GCIS-124
Software Development & Problem Solving

1.1: Procedural Programming in Java

RIT | Golisano College of Computing and Information Sciences

This Week: Python to Java

Unit 1: Python to Java				
Procedural Programming		Unit Testing with JUnit 5		File I/O and Exceptions

Python to Java

This unit will mostly focus on showing how to use Java to do things that you already know how to do in Python. This mostly involves learning ***Java syntax***.

As we move further into the semester, we will more deeply explore ***object-oriented programming***, and advanced topics such as ***threading*** and ***networking***.

- In GCIS-123 we wrote software exclusively in the Python programming language.
  - For most of the course we used ***procedural programming***, which is a term for programs that use ***functions*** to implement most of the program requirements.
  - Towards the end of the course, we introduced ***object-oriented programming***.
- In GCIS-124 we will be using the ***Java Programming Language***, a fully object-oriented language.
  - Unlike Python, in Java ***all*** code must be inside of a class.
- During this unit we will focus on learning how to use Java to implement many of the programs that you already know how to write in Python.
  - Types, & Variables
  - Methods, Parameters, Arguments, & Return Values
  - Boolean Expressions, Conditionals, & Loops
  - Unit Testing with JUnit
  - Exceptions & File I/O
- Today we will focus on ***procedural programming in Java***.

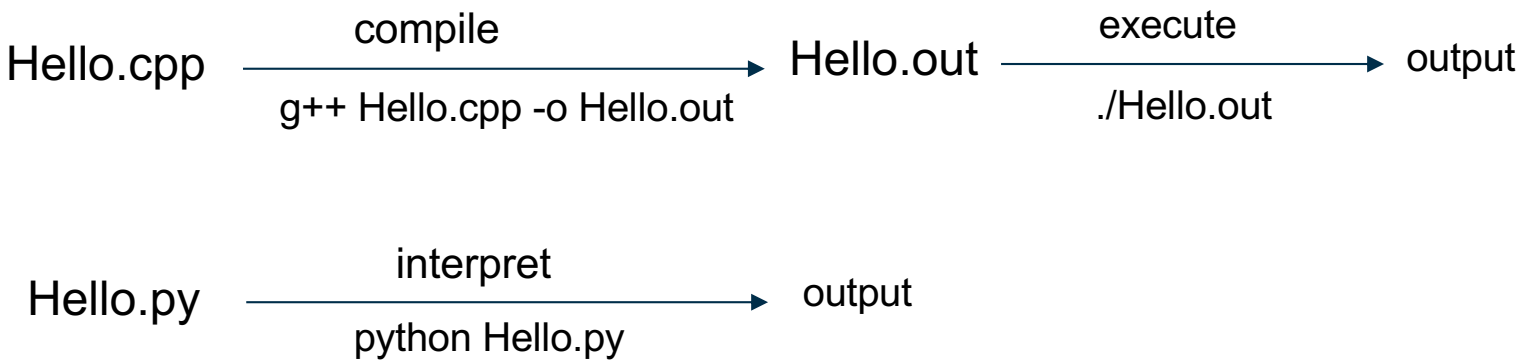
- Python is an incredibly powerful and flexible language.
  - It is also one of the most popular languages used today according to GitHub and Stack Overflow.
- It is also relatively quick and easy to pick up the basics.
  - "Hello, World!" is just `print("Hello, World!")`
- However, while Python supports classes and objects, it is not a great object-oriented language.
  - OO concepts like encapsulation, inheritance, overriding, and overloading all very oddly implemented in Python.
- There are also significant benefits to learning a new programming language.
  - Not only is it great for your **resume**, but the more languages that you learn, the more easily you will **pick up the syntax of new languages**.

Why Another Language?

Said no one, ever.

It's great to have a favorite language that is your "go to" for personal projects, but be careful not to get stuck in a professional rut.

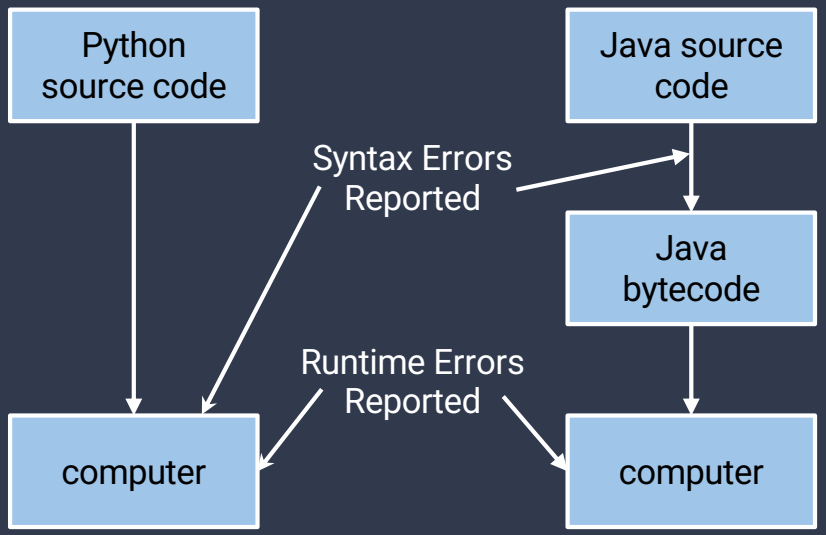
Compiler vs Interpreter



What about Java?

Java

Python is ***implicitly compiled*** and executed in a single step. Both ***syntax*** and ***runtime errors*** are reported at the same time.

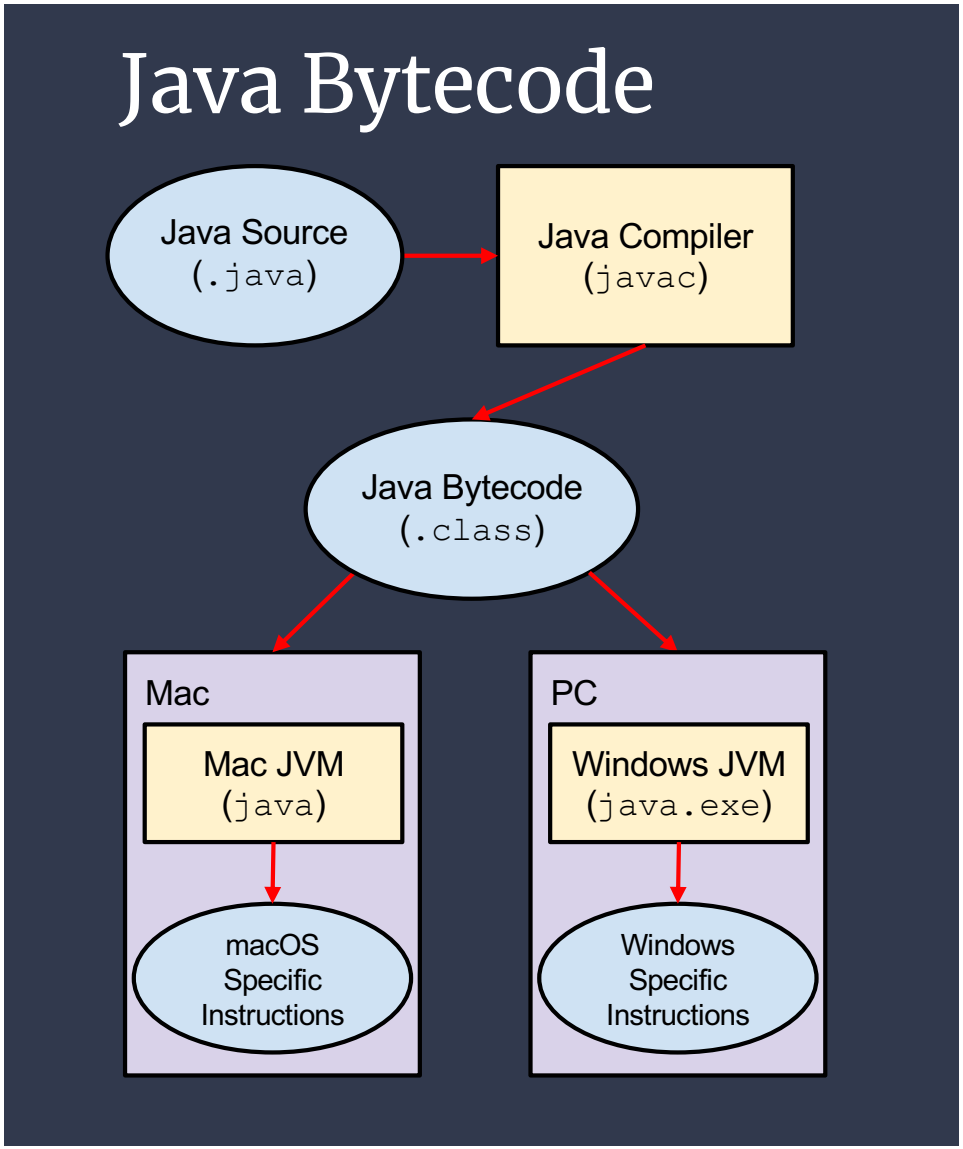


Java is ***explicitly compiled*** into ***bytecode*** first, at which time ***syntax errors*** are reported.

Once compiled, a Java program is executed as a separate step, at which time ***runtime errors*** are reported.

- Java is a ***fully*** object-oriented programming language.
  - This means that ***all*** code must be inside the ***body of a class***.
- Python is an ***interpreted*** language, meaning that code written in a Python program is interpreted at the time of execution.
  - This means that any ***syntax*** or ***runtime errors*** are reported at the same time.
- Java, on the other hand, is a ***compiled*** language.
  - Java programs are text files saved with a `.java` extension, e.g. `HelloWorld.java`.
  - Each must be ***explicitly compiled into bytecode*** using `javac` before it can be executed.
  - Any ***syntax errors*** are reported at this time.
  - The bytecode is saved in a `.class` file with the same name, e.g. `HelloWorld.class`.
- Once a Java program is compiled, it can be executed using `java`.
  - The program ***must*** include a ***main method*** with a specific signature to be executed.
  - Any ***runtime*** errors are reported at this time.

- ***High-level languages*** like Python and Java are meant for ***humans*** to understand.
  - A higher-level language has syntax that is closer to written or spoken language.
  - Python is a (slightly) higher-level language than Java.
  - On the other hand, ***assembly*** is a ***very*** low-level language.
- ***Machine code*** is very low-level code that can be executed directly on a computer.
  - Different computers understand different dialects of machine code.
- ***Bytecode*** is somewhere in the middle.
  - ***Lower-level*** than Python or Java source code.
  - ***Higher-level*** than machine code.
- Bytecode is much faster/easier to ***interpret*** (translate) into machine instructions than source code is.
- A ***virtual machine*** (***VM***) interprets bytecode into machine code at runtime.
  - In this way, bytecode is like machine code for the virtual machine.
  - The compiled bytecode never needs to change, but a different VM is needed for each operating system and/or processor to translate it properly.



1.1.2

Java Version

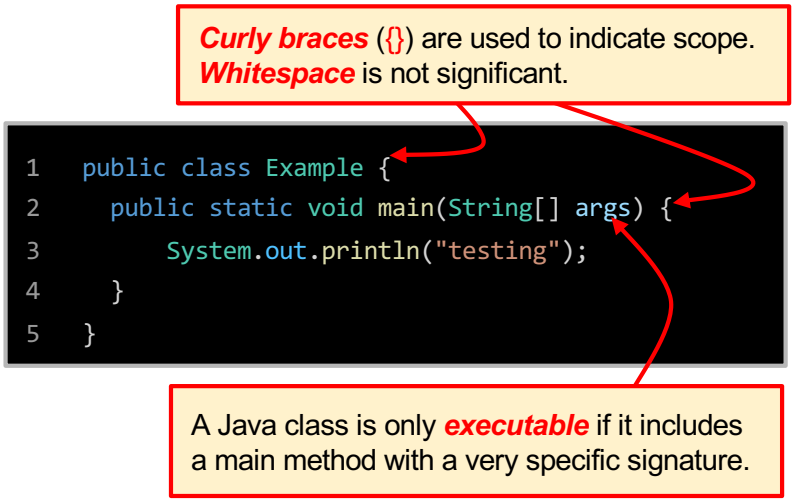
Before we can do anything else, you will need to make sure that the correct version of Java has been installed on your computer. Take a moment to verify that both Java and the Java Compiler have been installed and configured correctly.

- Open the terminal in VS Code (`CTRL-`` `) and run `javac -version` to verify the version of the ***Java Compiler*** that you have installed.
  - This is the executable that you will use to compile your Java programs.
- Next, run the `java -version` command to display the version of the ***Java Development Kit*** (***JDK***) that you have installed.
  - This is the executable that you will use to run your Java programs.
- Both executables should have the same version number.

```
C:\Users\dick\SoftDevII\Unit01\javac -version
javac 15.0.1
C:\Users\dick\SoftDevII\Unit01\java -version
java version "15.0.1" 2020-10-20
Java(TM) SE Runtime Environment (build 15.0.1+9-18)
Java HotSpot(TM) 64-Bit Server VM (build 15.0.1+9-18, mixed mode, sharing)
```

Scaffolding

- Unlike Python, Java requires quite a bit of ***scaffolding*** to write even the most basic programs.
  - Part of the reason for this is that ***all*** code in Java must be ***in the body of a class***, and so at a minimum a class must be declared.
- Java has a C-like syntax meaning:
  - ***Whitespace*** is ***not significant*** beyond improving readability for humans. This includes newlines.
  - ***Statements*** are terminated by a ***semi-colon*** (***;***) and may be written on multiple lines.
  - ***Curly braces*** ({ }) are used to indicate a ***block of statements***, e.g. the body of a method or a class.



A Closer Look (or "Why we don't use Java in 123.")

In Java, using the `public` ***access modifier*** makes things (classes, methods, fields) accessible from ***anywhere*** outside of the class. For now will make all classes and methods `public`.

The name of the class must ***exactly match*** the name of the file (including case), e.g. the `Example` class must be in a file named `Example.java`.

```
1   public class Example {
2       public static void main(String[] args)  {
3           System.out.println("testing");
4       }
5   }
```

The `static` keyword indicates that the method ***belongs to the class*** and can be called ***without*** creating an object.

Whitespace is ***insignificant*** in Java. Instead, ***curly braces*** (`{}`) define blocks of code. Indents are used for ***readability only***.

In Java, `System.out` refers to ***standard output***. The `print` and `println` methods can be used to print virtually ***any*** type including strings, integers, and so on.

Java strings ***must*** be enclosed in ***double-quotes*** (`""`). You ***do not*** have the option of using single-quotes or triple-quotes of any kind.

1.1.3

Hello, World!

"Hello, World!" is often the first program written when learning the syntax of a new programming language. Let's start learning Java syntax by implementing it now.

```
1   public class Example {
2     public static void main(String[] args){
3         System.out.println("testing");
4     }
5   }
```

```
C:\Users\damian\>javac Example.java
C:\Users\damian\>java Example
testing
C:\Users\damian\> _
```

- Create a new Java class in a file named "`HelloWorld.java`".
  - Define a `main` method with the appropriate signature.
  - Print `"Hello, World!"` to standard output.
- Open the terminal in VS Code (`CTRL-`` `) and use the Java Compiler to compile your new class.
  - e.g. `javac HelloWorld.java`
  - If there are any syntax errors, correct them in the editor and try to compile your class again.
  - List the files in the directory to verify that your `.class` file has been created.
- Once you have successfully compiled the class, run it using `java`.
  - e.g. `java HelloWorld`
  - ***Do not*** include the `.class` extension.

- Java is a ***statically typed*** language, meaning that the type of a variable must be specified ***when it is declared***.
  - Once declared, only ***compatible*** values may be assigned to the variable.
- Unlike Python, a Java variable may be declared ***without*** immediately assigning a value.
  - e.g. `int radius;`
  - Variables may also be ***declared and initialized*** at the same time, e.g. `double pi = 3.14159;`
- Like Python, Java includes ***two*** basic kinds of types.
  - ***Primitive types*** include ***integers***, ***floating point values***, ***characters***, and ***booleans***.
    - Primitive types are very similar to Python's value types.
    - Java includes a ***character type***, a single character enclosed in single-quotes, e.g. `char c = 'c';`
  - ***Reference types*** are, well, everything else including ***strings***, ***arrays***, and other ***objects***.
    - Reference types work the same in Java as they do in Python.

Variables & Assignment

Type	Description	Example
`byte`	8-bit signed integer, `-128` to `127`.	`123`
`short`	16-bit signed integer.	`12345`
`int`	32-bit signed integer.	`1234567`
`long`	64-bit signed integer.	`1234567l`
`char`	16-bit unsigned integer (Unicode).	`'a'`
`float`	32-bit signed floating point value.	`3.14159f`
`double`	64-bit signed floating point value.	`-0.1234`
`boolean`	Boolean value; `true` or `false`.	`true`

Java assumes that ***literal numbers*** are `int` or `double`, and so a suffix of `l` (lowercase `L`) is needed for a `long` literal (e.g. `30987653411`), and `f` for a `float` literal (e.g. `1.3f`).

Variables in Java

Because Java is a statically typed language, variables must be declared with a valid type. Only values of a compatible type may be assigned to the variable.

Like any other statement in Java, a variable declaration must be terminated with a semicolon.

```
1  public static void variables() {
2          double weight = 65.5;

           int age;

           age = 10;

           System.out.println("weight = " + weight
           + ", age = " + age);
10 }
```

Unlike Python a variable may be declared without immediately assigning a value…

…but a value must be assigned before the variable can be used.

Unlike Python, the + operator can be used to concatenate ***any*** type onto a string (not just other strings).

1.1.4

```
public static void variables() {
   String name = "Buttercup";

   int age;

   age = 10;

   System.out.println("name = " + name
      + ", age = " + age);
}
```

Primitive Types

Primitive types in Java include integers (`byte`, `short`, `int`, `long`), floating point values (`float`, `double`), characters (`char`), and `boolean` values. Practice declaring a few variables using different primitive types and printing them to standard output.

- Create a new Java class in a file named "`Primitives.java`" and define a `main` method with the appropriate signature.
  - Declare variables of at least ***three*** different ***primitive types*** and print the ***value*** to standard output.
  - e.g. `int x = 5;`
  - ***Hint***: unlike Python, you can concatenate a value of ***any*** Java type onto a string using the `+` operator, e.g. `System.out.println("four = " + 4);`
- ***Compile*** and ***run*** your new class from the terminal.

- Java supports ***most*** of the same basic ***arithmetic operators*** that Python does with a couple of notable exceptions.
  - There is no power (`**`) operator in Java, but the `Math.pow()` method may be used instead, e.g. `Math.pow(2, 7)` will return $2^7 = 128$.
  - There is no integer division operator (`//`) in Java, but using standard division with two integer values has the same result.
- Applying an operator to two values of the ***same type*** will produce ***a value of that type***.
  - For example, `10 / 3` will perform ***integer division***. In this case, the expression would evaluate to an `int` (`3`).
- If values of different types are mixed in the same expression, the result is "***promoted***" to the ***most complex type***.
  - For example `2.5 * 3` would evaluate to a `double` (`7.5`).

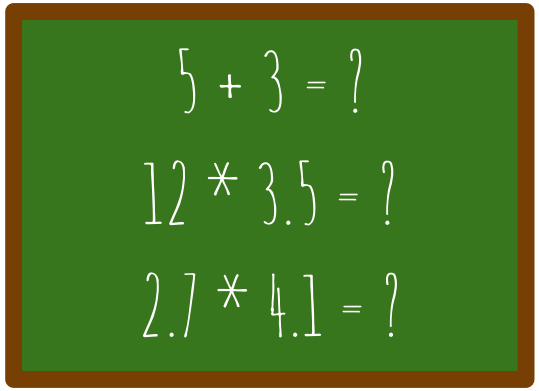
Arithmetic Operators

Operator	Description	Example
`+`	Addition	`double x = 5.1 + 3;`
`-`	Subtraction	`double z = 5.1 - 3;`
`*`	Multiplication	`int x = 3 * 4;`
`/`	Division	`double x = 5.1 / 3;`
`/ (integers)`	Integer Division	`int x = 5 / 2;`
`Math.pow(x, y)`	Power: X^Y	`double z;` `z = Math.pow(x, y);`
`%`	Modulo	`m = 10 % 3`

The `+` ***operator*** also works for ***string concatenation***, just as it does in Python. A notable difference is that ***any type*** can be concatenated onto a string without using a function like `str()`, so `"abc" + 123` ***is valid***.

1.1.5

Arithmetic



Operator	Description	Example
+	Addition	`double x = 5.1 + 3;`
-	Subtraction	`double z = 5.1 - 3;`
*	Multiplication	`int x = 3 * 4;`
/	Division	`double x = 5.1 / 3;`
/ (integers)	Integer Division	`int x = 5 / 2;`
Math.pow(x, y)	Power: X^Y	`double z;` `z = Math.pow(x, y);`
%	Modulo	`m = 10 % 3`

- Create a new Java class in a file called "`Arithmetic.java`" and define a `main` method with the appropriate signature.
  - Use `System.out.println` to print the results of using arithmetic operators with combinations of different types, e.g.:
    - `int` and `int`
    - `int` and `double`
    - `double` and `double`
  - For example: `System.out.println("12 * 3.6 = " + (12 * 3.6));`
  - There is no need to declare variables, but remember that Java will consider a ***literal integer*** to be an `int` (32-bit) and a literal ***floating point value*** to be a `double` (64-bit).
- ***Compile*** and ***run*** your class from the terminal.

Strings & Characters

```
1  String aString = "Jason";
2  char first = aString.charAt(0);
3  char last = aString.charAt(4);
4
5  System.out.println(first);
6  System.out.println(last);
```

Java strings do not support the use of square brackets (`[]`), but the `charAt(int index)` method will return the `char` at a specific index.

As with Python, ***valid indexes*** range from ***0*** to ***length-1***. Java ***does not*** support negative indexes.

- Java ***strings*** have a lot in common with Python strings.
  - They comprise ***characters***.
  - ***Literal strings*** are enclosed in ***double-quotes*** (`""`).
  - Strings are ***immutable***.
  - The `+` ***operator*** can be used to create a new string by ***concatenating*** two strings together.
  - Strings are ***reference types***.
- There are some key differences as well.
  - Literal strings ***cannot*** use single-quotes (`''`) or triple-quotes of either type.
  - Java includes a `char` type that represents a single character. Single quotes are used for `char` ***literals***, e.g. `'a'`, `'1'`, `'&'`.
  - The `+` ***operator*** can concatenate strings together with ***any*** other Java type.
  - Java strings ***do not*** work with square brackets (`[]`). Instead the `charAt(int index)` method is called on the string to get the `char` at a specific index.
  - Strings are not ***iterable***.
- Also like Python, Java makes use of a ***string literal pool***.
  - If the same string literal is used in more than one place, a single copy of the string is stored and reused.

Methods & Parameters

All Java methods must declare a ***return type***. A `void` return type indicates that the method does not return a value.

```
1  public static void example(int x, double y) {
2      System.out.println(x + " * " + y
3          + " = " + (x * y));
4  }
```

Parameters must declare a ***type*** as well as a ***name***. When the method is invoked, a ***compatible*** argument ***must*** be provided for each parameter.

- There are some key differences between Python and Java.
  - There is ***no*** `self` ***parameter***.
  - Method parameters ***must*** be declared with a ***type*** as well as a ***name***.
  - All methods ***must*** declare a ***return type***. A method that does not return anything declares a void return type.
  - An ***access modifier may*** be used to set the ***visibility*** of the method outside of the class. In this unit, all of the methods that we write will be public.
- In Python, a method that does not explicitly return a value returns `None` by default. In Java, a method with a void return type ***does not return anything*** at all.
  - Trying to assign the return value of a void method to a variable is a ***compiler error***.

1.1.7

Hello Methods

Practice now by writing a Java method and calling it from main.

```
public static void example(int x, double y) {
   System.out.println(x + " * " + y
      + " = " + (x * y));
}
```

- Define a method called "`helloName`" that declares `String` parameters for a `first` and last `name`.
  - Don't forget to make the method `public` and `static`.
  - Print a message in the format `"Hello, <first> <last>!"` to standard output.
- Call your method from `main`.
- ***Compile*** and ***run*** your class from the terminal.

- All Java methods ***must*** declare a ***return type***, even if no value is returned.
  - The return type is declared as part of the ***method signature***.
- Unlike Python, there is ***no*** default return type in Java. The `void` return type indicates that a method ***does not*** return a value of ***any type***.
  - A `void` method ***cannot*** be assigned to a variable, and trying to do so will cause a ***compiler error***, e.g. `int x = someVoidMethod();`
  - A `void` method ***cannot*** return a value, though it may use an empty `return` ***statement***, e.g. `return;`
- If a method declares a return type of ***anything other than*** `void`, then a value of that type ***must*** be returned by the method using a `return` statement, e.g. `return 5;`

Return Values

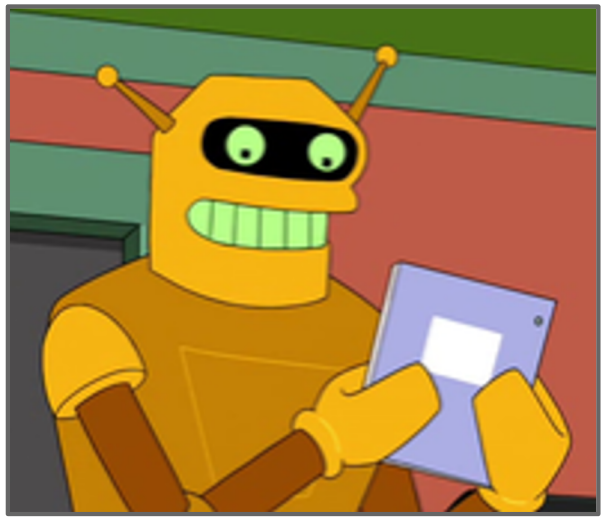
A method that declares a void ***return type*** does not return a value of any type, though a return statement can be used without a value.

```
1  public static void someVoidMethod() {
2      System.out.println("no return");
3      return;
4  }
5
6  public int intReturn() {
7      return 5;
8  }
```

A method that declares a return type of any other type ***must*** return a value of a ***compatible type***. If not, the method will ***not compile***.

1.1.8

Calculon



In Python, all methods return a value (even if that value is `None`). In Java, a method must declare its return type - if the return type is not `void`, the method ***must*** return a value of the correct type or the program will not compile.

```
public int intReturn() {
   return 5;
}
```

- Define a method for below operations, each of which declares ***two floating point parameters*** and returns a floating point result.
  - `add`
  - `divide`
- Test your methods by calling them from `main` and printing the results to standard output.
- ***Compile*** and ***run*** your new class from the terminal.

Conditionals

Python	Java	Example
`not`	`!`	`!a, !(a || b)`
`or`	`||`	`a || b`
`and`	`&&`	`a && b`
`^`	`^`	`a ^ b`
`is, is not`	`==, !=`	`a == b, a != b`
`==`	`== (primitives)` `equals(Object)`	`x == 5` `a.equals(b)`
`<, <=`	`<, <=`	`a < b, a <= b`
`>, >=`	`>, >=`	`a > b, a >= b`

- Recall that a ***boolean expression*** is one that, when evaluated, results in a `boolean` ***value***, i.e. `true` or `false`.
- Just like Python, ***logical*** and ***comparison operators*** can be combined to create complex boolean expressions in Java.
- Also like Python, Java supports ***conditional statements*** that work very much the same.
  - `if(expression) {...}` - executes the statements in the body if the expression is `true`.
  - `else {...}` - executes the statements in the body if all of the ***preceding conditions*** are `false`.
- The most notable difference is that Java ***does not*** include an `elif` statement.
  - Instead, `else if(expression) {...}` is used.

```
1  if(a && b) {
2      System.out.println("foo");
3  } else if(b ^ c) {
4      System.out.println("bar");
5  } else {
6      System.out.println("foobar");
7  }
```

`while` Loops

- Java `while` loops work ***exactly*** the same way as while loops in Python with a few small ***syntactic differences***.
  - The boolean expression that controls the loop ***must*** be enclosed in ***parentheses*** (`()`).
  - If the ***body*** of the loop contains more than one statement, it ***must*** be enclosed in ***curly braces*** (`{}`).
- Java also supports statements for more ***finely grained*** control of a `while` loop.
  - The `break` statement will ***terminate*** a loop ***immediately***, regardless of the boolean expression.
  - The `continue` statement will ***stop the current iteration***. The boolean expression will be evaluated to determine whether or not the next iteration should be executed.

A `while` loop will execute ***zero or more*** iterations depending on whether the boolean expression is `true` the very first time it is evaluated.

```
1   int i = 1048576;
2   while(i != 2) {
3     System.out.println(i);
4     i = i / 2;
5   }
```

The loop will continue iterating until the boolean expression evaluates to `false`.

"Classic" `for` Loops

- The ***loop declaration*** includes three statements separated by ***semicolons*** (`;`).
  - The ***initialization statement*** initializes the counting variable and is executed ***exactly once*** before the first iteration, e.g. `int i=1048576;`
  - The ***boolean expression*** that controls the loop. It is evaluated ***before*** each iteration of the loop, e.g. `i!=2;`
  - The ***modification statement*** that is executed ***after*** each iteration of the loop, e.g. `i=i/2;`
- All three statements are ***optional***.
  - If omitted, the boolean expression is ***always*** `true`.
- Java ***also*** includes a `for` ***each loop*** similar to Python's that works with iterable types.
  - We'll take a look at it later in this unit.

The ***"classic"*** `for` ***loop*** uses a ***C-like syntax*** to make a more syntactically compact version of a counting `while` loop.

```
1  for(int i=1048576; i != 2; i=i/2) {
2    System.out.println(i);
3  }
```

```
1   int i = 1048576;
2   while(i != 2) {
3     System.out.println(i);
4     i = i / 2;
5   }
```

The ***initialization statement***, ***boolean expression***, and ***modification statement*** are all included in the loop declaration.

1.1.11

Counting up with `for`

Implement a method that counts from `0` to `n` using a `for` loop.

- Define a method called "`countFor`" that declares a parameter for an integer `n`.
  - Use a `for` ***loop*** to print a count from `0` ***to*** `n`.
  - Return the ***sum*** of the numbers that are counted.
- Call your method from `main` and print the sum.
- ***Compile*** and ***run*** your new class from the terminal.

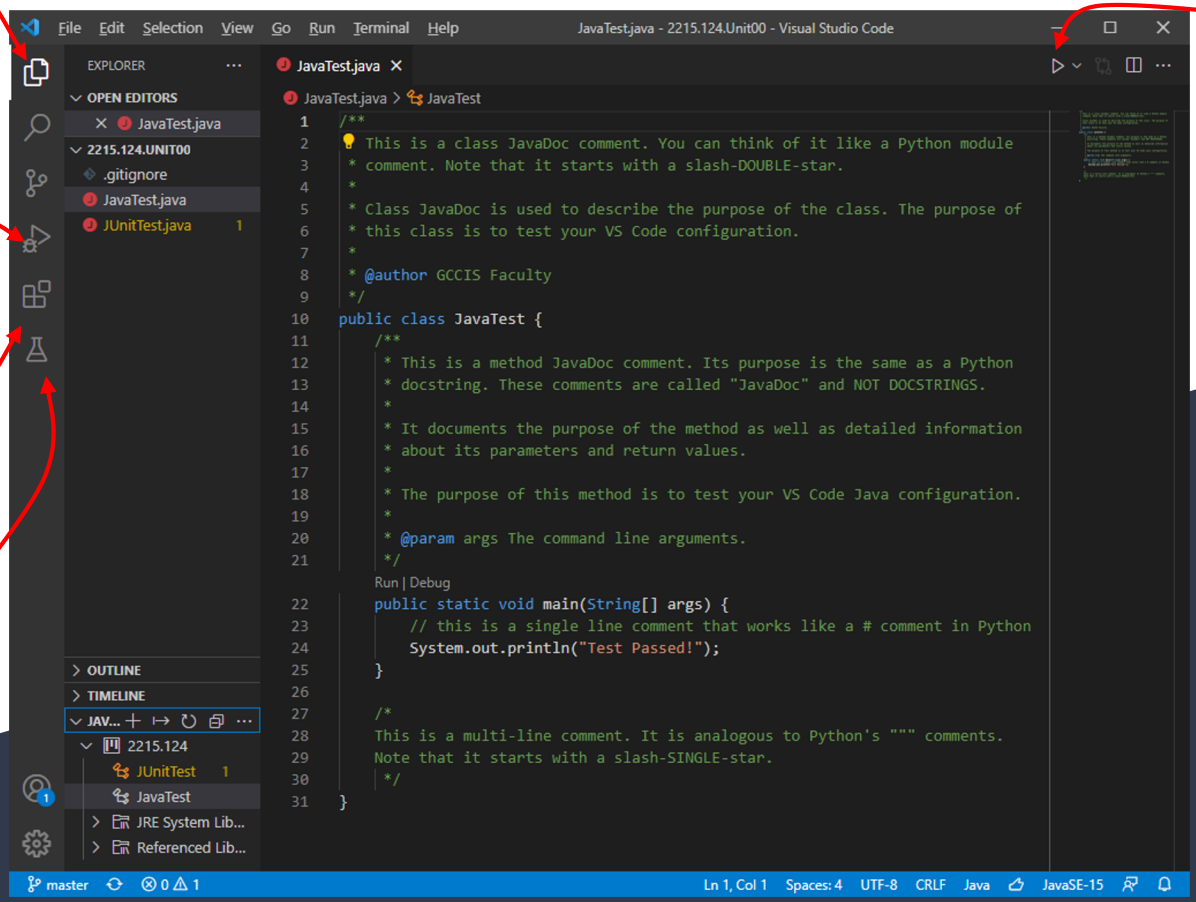
What if I want to use an IDE?
See next slides for using VS Code

EXPLORER
Used to view project files & settings

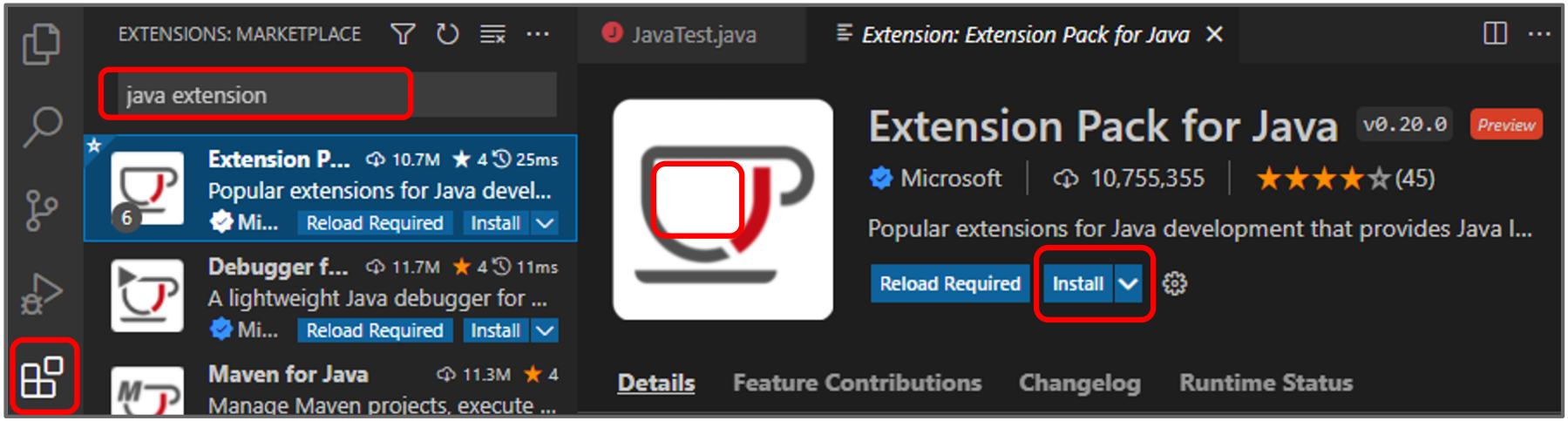
DEBUG
Used to run Java programs through the debugger

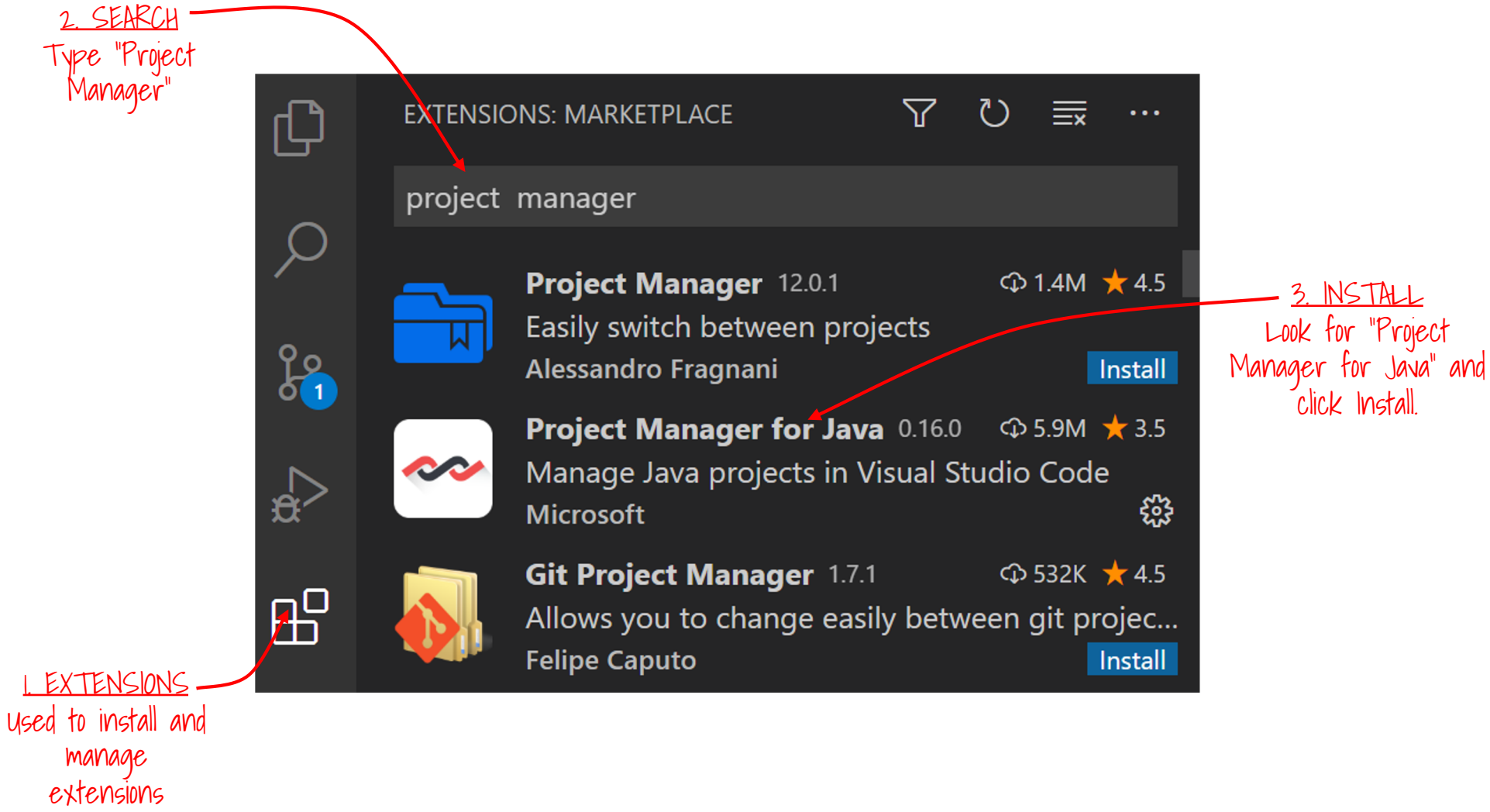
EXTENSIONS
Used to install and manage extensions

TEST (FLASK)
Used to run Junit unit tests

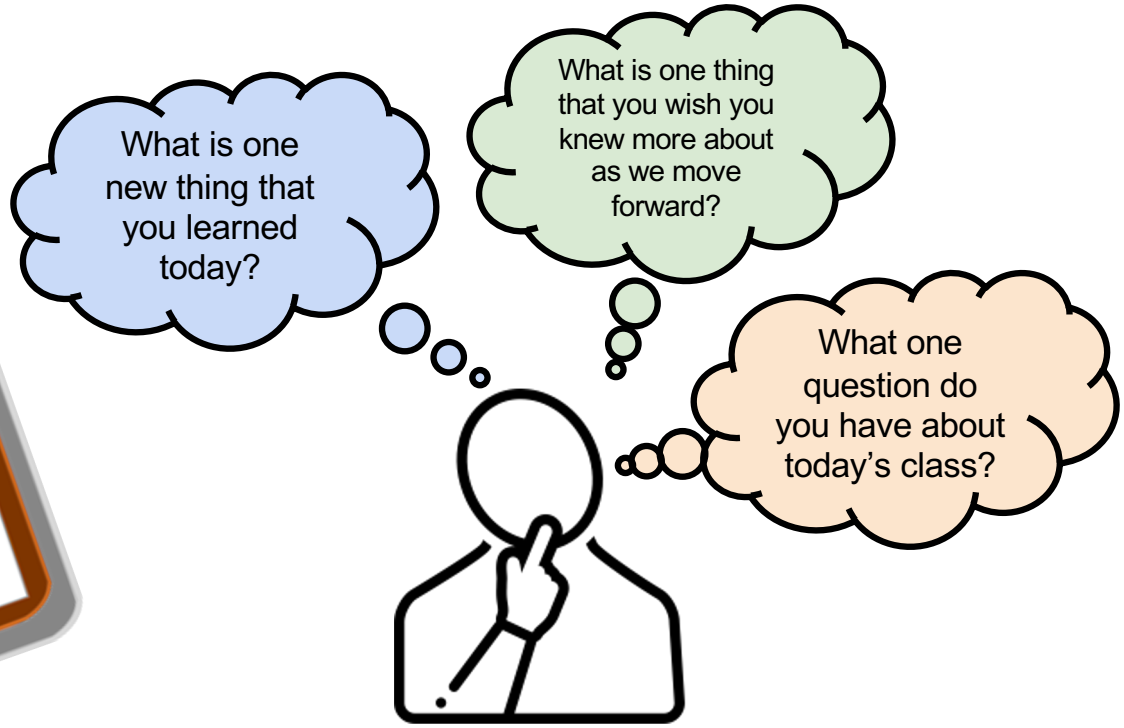


RUN BUTTON
Used to run Java classes


EXTENSIONS: MARKETPLACE
java extension
Extension P... 10.7M 4 25ms
Popular extensions for Java devel...
Mi... Reload Required Install
Debugger f... 11.7M 4 11ms
A lightweight Java debugger for ...
Mi... Reload Required Install
Maven for Java 11.3M 4
Manage Maven projects, execute ...
JavaTest.java
Extension: Extension Pack for Java
Extension Pack for Java
v0.20.0
Preview
Microsoft
10,755,355
(45)
Popular extensions for Java development that provides Java l...
Reload Required
Install
Details
Feature Contributions
Changelog
Runtime Status


2. SEARCH
Type "Project Manager"
EXTENSIONS: MARKETPLACE
project manager
Project Manager 12.0.1
1.4M
4.5
Easily switch between projects
Alessandro Fragnani
Install
3. INSTALL
Look for "Project Manager for Java" and click Install.
Project Manager for Java 0.16.0
5.9M
3.5
Manage Java projects in Visual Studio Code
Microsoft
Git Project Manager 1.7.1
532K
4.5
Allows you to change easily between git projec...
Felipe Caputo
Install
1. EXTENSIONS
Used to install and manage extensions

Summary & Reflection



Please answer the questions above in your notes for today.