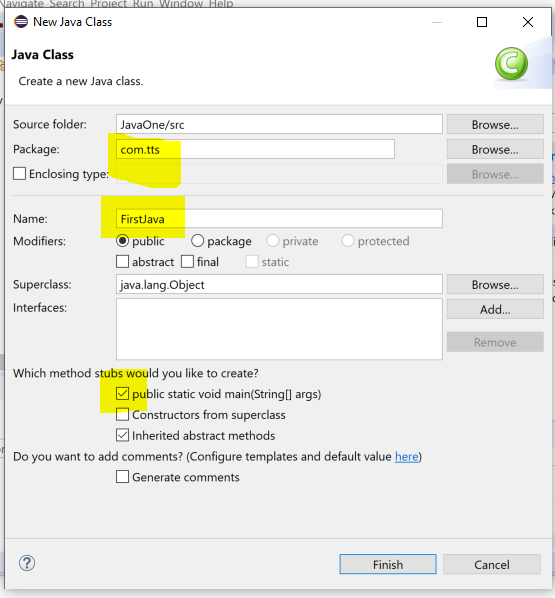
Package explorer

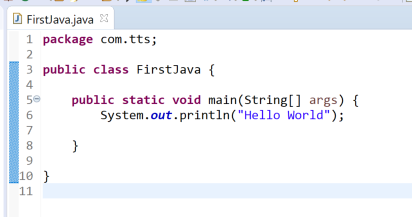
Editor pane

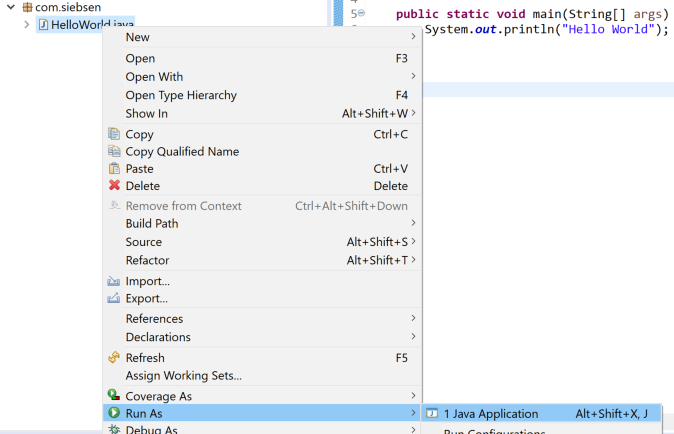
Window -> perspective -> reset perspective



// Call this Main, not FirstJava







**Review and discuss Main app**

Package shows what folder on the PC contains the file. Right Click -> Properties -> Location

Public class HelloWorld name of Java class, usually camel case

Name must match file name on disk

**public** **static** **void** main(String[] args)

public vs protected vs private

static – means it can be called without creating an instance of the class

void – return value

args is string array.

**Java: An Introduction Slides**

Show slides 1-6

**TOPIC: Variables**

Declare simple variable (in main())

String name = "Bill";

System.out.println("Hello " + name);

Add more args, convention is to start lowercase with variable names

String name = "Bill";

int age = 29;

boolean married = true;

double hourlyRate = 60.02;

System.out.println("Name " + name);

System.out.println("Age " + age );

System.out.println("Married " + married);

System.out.println("Rate " + hourlyRate);

Show what happens if you don’t initialize age? Unlike JavaScript

**Challenge (Slide 26)**

Create Madlib don’t use scanner.

String name = “Joe”;

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

String name = "Joe";

String adverb = "smokey";

String noun = "horse";

String noun2 = "leg";

**int** days = 5;

String yourName = "Chuck";

System.***out***.println("Dear " + name + "'");

System.***out***.println("You are " +adverb + " and I want to be your " + noun +

"! I want to go to the " + noun2 + " with you in " + days + " days.");

System.***out***.println("Sincerely, " + yourName);

}

Add an age limit

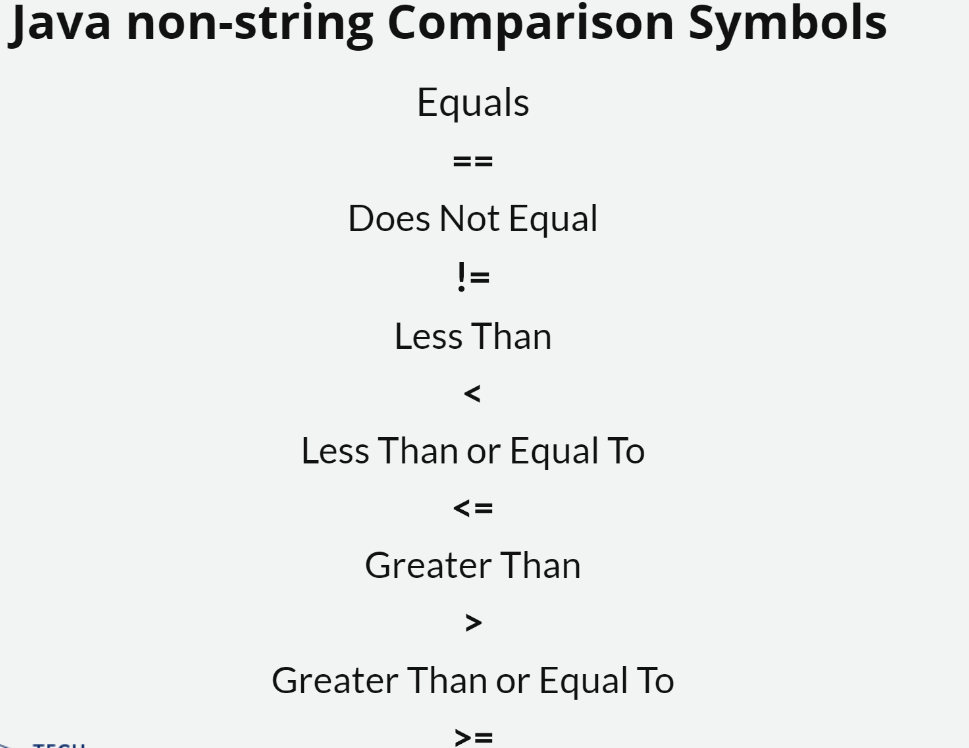
**int** age = 17;

**if** (age < 18) {

System.***out***.println("You are too young to play");

} **else** {

Show Slide 42



// String equals

// These two have the same value

**boolean** test1 = **new** String("test").equals("test");

System.***out***.println(test1);

// ... but they are not the same object

**boolean** test2 = **new** String("test") == **new** String("test"); // --> false

(Show class slide 43)

**CHALLENGE Slide 44**

* Dog Says Cat Says: ask user to enter 'dog' or 'cat', program prints animal's sound
* Guessing Game: user provides a number (between 1 and 10), if the number stored in the program is the same, "Wow!", else, "Nope!"
* Ask the user for their number grade, if the grade is at least 60, tell them they passed! If it's lower than 60, tell them they have to take the class again.

**Finish up first slide deck**

Show else if ()

Show && ||

**Java-Types, Variables, & Branching Slides (Slide deck 2)**

Variable naming conventions

// camel case, used for instance variables or fields of a class.

int currentScoreOfGame;

// Pascal case, used for naming classes

public class FinalScoreOfGame;

// leading underscore sometimes used for short-lived or special purpose names

int \_membersOutOfTown;

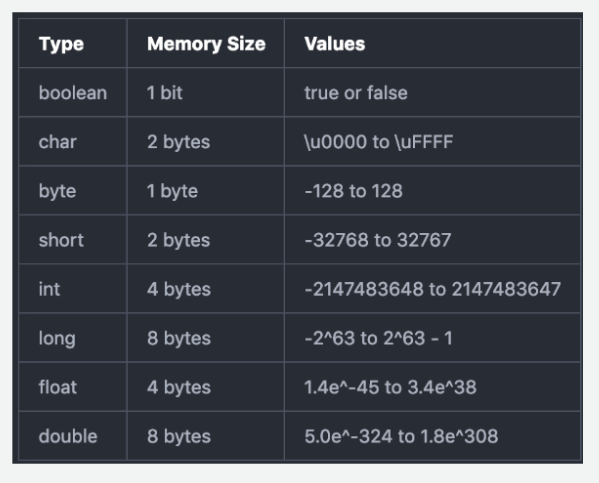
// Not used often in Java, more often in JavaScript to denote a variable tied to an event

int $amountOfBill;

// All Caps with an underscore between words is a common convention for constant values

final String COMPANY = "TTS";

Show slide for primitive datatypes (Slide 3)



Integer types vs. decimal.

Notice there is no String, String is a reference datatype

Show typecasting

**byte** b = (**byte**)12;

**float** f = (**float**)1234.34;

**byte** b2 = (**byte**)550; //byte is max 128 Show truncation System.out.println("Byte " + b2);

Other methods of typecasting

**long** BigLong = 230L; // long (l, L)

**float** SomeFloat = 927.42f; // float (f, F)

**double** someDbl = 392.2d; // double (d, D)

**double** wishfulSalary = 123\_000\_100\_325.0d; // stored/printed as 123000100325.0

**TOPIC: Operators**

Create variables to test

**int** x = 5;

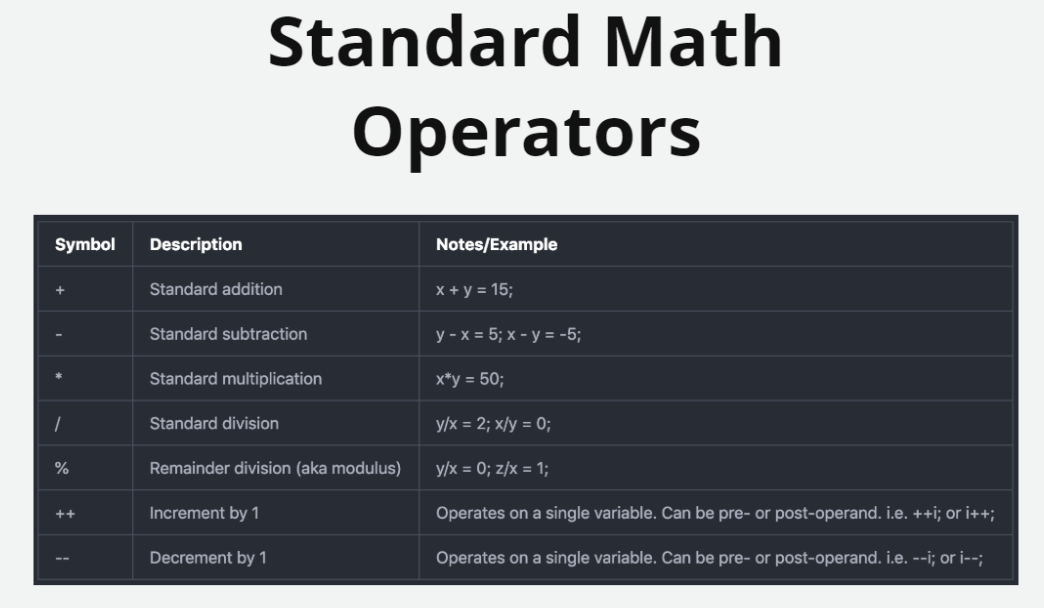
**int** y = 10;

**int** z = 11;

**boolean** fun = **true**;

**boolean** done = **false**;

**(Slide 17)**



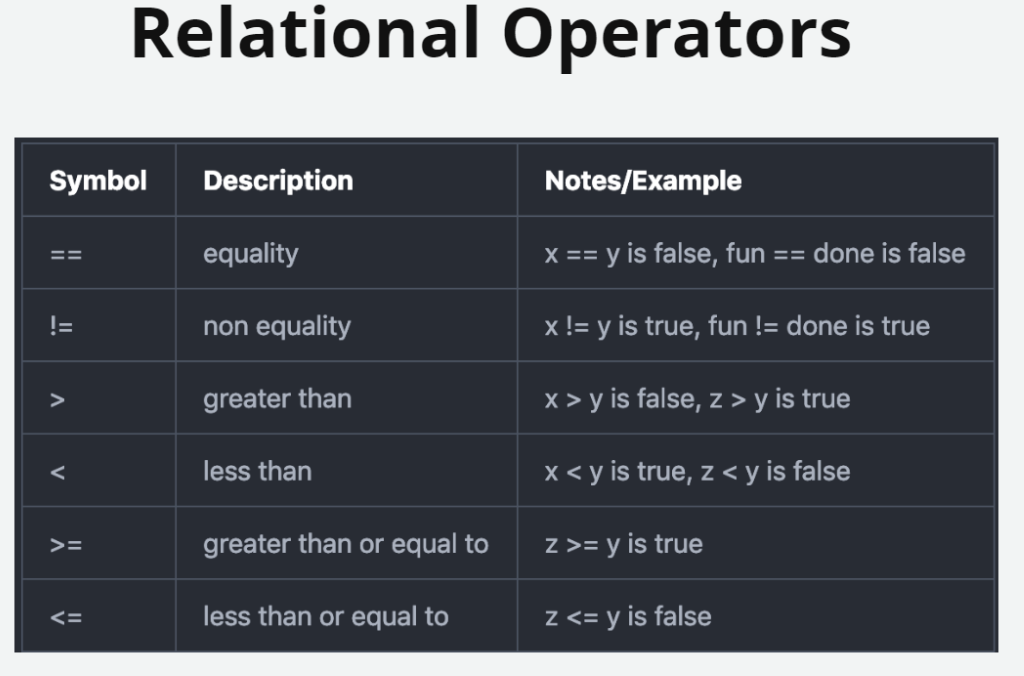
**int** modulus = z%x; // returns 1

**int** after = x++;

**int** before = ++x;

System.***out***.println(after + " " + before);

(Slide 20)



Talk through these with class

**int** x = 5;

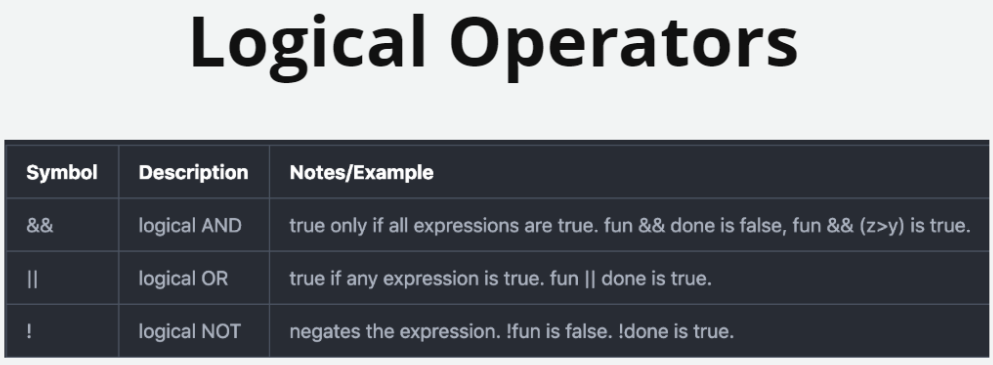
**int** y = 10;

**int** z = 11;

**boolean** fun = **true**;

**boolean** done = **false**;

**(Slide 21)**



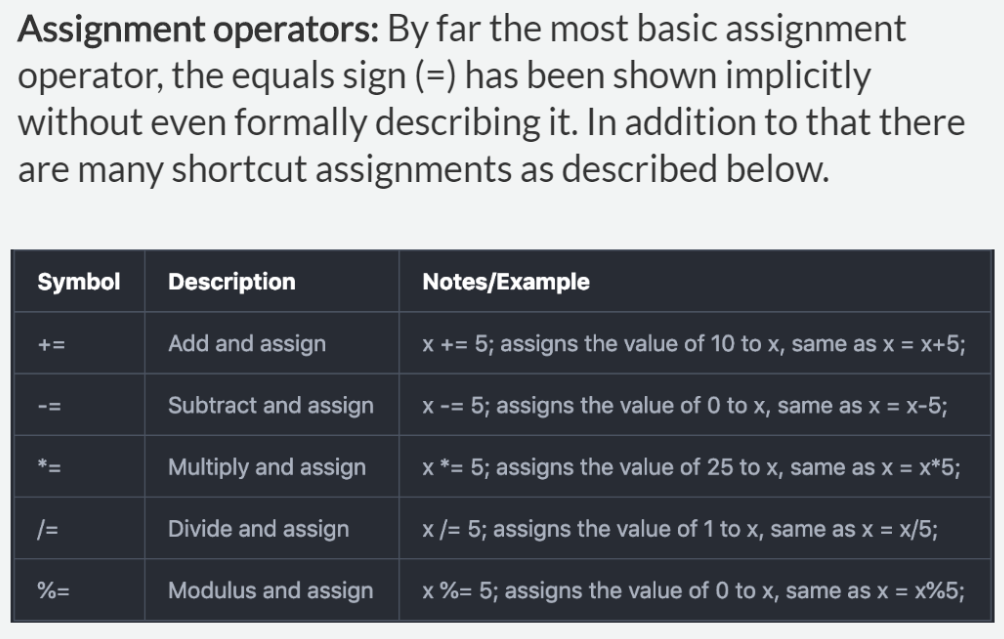
Enter in Main() and discuss

**boolean** andTest = fun && done;

**boolean** orTest = fun || done;

**boolean** notTest = !done;

(Slide 22)



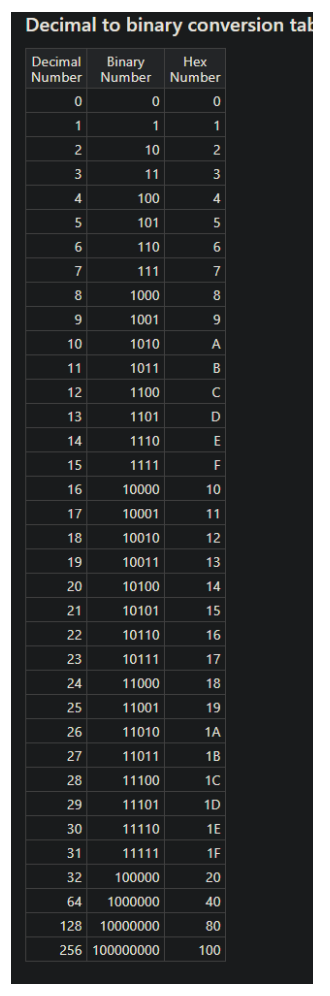
Skipping bitwise programming

Rarely used anymore

Conserves memory but extremely convoluted.

You can look it up if you ever have to deal with it.

Interviewer, “Have you ever done bit wise programming”



**TOPIC: Scope and Modifiers**

**Sample code from slide (Slide 32 )**

**Have students make new ScopeExamples class and replace all but package**

Package com.tts;

public class ScopeExamples

{

// a class-level variable

int aClassVaraible = 0;

public void methodOne()

{

// a method-level variable - accessible anywhere in methodOne()

int outer = 0;

if ( outer == 0 )

{

// Only accessible inside of this if block and any other block inside

int inTheIf = 33;

for( int i=0; i<20; i++ )

{

// both inTheIf and the for-loop varible are accessible in here

System.out.println("Value of inTheIf: " + inTheIf++);

System.out.println("Index: " + i);

}

outer = inTheIf % 6; // should set outer to 5

// This will be a compile error - i is not visible here

if ( i == 20)

{

System.out.println("Value of i: " + i);

}

// the closs varible, the method varible, and inTheIf are all visible here

aClassVariable = outer \* inTheIf

}

}

public void methodTwo()

{

// again, a class variable is access inside a class method

if (aClassVaraible != 0)

{

System.out.println("Method one must have been called!");

}

try

{

// variables declared inside a try block

int noom = 9;

int denom = 0;

int boom = noom/denom;

}

catch(Exception e)

{

System.out.println("Oops! An exception");

// This statement causes a compile error because denom not visible here

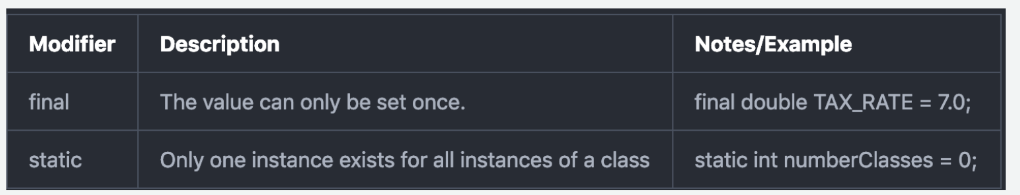
System.out.println("Cannot divide by " + denom);

}

}

}

**TOPIC:Modifiers (Slide 34)**



**public** **class** FirstJava {

**static** **int** *numClasses* = 0;

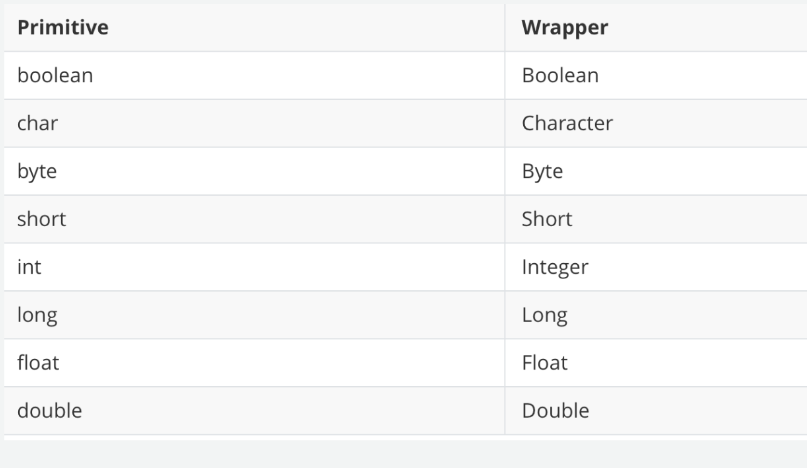
**final** **double** taxRate = 7.0;

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

*numClasses* = 5;

taxRate = 4.0;} // Compile error

**TOPIC: Wrappers (Side 36)**



**double** number = Double.*parseDouble*("1203.44");

**long** aLong = Long.*parseLong*("349392");

int x = Integer.*max*(4,5);

// Autoboxing

**int** x = **new** Integer(4);

Integer y = 4;

Integer smartInt = Integer.*valueOf*(44); // more efficient

**The String Type (Slide 46)**

The Java type String is a class type provided by the Java language. It is not a native type but is often thought of as a fundamental type since it is not user-defined, is provided by the framework itself, and logically is a collection of character values.

String s1 = "Hello";

String s2 = new String("World");

System.***out***.println(s1.contains("el"));

s1.indexOf("e", 1);

s1.substring(3, 4);

s1.replace("e", "a");

String s3 = s1.concat(s2); // not efficient

StringBuffer sb1 = **new** StringBuffer();// StringBuilder is faster but not synchronized

sb1.append("a");

sb1.append("b");

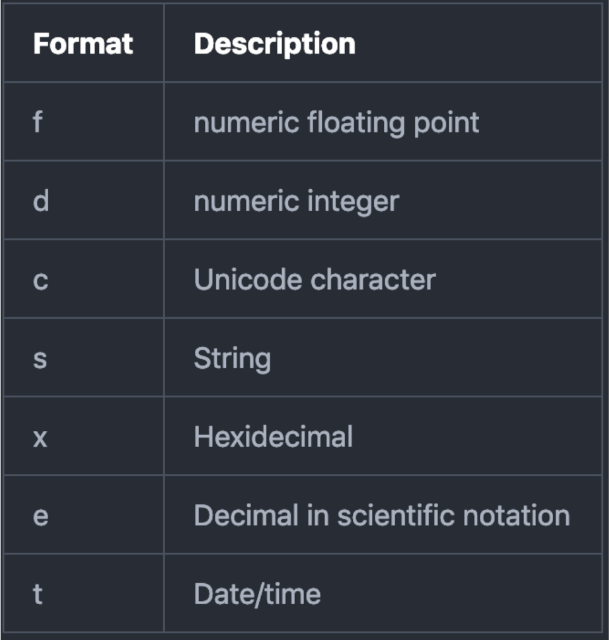
sb1.append("c");

String s4 = sb1.toString();

**TOPIC:String.*format***

String str = String.*format*("%s was found at position %d.", "mars", 3); // show message doc in Eclipse, many params

(Side 55)



**QUESTIONS at end of slide deck (Slide 57)**

* Int plus double > When one operand is an int and the other is a double, Java creates a new temporary value that is the double version of the int operand
* Int times float > any of the operand is a floating point number than other operand will be widened to a floating point
* Skip last 4 bitwise methods

**Java: Types, Variables, & Branching Lab**

Homework:

Review with Students

Show switch statement reference <https://www.w3schools.com/java/java_switch.asp>