CS1530, LECTURE 7: THE GRADLE BUILD TOOL

Bill Laboon

WHAT IS A BUILD TOOL?

- A tool or framework which allows you to centrally manage aspects of developing software
- Especially useful on multi-party teams, to synchronize development environments
- Recall that developing software is much more than writing code!

WHAT DOES THAT ENTAIL?

- Compile and run code
- Create and initialize new projects
- Download, update, and manage external libraries and other dependencies
- Automatically run tests (unit, system, other)
- Update databases, files or other data
- … and more!

NOT TO BE CONFUSED WITH CI (CONTINUOUS INTEGRATION) TOOLS

- Continuous integration: a philosophy of constantly integrating features / functionality / bug fixes to the mainline (master branch), instead of scheduling integration at certain points
- As you might imagine, keeping track of this can be difficult!
- Tools such as Jenkins can help to ensure quality of code checked in (by e.g. automatically running tests), but these are run ON TOP OF a build system
- We will discuss more later when we discuss integration!

THE BUILD TOOL WE WILL USE: GRADLE

- Remember that my goal is NOT to teach you all the ins and outs of Gradle - it is to teach you how to engineer software, and a build system is almost a prerequisite for non-trivial systems
- Software Engineering is learned through experience, and so you need to learn SOME build tool!

LOTS OF POPULAR BUILD TOOLS

- Usually only one or a few are extremely popular for a language
  - leiningen: Clojure
  - rake: Ruby
  - stack or cabal: Haskell
  - Nant or MSBuild: .NET frameworks
  - cargo: Rust
  - GNU make, cmake, nmake: C / C++

JAVA HAS THREE BIG ONES!

- Apache Ant + Ivy
- Apache Maven
- Gradle

WHY GRADLE?

- **Batteries included -** You don't have to specify/pull down all sorts of extras like with ant.
- **But still very flexible -** It's not nearly as rigid as maven.
- **Powerful -** You can write tasks in their own Turing-complete language, Groovy
- **Easy to understand -** You'll have a fully functioning system up in no time.
- **Seems to be the direction industry is heading in -** I asked my friends in the Java development community, everybody I discussed said that gradle's the way to go.
- **Polyglot -** You can use Gradle with other languages (> 60).
- **I hate XML, and Gradle doesn't use it, unlike ant and maven -** Self-explanatory.

HOW DO I GET IT?

- Go to https://gradle.org/gradle-download/
- Windows: Download and install
- OS X: You can download, but if you use Homebrew, just "brew install gradle" (if not, install homebrew first!)
- Linux (Ubuntu): With apt-get -

```
sudo add-apt-repository ppa:cwchien/gradle
sudo apt-get update
sudo apt-get install gradle
```

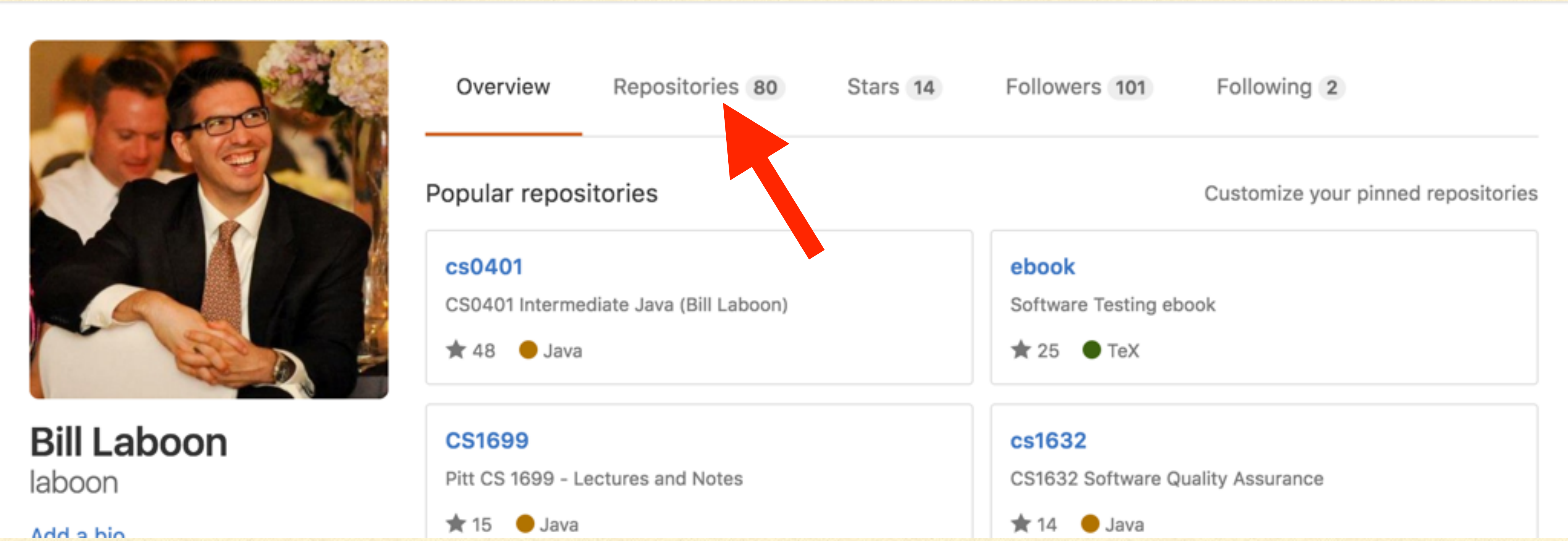
WHAT CAN I DO WITH IT?

- Let’s create an application…
- Add some tasks…
- Add some code…
- Add an external dependency…
- Run some tests…
- … and put it all together!

CREATING A NEW GRADLE BUILD

- Terminology note: A “build” can contain multiple “projects”. But in this case, the build will only contain one project.
- Let’s start by creating a git repository
    - Honestly, you should be doing all of your development with some sort of version control!

CLICK ON "REPOSITORIES" ON YOUR GIT USER PAGE



Bill Laboon
laboon

Popular repositories

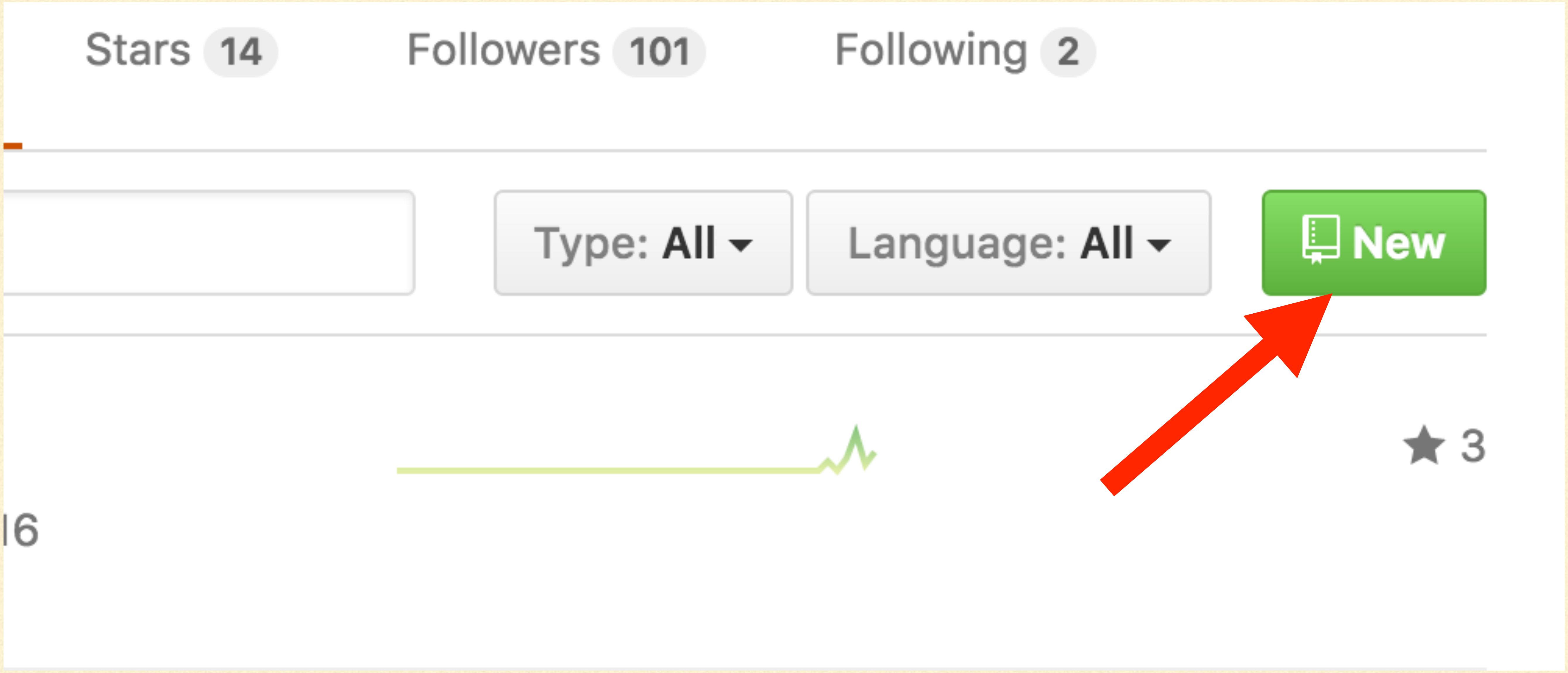
cs0401
CS0401 Intermediate Java (Bill Laboon)
★ 48 ● Java

ebook
Software Testing ebook
★ 25 ● TeX

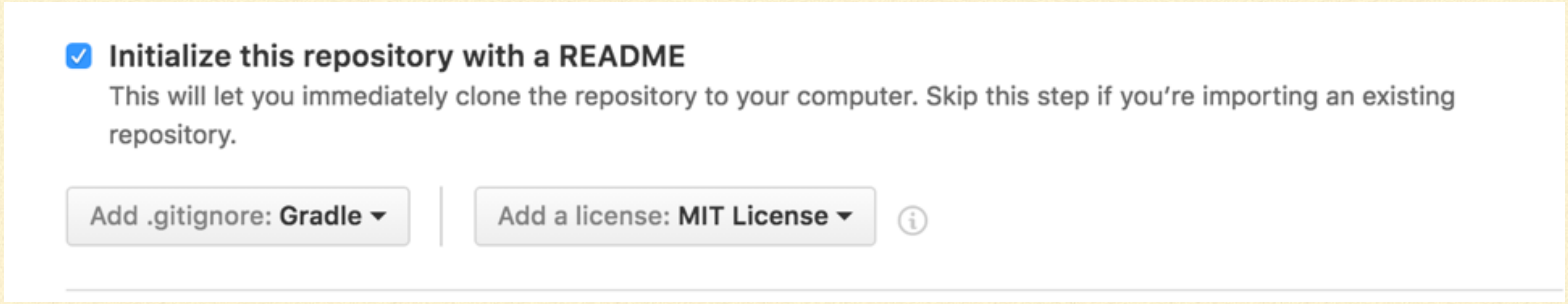
CS1699
Pitt CS 1699 - Lectures and Notes
★ 15 ● Java

cs1632
CS1632 Software Quality Assurance
★ 14 ● Java

MAKE A NEW REPOSITORY



CREATING A REPO FOR OUR GRADLE BUILD



☑ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **Gradle** | Add a license: **MIT License**

PROTIP: Add a default Gradle .gitignore file when you create the repository (on the repo create page). This will automatically prevent git from adding unnecessary files to your repo.

CLONE DOWN THE REPO TO YOUR LOCAL MACHINE

```
(13591) $ git clone git@github.com:laboon/GradleSampleBuild.git
Cloning into 'GradleSampleBuild'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
Checking connectivity... done.
```

LET'S ADD A TASK!

- A "task" is something that gradle can do for you. Examples:
  - build (compile and link code)
  - run (execute)
  - test (run tests)
  - plumbus (hopefully I don't need to explain this one)

GROOVY

- We will need to add a task definition to the build.gradle file (which does not yet exist)
- This file is written in the language Groovy
    - Scripting language (like Python or Ruby)
    - Dynamically typed
    - Java-like syntax (most .java files can be run as Groovy!)
    - Runs on JVM

LET'S CREATE A PLUMBUS TASK

- In the root directory of your repo, add a file called build.gradle
- Add the following code:

```
task plumbus << {
  println "A plumbus for you!"
}
```

NOW PLUMBUS IS A TASK WE CAN RUN VIA GRADLE

```
(13599) $ gradle plumbus
:plumbus
A plumbus for you!

BUILD SUCCESSFUL

Total time: 1.779 secs

This build could be faster, please consider using the
Gradle Daemon: https://docs.gradle.org/2.13/userguide/
gradle_daemon.html
```

UGH CAN'T I GET RID OF ALL THE NON-PLUMBUS TEXT?

```
(13600) $ gradle -q plumbus
A plumbus for you!
```

TASKS CAN CALL OTHER TASKS

```
task plumbus << {
   println "A plumbus for you!"
}

task schleem(dependsOn: plumbus) << {
   println "schleem has been repurposed"
}

task grumbo(dependsOn: schleem) << {
  println "that leaves you with a regular old plumbus"
}

(13610) $ gradle -q grumbo
A plumbus for you!
schleem has been repurposed
that leaves you with a regular old plumbus
```

IF YOU SEE THIS, YOUR BUILD.GRADLE FILE IS NOT COMPILING (PROBABLY A TYPO)

```
13607) $ !!
gradle -q monkey

FAILURE: Build failed with an exception.

* Where:
Build file '/Users/laboon/pitt/GradleSampleBuild/build.gradle' line: 3

* What went wrong:
A problem occurred evaluating root project 'GradleSampleBuild'.
> Could not find property 'fdsnjlfsd' on root project 'GradleSampleBuild'.

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --
debug option to get more log output.
```

LIST ALL TASKS THAT YOUR GRADLE BUILD CAN HANDLE

```
(13602) $ gradle -q tasks

------------------------------------------------------------
All tasks runnable from root project
------------------------------------------------------------

Build Setup tasks
-----------------
init - Initializes a new Gradle build. [incubating]
wrapper - Generates Gradle wrapper files. [incubating]

Help tasks
----------
buildEnvironment - Displays all buildscript dependencies declared in root project 'GradleSampleBuild'.
components - Displays the components produced by root project 'GradleSampleBuild'. [incubating]
dependencies - Displays all dependencies declared in root project 'GradleSampleBuild'.
dependencyInsight - Displays the insight into a specific dependency in root project 'GradleSampleBuild'.
help - Displays a help message.
model - Displays the configuration model of root project 'GradleSampleBuild'. [incubating]
projects - Displays the sub-projects of root project 'GradleSampleBuild'.
properties - Displays the properties of root project 'GradleSampleBuild'.
tasks - Displays the tasks runnable from root project 'GradleSampleBuild'.

Other tasks
-----------
plumbus
```

GROOVY

- A cool language, but we're just going to learn it as we go along - we won't need to know much syntax
- But it's Turing-complete and can access the Java API! You could write your whole program in Groovy if you want.

MAKING OUR GRADLE BUILD A JAVA BUILD

- Add the following to the top of your build.gradle file. We are adding the Java plug-in, and saying that we are going to download any needed dependencies from a place called mavenCentral.

```
apply plugin: 'java'
repositories {
    mavenCentral()
}
```

DECLARE DEPENDENCIES.

- Now add the following after the repositories block. We are only going to need junit for testing, so we add it to the testCompile group. If we needed it at runtime, we could also add it to the runtime group.

```
dependencies {
  testCompile 'junit:junit:4.12'
}
```

LET'S ADD SOME CODE

- Gradle assumes some things about your directory structure if you use the java plug-in. Specifically:
- src/main/java <— Where code lives
- src/test/java <— Where unit tests (automated tests) live

LET'S WRITE A FIBONACCI GENERATOR. KIDS LOVE FIBONACCI GENERATORS.

- In the root directory of your project, make the following directory /src/main/java/laboon and add the file Fibonacci.java
- Now insert the code in the next slide into that file.

FIBONACCI.JAVA CODE

```
package laboon;
public class Fibonacci {
    public static long fibonacci(long num) {
   if (num <= 1) {
       return num;
   } else {
       return fibonacci(num - 1) + fibonacci(num - 2);
   }
}


public static void main(String[] args) {
   if (args.length == 0) {
       System.out.println("Enter something buddy");
       System.exit(1);
   } else {
       try {
      long x = fibonacci(Long.parseLong(args[0]));
      System.out.println("Fibonacci is: " + x);
       } catch (NumberFormatException ex) {
      System.out.println("Enter an integer buddy");
      System.exit(1);
       }
   }

    }
}
```

BUILDING

- Go back to root directory and type: **gradle build**

```
:compileJava
:processResources UP-TO-DATE
:classes
:jar
:assemble
:compileTestJava UP-TO-DATE
:processTestResources UP-TO-DATE
:testClasses UP-TO-DATE
:test UP-TO-DATE
:check UP-TO-DATE
:build

BUILD SUCCESSFUL

Total time: 3.396 secs
```

IT BUILT OUR PROJECT FOR US!

- All of your classes are in the ./build subdirectory
- But we don't really need to know about that! We can do everything from the root directory.

LET'S ADD SOME TESTS

- In ./src/test/java/laboon, add the file FibonacciTest.java. These will be unit tests for our Fibonacci.fibonacci(long n) method.
- Add the code on the following slide to FibonacciTest.

FIBONACCITEST.JAVA

```
import org.junit.Test;
import static org.junit.Assert.*;

import laboon.Fibonacci;

public class FibonacciTest {

    @Test
    public void testFibonacci0() {
       assertEquals(Fibonacci.fibonacci(0), 0);
    }


    @Test
    public void testFibonacci1() {
       assertEquals(Fibonacci.fibonacci(1), 1);
    }

    @Test
    public void testFibonacci9() {
       assertEquals(Fibonacci.fibonacci(9), 34);
    }


}
```

CLEAN UP ANY OLD FILES (E.G. CLASS FILES)

```
(13644) $ gradle clean
:clean

BUILD SUCCESSFUL

Total time: 2.835 secs
```

RUN ALL UNIT TESTS (COULD BE OTHER TEST CLASSES HERE, TOO!)

```
A SUCCESSFUL TEST RUN – ALL TESTS PASS

(13645) $ gradle test
:compileJava
:processResources UP-TO-DATE
:classes
:compileTestJava
:processTestResources UP-TO-DATE
:testClasses
:test

BUILD SUCCESSFUL

Total time: 4.949 secs
```

SOMETIMES TESTS DON'T PASS

```
(13646) $ sed -n 8,11p ./src/test/java/laboon/FibonacciTest.java
    @Test
    public void testFibonacci0() {
       assertEquals(Fibonacci.fibonacci(0), 17);
    }

(13647) $ gradle test
...  I cut some text here ...

FibonacciTest > testFibonacci0 FAILED
    java.lang.AssertionError at FibonacciTest.java:10

3 tests completed, 1 failed
:test FAILED

FAILURE: Build failed with an exception.
```

RUN THE PROGRAM

- We will need to add another plug-in, ‘Application’
- Alternatively, we could make this a task!

ADD APPLICATION PLUG-IN

In build.gradle, add the following lines after the initial `apply plugin: ‘java’` line…

```
apply plugin: 'application'
mainClassName = “laboon.Fibonacci"
```

This tells Gradle that this an application whose main() method is in laboon.Fibonacci. We now have access to the gradle task “run”.

RUN THE APPLICATION

```
(13652) $ gradle -q run
Enter something buddy

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':run'.
> Process 'command '/Library/Java/JavaVirtualMachines/jdk1.8.0_73.jdk/
Contents/Home/bin/java'' finished with non-zero exit value 1
```

PASSING ARGUMENTS

- More complex than you would think! Add this re-definition of the run task to your build.gradle file…

```
run {
    systemProperties System.getProperties()
    args System.getProperty("args").split()
}
```

- Now pass in arguments to gradle with the -Dargs argument:

```
(13658) $ gradle -q run -Dargs="10"
Fibonacci is: 55
```

FINAL JOB: ADD TO GIT

- From root directory of build, type `git add .`
- Adds everything in directory and subdirectories to git repo. But ignores .class files and anything else in build subdirectory as directed by .gitignore! Now you can commit and push.

```
(13666) $ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

	modified:   build.gradle
	new file:   src/main/java/laboon/Fibonacci.java
	new file:   src/test/java/laboon/FibonacciTest.java
```

WORKED-THROUGH EXAMPLE

https://github.com/laboon/GradleSampleBuild