

# Principles of Software Construction: Objects, Design, and Concurrency

## IDEs, Build system, Continuous Integration, Libraries

Bogdan Vasilescu

Claire Le Goues



# Administrivia

Wait list update: should be able to basically clear it. We'll give updated deadlines for students who enroll late when they're officially enrolled.

Please don't email just one of us unless it's very sensitive.

- Only email us about big picture stuff.
- Waitlist questions: include Jenni Cooper on the CC

Please check Piazza periodically both for technical and non-technical questions!

- We encourage you to help each other out.

We'll be in a classroom next week.

# New staff, new OH, **new recitation.**

We have hired three new TAs! Welcome to Li Guo, Deyuan Chen, and Yuwei Li.

***There is a new recitation section: W 7-7:50.***

- It is remote only.
- AFAIK it doesn't exist yet but the registrar is working on it
- So we will have it this week.
- If you are on the waitlist and can attend this one, do try to.

***We now have a lot more office hours!***

Links/info on course calendar on website. Click on “more info” if necessary.

# Homework 1: Welcome to the deep end?

The milestone has set you up for the rest of the semester! Hooray!

All of the code necessary for the options we're asking you to support with the command line library exist in the code.

We're here to help! Come to recitation and/or Office Hours.

- We're using OHQueue (see Piazza).

# Mini-quiz

<https://forms.gle/gaCB7xkPFijaUYWwy6>



# Abstraction, Reuse, and Programming Tools

- For each in {IDE, Build systems, libraries, CI}:
  - What is it today?
  - What is under the hood?
- What is next?

# Automation Requires Abstraction



# Automation Requires Abstraction

```
C++ source #1
1 // Type your code here, or load an example.
2 int square(int num) {
3     return num * num;
4 }

#1 with MSP430 gcc 4.5.3 x
MSP430 gcc 4.5.3 compiler options...
11010 .LX0: .text // Intel
1
2 /*****
3  * Function `square(int)'
4  *****/
5 square(int):
6     push    r10
7     push    r4
8     mov     r1, r4
9     add     #4, r4
10    sub     #2, r1
11    mov     r15, -6(r4)
12    mov     -6(r4), r10
13    mov     -6(r4), r12
14    call    #__mulhi3
15    mov     r14, r15
16    add     #2, r1
17    pop     r4
18    pop     r10
19    ret
20 ;; End of function
```



# Automation Requires Abstraction

We all treat familiar levels of abstraction as normal/natural

- That's fine if you only drive your car
  - Not so much if you are a mechanic
  - How to debug a broken transmission?
- Also slow to evolve
  - *Conf.* people adamantly refusing to use an automatic
- Engineers seek out abstractions that simplify their work, help focus on the hard parts
  - They also know what is beneath the abstractions

# Automation Requires Abstraction

Today's “normal”:

- Integrated-development environments (IDEs) galore
  - Web-based too! Press “.” on a GitHub (file) page 🤖
- Frequent build, test, release
  - In some companies, every commit is a “release”
- Never write code for which there is a useful library
  - Define “useful”?
- All of the above, entangled

# Abstraction, Reuse, and Programming Tools

- For each in {IDE, Build systems, libraries, CI}:
  - What is it today?
  - What is under the hood?
- What is next?

# Abstraction, Reuse, and Programming Tools

- For each in {**IDE**, Build systems, libraries, CI}:
  - **What is it today?**
  - What is under the hood?
- What is next?

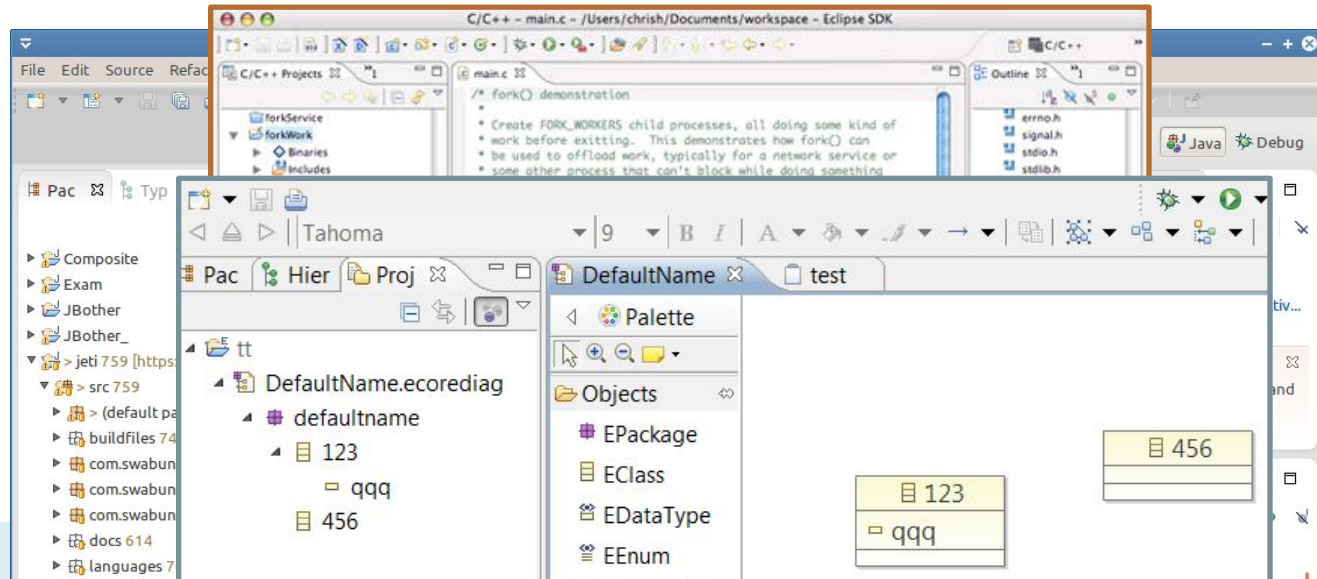
# Quick overview of today's toolchain: IDEs

Integrated Development Environments, bundle development workflows in a single UI

- Editing, refactoring, running & debugging, adding dependencies, compiling, deploying, plugins, you name it
- They often try to be everything, with mixed results
- Leverage them to the fullest extent, to automate and check your work

# Quick overview of today's toolchain: IDEs

Eclipse was the dominant player in Java for 20-odd years, owing to its powerful backbone and plugin architecture



# Quick overview of today's toolchain: IDEs

Recently, IntelliJ has been more dominant

- Packs a lot of “recipes” to create certain types of projects (e.g., web-app with Spring & Maven)

VSCode is surging in popularity

- Local & web, lightweight but with a massive plugin ecosystem
  - Quick tangent: if you can build either a large product or a platform, build a platform

But choose based on need!

- You can relearn key-bindings; “killer features” are rare and temporary
- E.g., Android: might want Android Studio (itself built on IntelliJ) since Google supports it
- We recommended VSCode for TS and IntelliJ for Java, but you can actually use either for both (and we don't care).

# Abstraction, Reuse, and Programming Tools

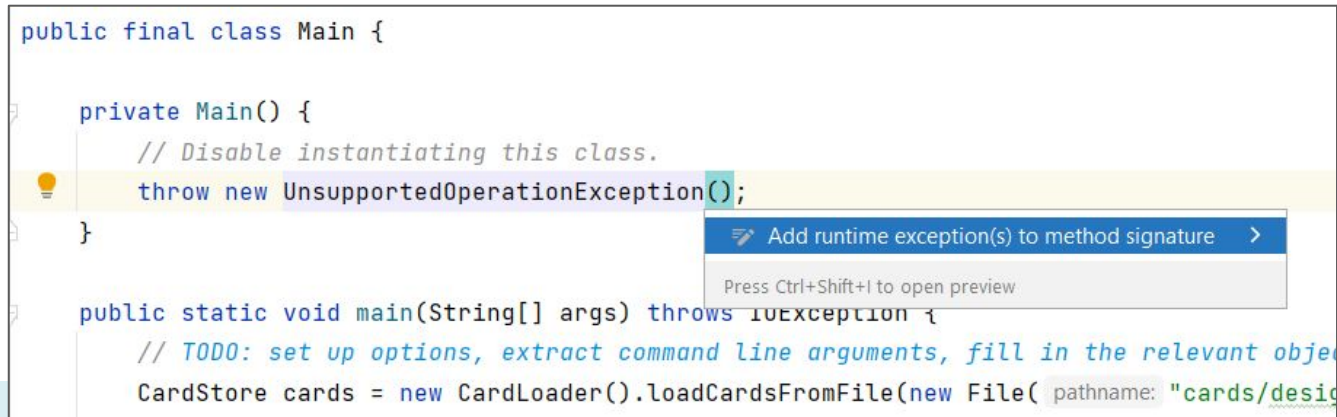
- For each in {**IDE**, Build systems, libraries, CI}:
  - What is it today?
  - **What is under the hood?**
- What is next?



# Under the Hood: IDEs

Automate common programming actions:

- Handy refactorings, suggestions
  - E.g., just press `alt+enter` in IntelliJ while highlighting nearly any code
    - Keyboard shortcuts are super useful: explore your IDE!
  - These can make you a better programmer: encode a lot of best-practices
    - Though, don't read into them too much



```
public final class Main {  
  
    private Main() {  
        // Disable instantiating this class.  
        throw new UnsupportedOperationException();  
    }  
  
    public static void main(String[] args) throws IOException {  
        // TODO: set up options, extract command line arguments, fill in the relevant objects  
        CardStore cards = new CardLoader().loadCardsFromFile(new File("cards/design"))  
    }  
}
```

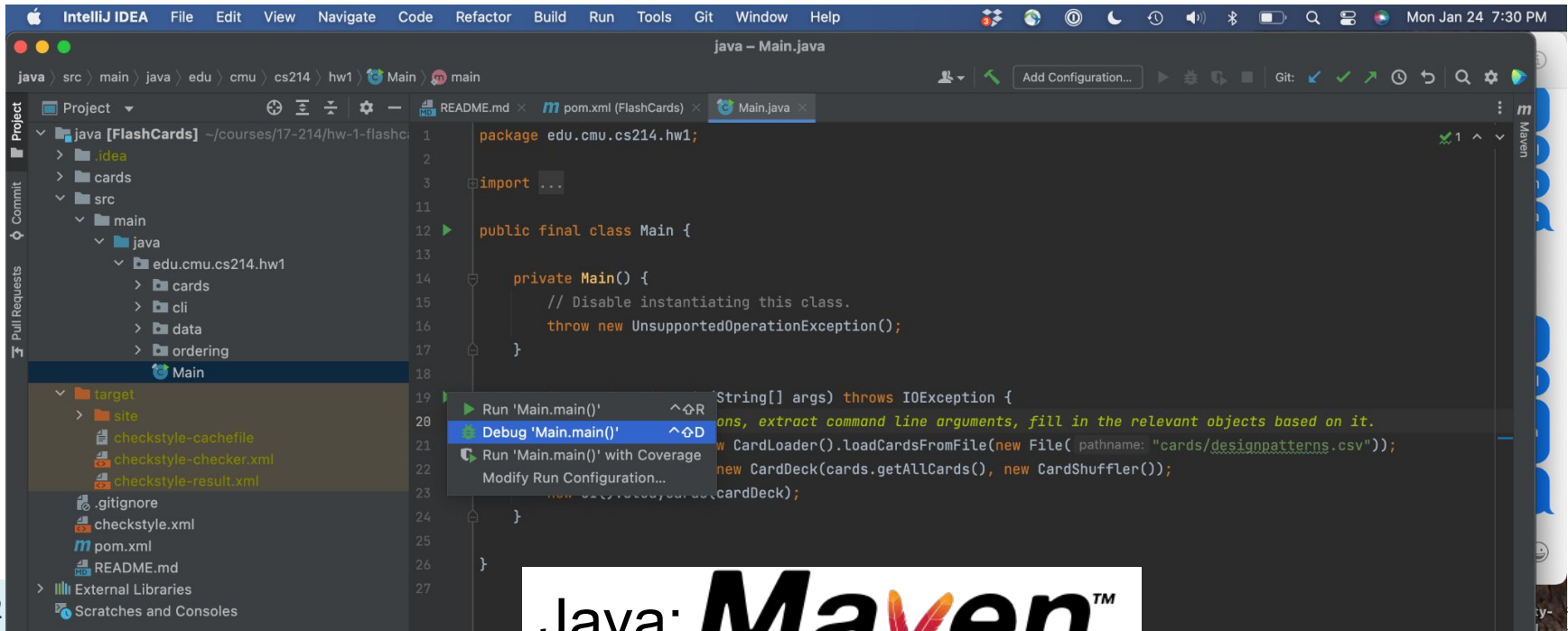
The screenshot shows the IntelliJ IDEA IDE with a code suggestion popup. The popup is triggered by the line `throw new UnsupportedOperationException();` in the `private Main()` method. The popup contains the text "Add runtime exception(s) to method signature" with a right arrow, and a hint "Press Ctrl+Shift+I to open preview". The code is partially visible, showing the `Main` class with a `private Main()` method and a `public static void main` method.

# Under the Hood: IDEs

- The engine: continuous parsing, building
  - Key feature: most partial programs don't parse, but IDEs make sense of them
  - That allows quickly relaying compile warnings/errors and useful suggestions
  - Same with API resolution
- Powered by rapid incremental compilation
  - Only build what has been updated
    - Virtually every edit you make triggers a compilation, re-linking
    - Of just the changed code and its dependencies
  - Works because *very little* of the code changes most of the time
    - But no free lunch: tends to drop optimizations (mostly fine), may struggle with big projects
  - Just try it: call an API with the wrong parameters & see how fast it triggers an alert; contrast with running a full Maven build (e.g., with `mvn install`)

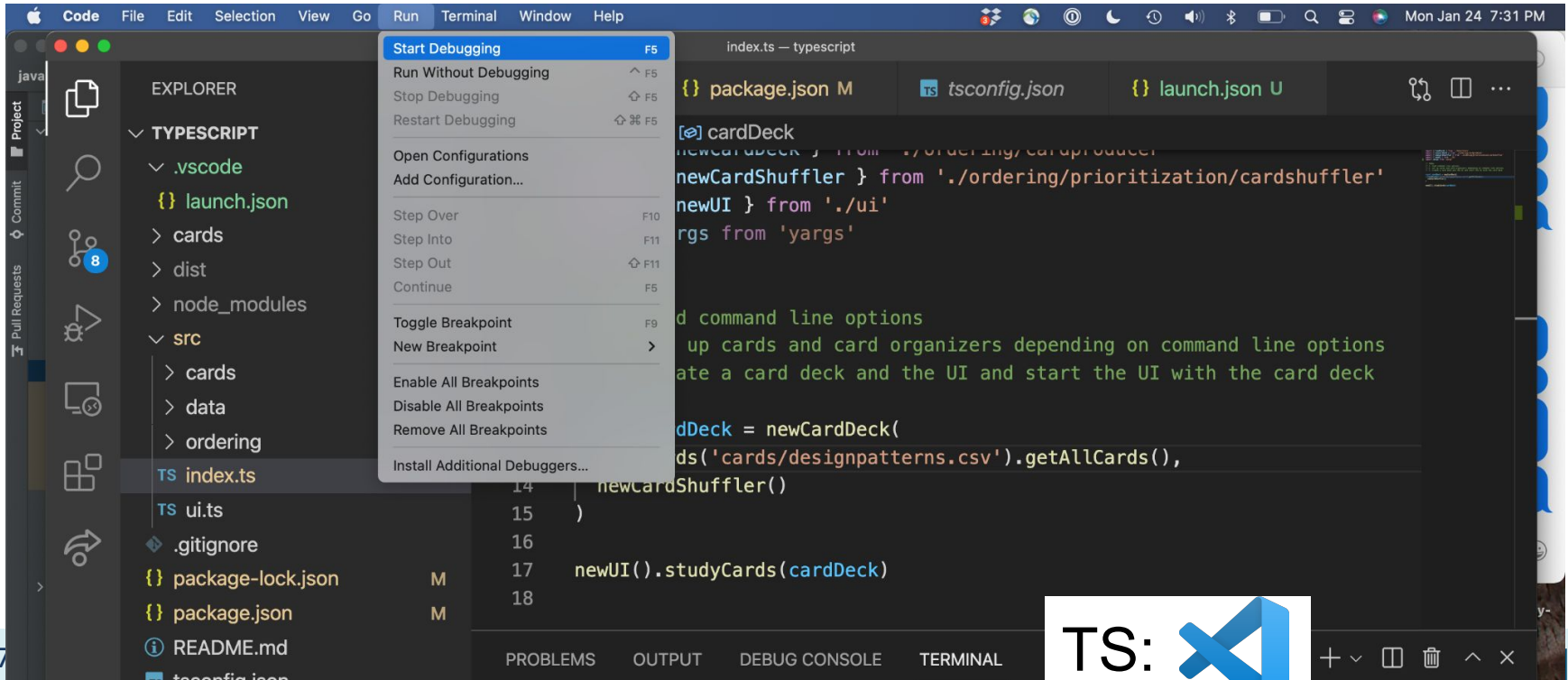
# Under the Hood: IDEs

Automate common programming actions, like debugging, which is often the default mode when you run in the IDE (like in VSCode)



# Under the Hood: IDEs

Debugging allows setting breakpoints in the GUI, access to rich execution info.



Code

File

Edit

Selection

View

Go

Run

Terminal

Window

Help

index.ts — typescript

Launch Program

index.ts

package.json

tsconfig.json

launch.json

VARIABLES

Local

cardDeck: undefined

cardproducer\_1: {newCardDec...

cardshuffler\_1: {newCardShu...

exports: {\_\_esModule: true}

module: Module {id: '...',

WATCH

CALL STACK

La... PAUSED ON BREAKPOINT

<anonymous> src/index.ts

Show 6 More: Skipped by skipFi

LOADED SCRIPTS

BREAKPOINTS

Caught Exceptions

Uncaught Exceptions

index.ts

src

13

index.ts > [0] cardDeck

import { newCardDeck } from './ordering/cardproducer'

import { newCardShuffler } from './ordering/prioritization/cardshuffler'

import { newUI } from './ui'

import yargs from 'yargs'

// TODOs

// 1. load command line options

// 2. set up cards and card organizers depending on command line options

// 3. create a card deck and the UI and start the UI with the card deck

const cardDeck = newCardDeck(

loadCards('cards/designpatterns.csv').getAllCards(),

newCardShuffler()

)

newUI().studyCards(cardDeck)

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

S=—require "/Applications/Visual Studio Code.app/Contents/Resources/app/ex

tensions/ms-vscode.js-debug/src/bootloader.bundle.js" --inspect-publish-uid

=http' 'VSCODE\_INSPECTOR\_OPTIONS={"inspectorIpc":"/var/folders/6y/3yk0l2ms1

yn5t8ng5pd3cdtm0000gn/T/node-cdp.29317-1.sock","deferredMode":false,"waitFo

rDebugger":"","execPath":"/Users/clegoues/.npm/versions/node/v17.3.0/bin/no

de","onlyEntrypoint":false,"autoAttachMode":"always","fileCallback":"/var/f

olders/6y/3yk0l2ms1yn5t8ng5pd3cdtm0000gn/T/node-debug-callback-deb1hd35d68a

bbf0"}' /Users/clegoues/.npm/versions/node/v17.3.

Debugger attached.

+ v ^ x

npm

build - t...

Launch Pro...

17-2

main\*

0 0 0

Launch Program (typescript)

Claire

Live Share

Ln 13, Col 12

Spaces: 2

UTF-8

LF

TypeScript

isr

institute for SOFTWARE RESEARCH

# Under the Hood: IDEs

- IDE designers spend a lot of time automating common development tasks
  - Sometimes they get a little too helpful (modifying pom's)
  - Many plugins provide customized experience
  - Mostly evolve with new tools, prioritizing emerging routines
- Useful to know how these actions work
  - Often not much more than invoking commands for you
    - VSCode, IntelliJ are very explicit about this in the terminal -- great for customization

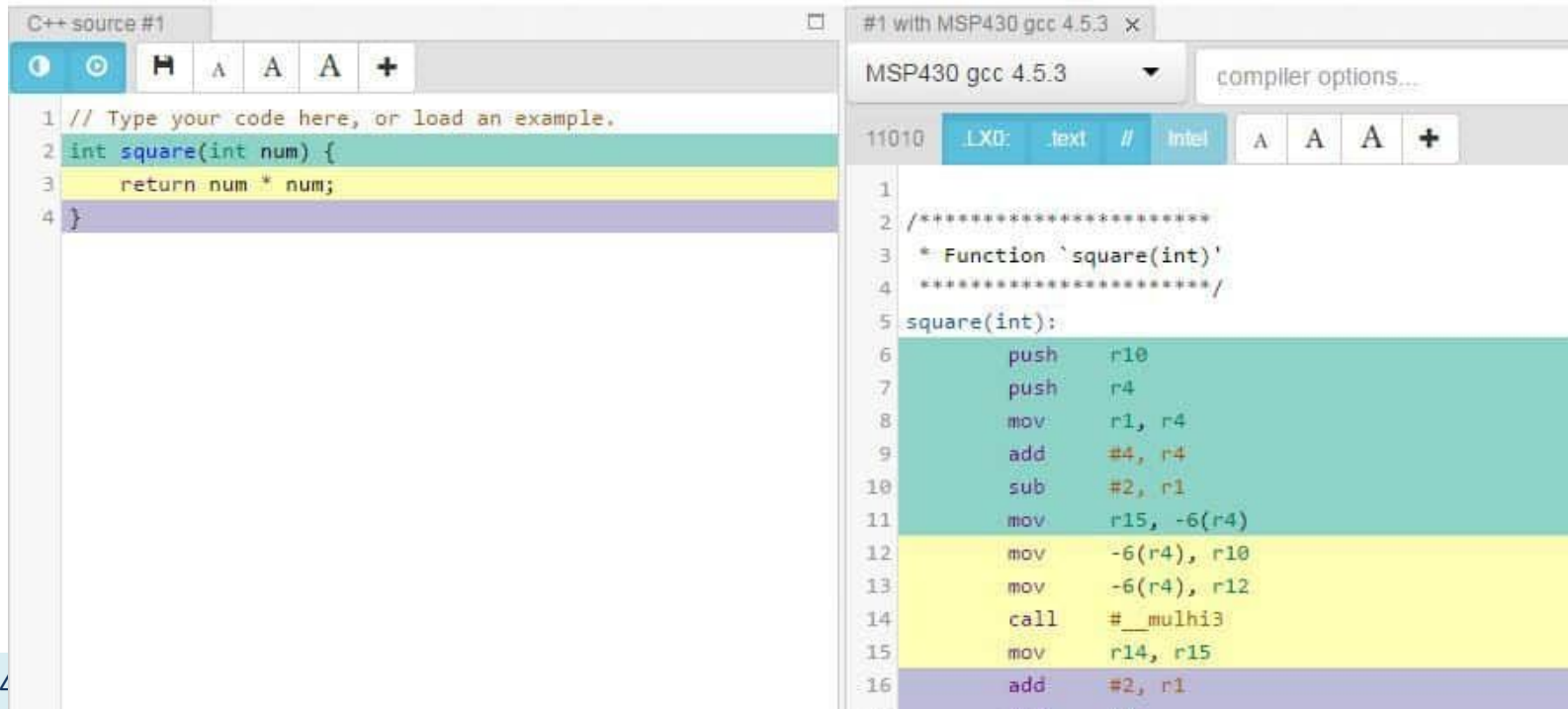
```
"C:\Program Files\Java\jdk-16.0.1\bin\java.exe" -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program Files\JetBrains\IntelliJ ID
Process finished with exit code 0
```

# Abstraction, Reuse, and Programming Tools

- For each in {IDE, **Build systems**, libraries, CI}:
  - **What is it today?**
  - What is under the hood?
- What is next?

# Quick overview of today's toolchain: Build Systems

How does this happen?



The image shows a side-by-side comparison of C++ source code and its assembly output. The left window, titled 'C++ source #1', contains a simple C++ function: `int square(int num) { return num * num; }`. The right window, titled '#1 with MSP430 gcc 4.5.3', shows the assembly generated by the compiler. It includes a header comment for the 'square(int)' function and the corresponding assembly instructions, such as `push r10`, `push r4`, `mov r1, r4`, `add #4, r4`, `sub #2, r1`, `mov r15, -6(r4)`, `mov -6(r4), r10`, `mov -6(r4), r12`, `call #_mulhi3`, `mov r14, r15`, and `add #2, r1`. The assembly code is color-coded to match the source code: teal for the function signature and initial setup, yellow for the main calculation, and purple for the final return statement.

```
C++ source #1
1 // Type your code here, or load an example.
2 int square(int num) {
3     return num * num;
4 }
```

```
#1 with MSP430 gcc 4.5.3
MSP430 gcc 4.5.3
11010 .LXD: .text // Intel
1
2 /*****
3  * Function `square(int)'
4  *****/
5 square(int):
6     push    r10
7     push    r4
8     mov     r1, r4
9     add     #4, r4
10    sub     #2, r1
11    mov     r15, -6(r4)
12    mov     -6(r4), r10
13    mov     -6(r4), r12
14    call    #_mulhi3
15    mov     r14, r15
16    add     #2, r1
```



# Quick overview of today's toolchain: Build Systems

Compiling is “easy” when all your source code is here

Nowadays, your code is not “here”

- Even libraries that you use in the IDE!
- Interfaces make that possible

Study the CI log:

- What is it doing?
- Downloading, compiling, running checks
- Most of this is “building”, using Maven
- More on CI later

```
217 [INFO] .....< org.example:FlashCards >.....
218 [INFO] Building FlashCards 1.0-SNAPSHOT
219 [INFO] .....[ jar ].....
220 [INFO] Downloading from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/plugins/maven-resources-
plugin/2.6/maven-resources-plugin-2.6.pom
221 [INFO] Downloaded from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/plugins/maven-resources-
plugin/2.6/maven-resources-plugin-2.6.pom (8.1 kB at 30 kB/s)
222 [INFO] Downloading from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/plugins/maven-plugins/23/maven-
plugins-23.pom
223 [INFO] Downloaded from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/plugins/maven-plugins/23/maven-
plugins-23.pom (9.2 kB at 788 kB/s)
224 [INFO] Downloading from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/maven-parent/22/maven-parent-
22.pom
225 [INFO] Downloaded from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/maven-parent/22/maven-parent-
22.pom (38 kB at 1.5 MB/s)
226 [INFO] Downloading from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/apache/11/apache-11.pom
227 [INFO] Downloaded from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/apache/11/apache-11.pom (15 kB at 1.2
MB/s)
228 [INFO] Downloading from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/plugins/maven-resources-
plugin/2.6/maven-resources-plugin-2.6.jar
229 [INFO] Downloaded from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/plugins/maven-resources-
plugin/2.6/maven-resources-plugin-2.6.jar (38 kB at 1.6 MB/s)
230 [INFO] Downloading from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/plugins/maven-compiler-
plugin/3.1/maven-compiler-plugin-3.1.pom
231 [INFO] Downloaded from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/plugins/maven-compiler-
plugin/3.1/maven-compiler-plugin-3.1.pom (10 kB at 928 kB/s)
232 [INFO] Downloading from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/plugins/maven-plugins/24/maven-
plugins-24.pom
233 [INFO] Downloaded from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/plugins/maven-plugins/24/maven-
plugins-24.pom (11 kB at 982 kB/s)
234 [INFO] Downloading from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/maven-parent/23/maven-parent-
23.pom
235 [INFO] Downloaded from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/maven-parent/23/maven-parent-
23.pom (33 kB at 1.4 MB/s)
236 [INFO] Downloading from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/apache/13/apache-13.pom
237 [INFO] Downloaded from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/apache/13/apache-13.pom (14 kB at 688
kB/s)
238 [INFO] Downloading from google-maven-central: https://maven-central.storage-download.googleapis.com/maven2/org/apache/maven/plugins/maven-compiler-
```

# Quick overview of today's toolchain: Build Systems

- Has a few basic tasks:
  - Compiling & linking, to produce an executable
  - Creating secondary *artifacts*, e.g. documentation-pages, linter reports, test suite reports
  - Different levels of “depth” may be appropriate, for large code bases (e.g. Google)
- Popular options:
  - For Java: Maven and Gradle -- historically Ant.
    - You could do any homework in either; we're not attached to one
  - For JS/TS: Node(JS)
    - Generally coupled with the Node Package Manager (NPM)
- Often built into IDEs, as plugins

# Abstraction, Reuse, and Programming Tools

- For each in {IDE, **Build systems**, libraries, CI}:
  - What is it today?
  - **What is under the hood?**
- What is next?

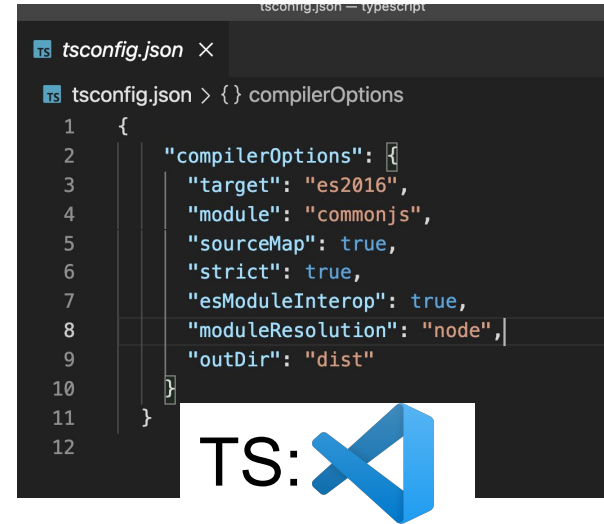
# Under the Hood: Build Systems

- These days: intricately tied with IDEs, package managers
- Projects often come with a build config file or two
  - 'pom.xml' for Maven
  - 'tsconfig.json' + 'package.json' for TypeScript+NPM -- the second deals with packages
  - These can be nested, one per (sub-)directory, to compose larger systems
    - On GitHub, you can create links across repositories

# Under the Hood: Build Systems

Projects often come with a build config file or two

- 'pom.xml' for Maven
- 'tsconfig.json' + 'package.json' for TypeScript+NPM -- the second deals with packages
- Specifies:
  - Compilation source and target version
  - High-level configuration options
  - Targets for various phases in development
    - "lifecycle" in Maven; e.g. 'compile', 'test', 'deploy'
  - Often involving plugins
  - Dependencies with versions





```
m pom.xml (FlashCards)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
```

## Maven Phases

Although hardly a comprehensive list, these are the most common *default* lifecycle phases executed.

- **validate**: validate the project is correct and all necessary information is available
- **compile**: compile the source code of the project
- **test**: test the compiled source code using a suitable unit testing framework. These tests should not require the code
- **package**: take the compiled code and package it in its distributable format, such as a JAR.
- **integration-test**: process and deploy the package if necessary into an environment where integration tests can be
- **verify**: run any checks to verify the package is valid and meets quality criteria
- **install**: install the package into the local repository, for use as a dependency in other projects locally
- **deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing

There are two other Maven lifecycles of note beyond the *default* list above. They are

- **clean**: cleans up artifacts created by prior builds
- **site**: generates site documentation for this project

<https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>

```
31 <version>RELEASE</version>
32 <scope>test</scope>
33 </dependency>
project > dependencies > dependency
Build Dependencies
```



- Node.js is a JS runtime. npm is its package manager.

```
package.json — claire-hw1-js
} package.json 1, M X
{} package.json > {} dependencies
1  {
2    "name": "hw1-flashcards",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    > Debug
7    "scripts": {
8      "compile": "tsc",
9      "lint": "ts-standard",
10     "start": "node dist/index.js"
11   },
12   "author": "",
13   "license": "ISC",
14   "devDependencies": {
15     "@types/node": "^17.0.8",
16     "@types/readline-sync": "^1.4.4",
17     "ts-standard": "^10.0.0",
18     "typescript": "^4.4.2"
19   },
20   "dependencies": {
21     "readline-sync": "^1.4.10",
22   }
23 }
```

EXPLORER

▼ TYPESCRIPT

▼ .vscode

- launch.json
- cards
- dist
- node\_modules
- ▼ src
  - cards
  - data
  - ordering
- index.ts
- ui.ts
- .gitignore
- package-lock.json
- package.json
- README.md
- tsconfig.json

- Start Debugging F5
- Run Without Debugging ^ F5
- Stop Debugging ⇧ F5
- Restart Debugging ⇧ ⌘ F5
- Open Configurations
- Add Configuration...**
- Step Over F10
- Step Into F11
- Step Out ⇧ F11
- Continue F5
- Toggle Breakpoint F9
- New Breakpoint ➤
- Enable All Breakpoints
- Disable All Breakpoints
- Remove All Breakpoints
- Install Additional Debuggers...

launch.json — typescript

package.json tsconfig.json launch.json U X

Launch Targets > {} Launch Program

IntelliSense to learn about possible attributes.  
ver to view descriptions of existing attributes.  
r more information, visit: <https://go.microsoft.com/fwlink/?linkid=...>


```
"version": "0.2.0",  
"configurations": [  
  
    {  
      "type": "pwa-node",  
      "request": "launch",  
      "name": "Launch Program",  
      "skipFiles": [  
        "<node_internals>/**"  
      ],  
      "program": "${file}",  
      "console": "integratedTerminal",  
      "preLaunchTask": "tsc: build - tsconfig.json",  
    }  
  ]  
}
```

Add Configuration...

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

5t8ng5pd3cdtm0000gn/T/node-cdp.27252-4.sock", "deferredMode": false, "waitForDebugger": "", "execPath": "/Users/clegoues/.nvm/versions/node/v17.3.0/bin/node", "onlyEntrypoint": false, "autoAttachMode": "always", "fileCallback": "/var/folders/6y/3yk0l2ms1yn5t8ng5pd3cdtm0000gn/T/node-debug-callback-b1bc858d1e8e06c9"}' /Users/clegoues/.nvm/versions/node/v17.3.0/bin/node ./dist/index.js  
Debugger attached.  
7 cards to go...

npm  
build - t... ✓  
Launch Pro...

TS: 





`{}` launch.json U X

.vscode > `{}` launch.json > Launch Targets > `{}` Launch Program

```
1  {
2      // Use IntelliSense to learn about
3      // Hover to view descriptions of e
4      // For more information, visit: ht
5      "version": "0.2.0",
6      "configurations": [
7          {
8              "type": "pwa-node",
9              "request": "launch",
10             "name": "Launch Program",
11             "skipFiles": [
12                 "<node_internals>/**"
13             ],
14             "program": "${file}",
15             "console": "integratedTerminal",
16             "preLaunchTask": "tsc: build - tsconfig.json",
17             "args": [ "--help" ],
18             "outFiles": [
19                 "${workspaceFolder}/dist/**/*.js"
20             ]
21         }
22     ]
23 }
24 }
```

```
node/v17.3.0/bin/node ./dist/index.js --help
Debugger attached.
zsh: killed /usr/bin/env /Users/clegoues/.nvm/versions/node/v17.3.0/bin/n
ode --help
(base) clegoues@clegoues-macbook-air typescript % npm run start -- --help[]
ram (typescript)  Claire 🔒 Live Share Ln 13, Col 12 Spaces: 2 UTF-8 LF {} TypeScript
```

TS:





IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help

Mon Jan 24 5:17 PM

Project Structure

- Project
- RecentMistakes
- repetition
- FlashCards
- FlashCard
- org.exar

Run/Debug Configurations

Name: Main

Build and run

java 17 SDK of 'claire-hw1-jav' edu.cmu.cs214.hw1.Main

Program arguments

CLI arguments to your application. \CR

Working directory: /Users/legoues/courses/17-214/claire-hw1-java

Environment variables:

Open run/debug tool window when started

Code Coverage

Packages and classes to include in coverage data

- edu.cmu.cs214.hw1.\*

Cancel Apply OK

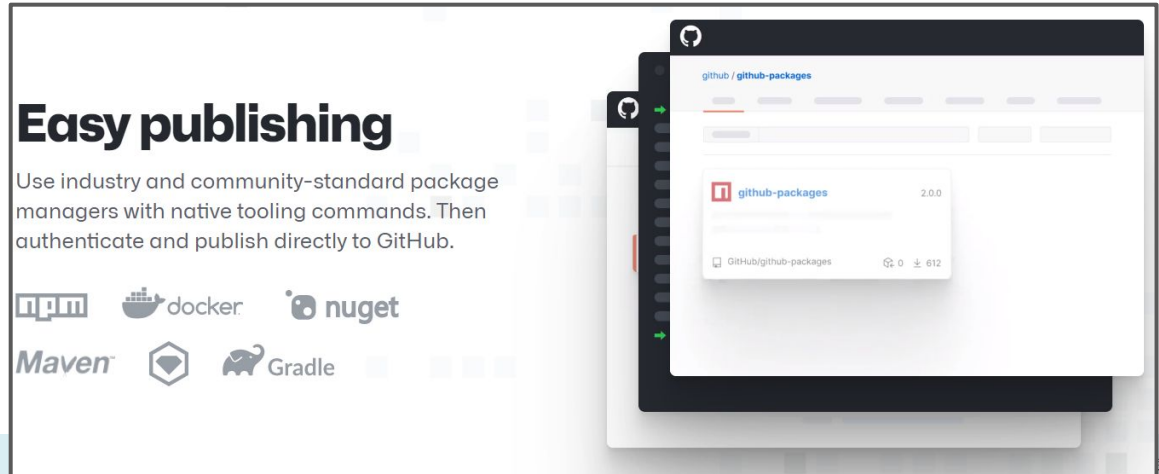
Java: **Maven**<sup>TM</sup>

# Abstraction, Reuse, and Programming Tools

- For each in {IDE, Build systems, **libraries**, CI}:
  - **What is it today?**
  - What is under the hood?
- What is next?

# Quick overview of today's toolchain: Libraries

- Myriad. Publicly hosted on various *package managers*
  - Often tied, but not inextricably linked, to build tools, and languages
  - Maven/Gradle for Java, NPM for JS/TS, Nuget for C#, ...
  - Registries of managers, e.g., GitHub Packages



IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help

java - pom.xml (FlashCards)

Project: java [FlashCards] ~/courses/17-214/hw-1-flashcards

- cards
- src
- target
- .gitignore
- checkstyle.xml
- m pom.xml**
- README.md
- External Libraries
- Scratches and Consoles

Run: m FlashCards [site] x

FlashCards [site]: At 1/24/22, 5:35 6 sec, 971 ms  
g.example:FlashCards:jar:1.0-1 4 sec, 943 ms  
site 1 error, 3 warnings 4 sec, 774 ms

- Input file encoding has not been set, using default
- No project URL defined - decoration link
- File encoding has not been set, using platform default
- Main.java src/main/java/edu/cmu/cs/214/flashcards/Main.java
- Comment matches to-do format 'TODO:'. [TodoComment]

Dependency

```
</build>
<dependencies>
  <dependency>
    <groupId>
    <artifactId>
    <version>
    <scope>
  </dependency>
</dependencies>
<reporting>
<plugins>
  <plugin>
    <groupId>
    <artifactId>
    <version>
    <configuration>
      <configuration>
    </configuration>
  </plugin>
</plugins>
```

Maven

- FlashCards
  - Lifecycle
    - clean
    - validate
    - compile
    - test
    - package
    - verify
    - install
    - site**
    - deploy
  - Plugins
  - Dependencies

Java: **Maven**™





# Quick overview of today's toolchain: Libraries

- Myriad. Publicly hosted on various *managers*
  - Often tied, but not inextricably linked, to build tools, and languages
  - Maven, Gradle, NPM, Nuget, Docker, ...
  - Registries of managers, e.g., GitHub Packages
- Releases are generally fast-paced or frigid
  - Almost all volunteer-based, so support waivers, as does documentation quality
  - Often open-source, so you can check out the status & details on GitHub
  - Beware of vulnerabilities and bugs, esp. with minor-releases and nightly's, old packages



NPM package with 3 million weekly downloads had a severe vulnerability



# Quick overview of today's toolchain: Libraries

- A Case-Study:

- 'pac-resolver' (3M weekly downloads) has a major security vulnerability
  - Uses 'degenerator' (same author), which misuses a Node module
  - [“The vm module is not a security mechanism. Do not use it to run untrusted code.”](#)
  - (a mistake that's been made before: people rarely read disclaimers)
- 'pac-proxy-agent' (**2M weekly** downloads, same author) uses the above
  - Is widely popular, the main reason people use 'degenerator'
  - Most people using this package have never heard of the latter -- many never will



## Log4j software bug: What you need to know

Casual computer users have probably logging software, but it's used across



Bree Fowler

Dec. 21, 2021 9:01 a.m. PT



Getty Images

<https://threatpost.com> › Vulnerabilities

### Third Log4J Bug Can Trigger DoS; Apache Issues Patch

Dec 20, 2021 — The latest **bug** isn't a variant of the Log4Shell remote-code execution (RCE) **bug** that's plagued IT teams since Dec. 10, coming under active ...

<https://www.scmagazine.com> › Application security

### Log4j, again, needs patching as new bug is found and ...

Dec 28, 2021 — Researchers at Checkmarx discovered a way to use **Log4j** to launch malicious code, forcing yet another round of patching for affected users.

# Abstraction, Reuse, and Programming Tools

- For each in {IDE, Build systems, **libraries**, CI}:
  - What is it today?
  - **What is under the hood?**
- What is next?

# Under the Hood: Libraries & Frameworks

Packages can be either:

- Libraries:
  - A set of classes and methods that provide reusable functionality
  - Typically: programmer calls, library returns data, that's it.

# Under the Hood: Libraries & Frameworks

Packages can be either:

- Libraries:
  - A set of classes and methods that provide reusable functionality
  - Typically: programmer calls, library returns data, that's it.
- Frameworks:
  - Reusable skeleton code that can be customized into an application
  - Framework calls back into client code
    - The Hollywood principle: "Don't call us. We'll call you."
  - E.g., Android development: you declare your UI elements, activities to be composed
  - Principle: [inversion of control](#)

# Under the Hood: Libraries & Frameworks

Packages can be either:

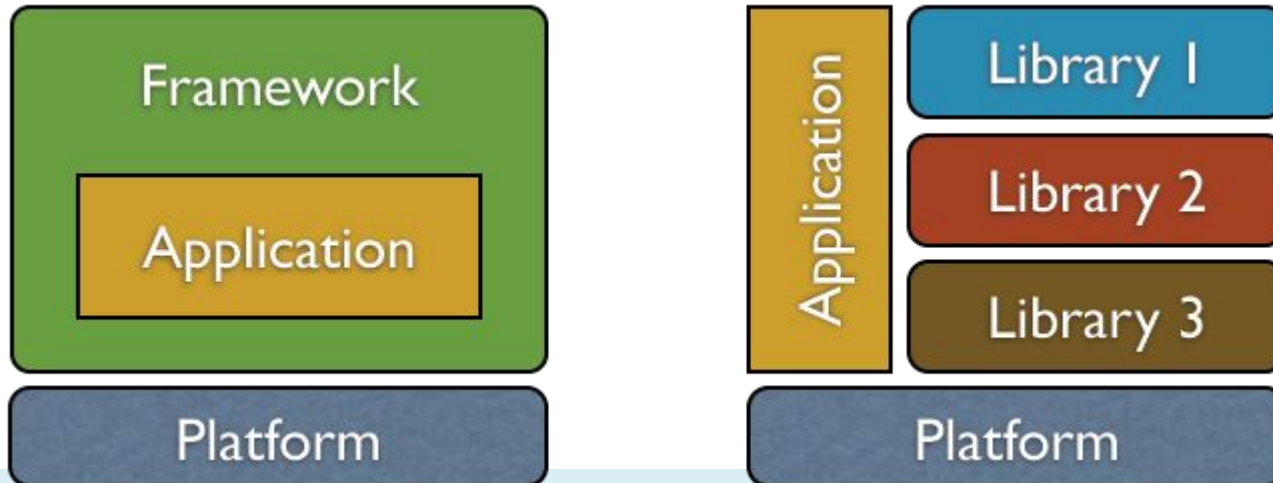
- Libraries:
  - A set of classes and methods that provide reusable functionality
  - Typically: programmer calls, library returns data, that's it.
- Frameworks:
  - Reusable skeleton code that can be customized into an application
  - Framework calls back into client code
    - The Hollywood principle: "Don't call us. We'll call you."
  - E.g., Android development: you declare your UI elements, activities to be composed
  - Principle: [inversion of control](#)
- You typically use zero/one framework and many libraries
  - Frameworks might be especially constraining, but for good reason.
  - Some tools are a bit of both, and not all frameworks quite invert control

# Under the Hood: Libraries & Frameworks

Which kind is a command-line parsing package?

Which kind is Android?

How about a tool that runs tests based on annotations you add in your code?



# Under the Hood: Libraries & Frameworks

Look into:

- **Stated Goal:**
  - A simple interface (“get started in one line!”) also means lots of abstraction
  - That’s neither good nor bad; know what you need
  - Docs with “advanced use cases” are always neat
- **Maintenance:**
  - Active release cycle, recent updates to documentation
  - GitHub build status, issue tracker (filled with unmerged ‘dependabot’ PRs?)
  - Lots of companies deliberately lag by one minor (or even major) version
- **Recursive dependencies**
  - Myriad, beyond inspection. Using OSS in corporate environments is a headache



# Abstraction, Reuse, and Programming Tools

- For each in {IDE, Build systems, libraries, **CI**):
  - **What is it today?**
  - What is under the hood?
- What is next?

# Quick overview of today's toolchain: Continuous Integration

CI: Automates standard build, test, deploy pipelines

(Technically, the latter is “CD”)

Typically builds from scratch in a clean *container*

Often tied to code-review; triggers on new commits, pull requests

- Ideally, official releases pass the build

Produces (long) logs with debugging outputs

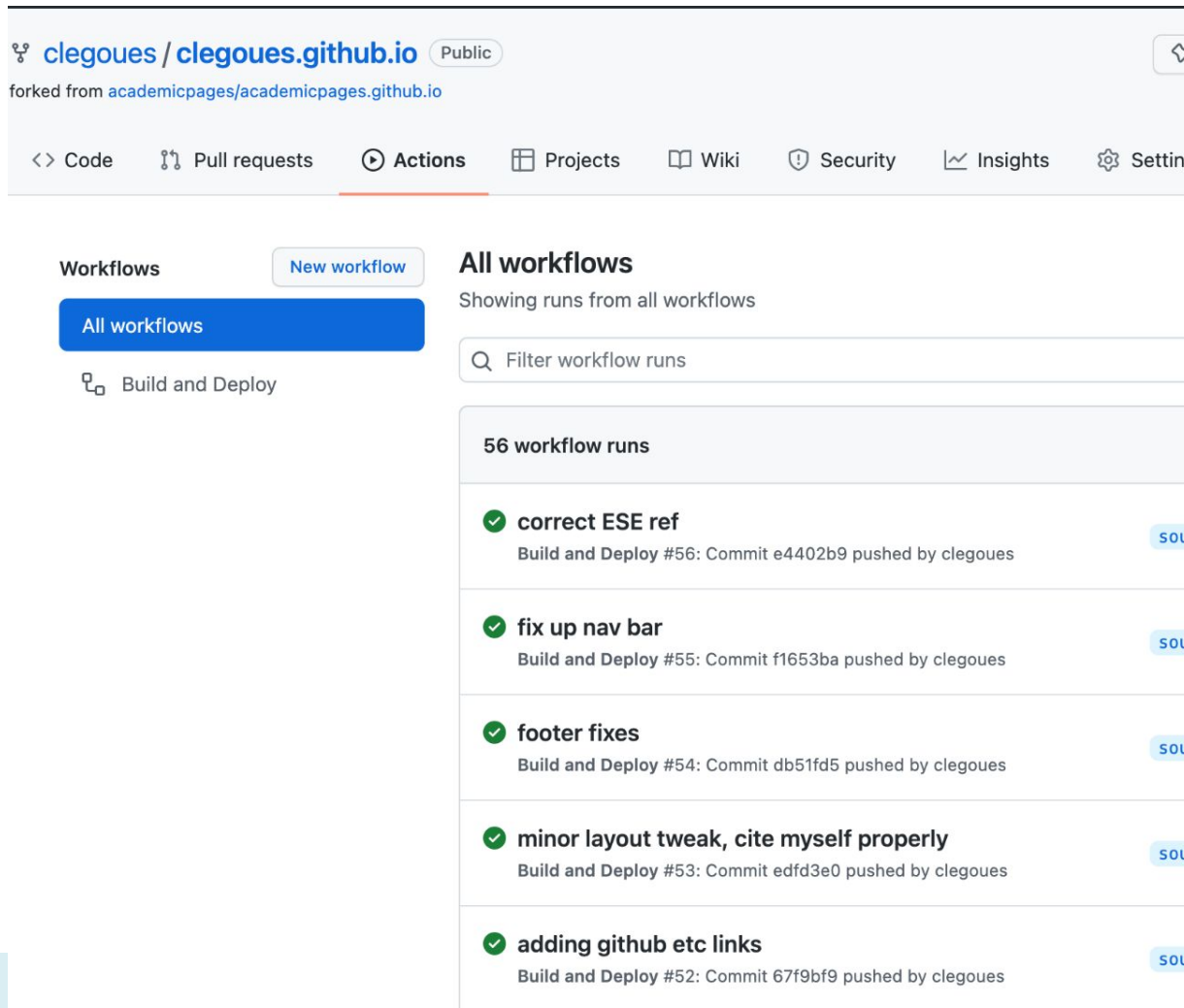
# Under the Hood: Continuous Integration

Defines a series of actions to be run in a clean build:

- Actions start from the very top:
  - Clone repository, checkout branch
  - Download & install Java/Node
  - Invoke commands with timeouts
- Travis allocates a new (Docker) container for each build
  - Think of this like a fresh, temporary computer
  - Usually with a few default libraries present (i.e., based on an *image*)
- That means: **fully replicable builds**

# Continuous integration – GitHub Actions

You can see the results of builds over time



**clegoues / clegoues.github.io** Public

forked from academicpages/academicpages.github.io

< > Code Pull requests **Actions** Projects Wiki Security Insights Settings

**Workflows** [New workflow](#)

**All workflows**

Showing runs from all workflows

Filter workflow runs

**56 workflow runs**

- ✓ **correct ESE ref**  
Build and Deploy #56: Commit e4402b9 pushed by clegoues
- ✓ **fix up nav bar**  
Build and Deploy #55: Commit f1653ba pushed by clegoues
- ✓ **footer fixes**  
Build and Deploy #54: Commit db51fd5 pushed by clegoues
- ✓ **minor layout tweak, cite myself properly**  
Build and Deploy #53: Commit edfd3e0 pushed by clegoues
- ✓ **adding github etc links**  
Build and Deploy #52: Commit 67f9bf9 pushed by clegoues

```

▶ 163 Installing SSH key from: default repository key
164 Using /home/travis/.netrc to clone repository.
165
166
▼ 167 $ git clone --depth=50 --branch=TypeScript https://github.com/CMU-17-214/template-21f-hw1.git CMU-17-214/template-21f-hw1 1.75s
168 Cloning into 'CMU-17-214/template-21f-hw1'...
169 remote: Enumerating objects: 117, done.
170 remote: Counting objects: 100% (117/117), done.
171 remote: Compressing objects: 100% (73/73), done.
172 remote: Total 117 (delta 50), reused 104 (delta 37), pack-reused 0
173 Receiving objects: 100% (117/117), 69.89 KiB | 2.25 MiB/s, done.
174 Resolving deltas: 100% (50/50), done.
175 $ cd CMU-17-214/template-21f-hw1
176 $ git checkout -qf 0d657225c8cbdd52751c2f88527f93f4099b041e
177
178 0.01s
▼ 179 $ nvm install 16 3.65s
180 Downloading and installing node v16.8.0...
181 Downloading https://nodejs.org/dist/v16.8.0/node-v16.8.0-linux-x64.tar.xz...
182 Computing checksum with sha256sum
183 Checksums matched!
184 Now using node v16.8.0 (npm v7.21.0)
185
▶ 186 Setting up build cache cache.1
187
188
▶ 189 cache.npm
190
191 $ node --version
192 v16.8.0
193 $ npm --version
194 7.21.0
195 $ nvm --version
196 0.38.0
197
201
▶ 202 $ npm ci 3.04s
203
204
205 $ timeout 5m npm run compile 4.93s
206
207 > hw1-flashcards@1.0.0 compile
208 > tsc

```

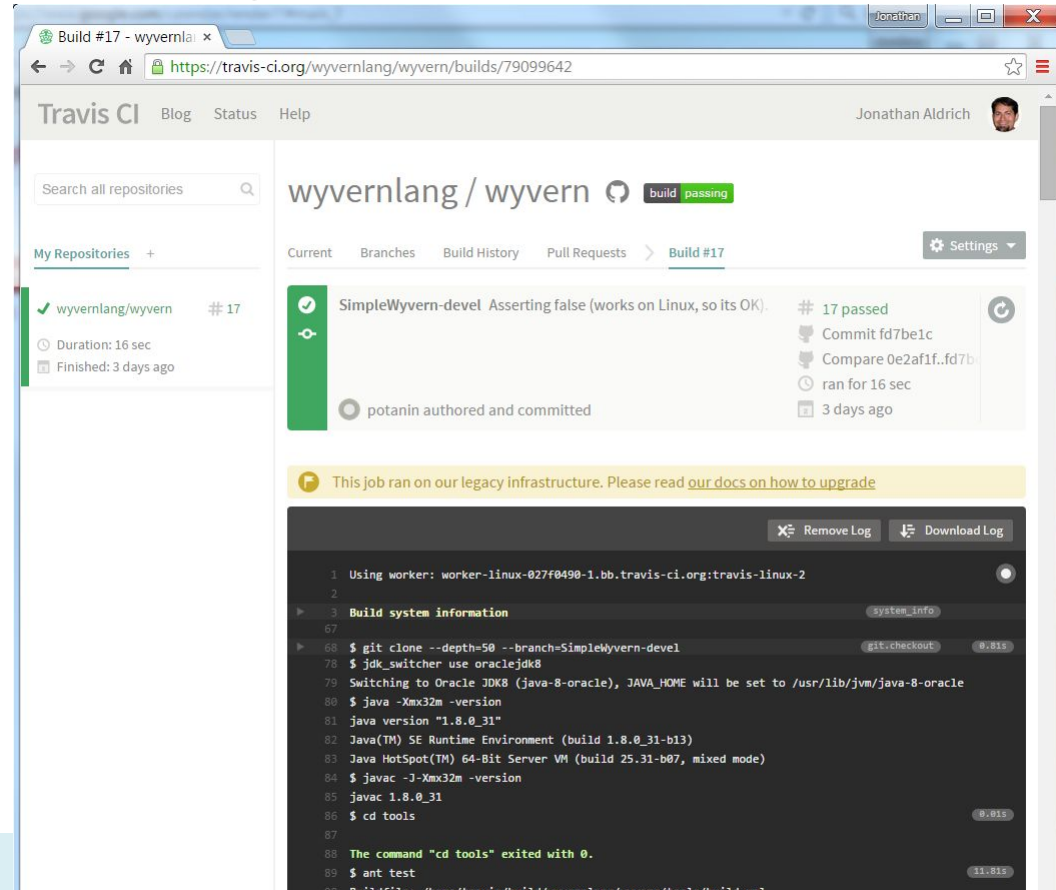
# Under the Hood: Continuous Integration

Automatically builds, tests, and displays the result

We – and everyone else – used to use Travis CI.

- Until they randomly stopped supporting OSS.

GitHub now has native CI support, and it's pretty good: GitHub Actions.



# Quick overview of today's toolchain: not mentioned

Docker: containerize applications for coarse-grained reuse

Cloud: deploy and scale rapidly, release seamlessly

Bug/Issue trackers, often integrated with reviews

# Behind the Abstraction: Some Nuance

- Automation vs. Reuse
  - We tend to automate common chains of actions
    - Gear-up := {Press clutch, switch gear, release clutch while accelerating}
  - To facilitate reusing such “subroutines”, we introduce abstractions
    - Accelerate in ‘D’ => Gear-up when needed
- Reuse vs. Interfaces
  - Interfaces facilitate reuse through abstraction
    - Allow upgrading implementation without breaking things
    - Provide explicit & transparent contract

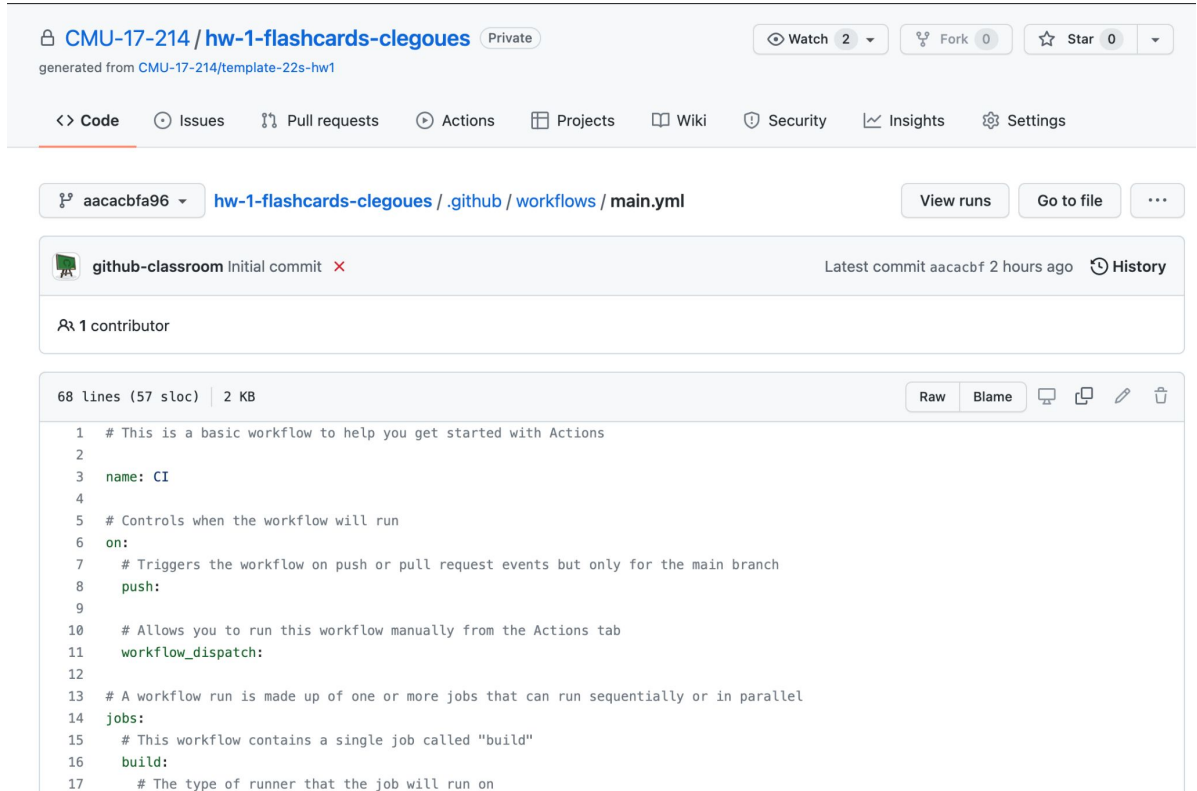




# Behind the Abstraction, Some Nuance

Most tools are abstractions of common commands

- Typically operated via GUI and/or a DSL
- Obvious for GitHub Actions: just read the Yaml
  - Script-like languages are common
  - Involving a vocabulary of “targets”
  - E.g., `mvn site`



The screenshot shows a GitHub repository for `CMU-17-214 / hw-1-flashcards-clegoues`, which is a private repository generated from `CMU-17-214/template-22s-hw1`. The repository has 2 watchers, 0 forks, and 0 stars. The navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The selected file is `hw-1-flashcards-clegoues / .github / workflows / main.yml`, with a commit hash of `aacacbf96`. The file was last committed by `github-classroom` 2 hours ago. The file content is a YAML workflow definition for a basic CI pipeline. It includes a `name` field set to `CI`, an `on` trigger for `push` events on the `main` branch, and a `jobs` section with a single job named `build` that runs on the `ubuntu-latest` runner.

```
68 lines (57 sloc) | 2 KB
Raw Blame

1 # This is a basic workflow to help you get started with Actions
2
3 name: CI
4
5 # Controls when the workflow will run
6 on:
7   # Triggers the workflow on push or pull request events but only for the main branch
8   push:
9
10  # Allows you to run this workflow manually from the Actions tab
11  workflow_dispatch:
12
13 # A workflow run is made up of one or more jobs that can run sequentially or in parallel
14 jobs:
15   # This workflow contains a single job called "build"
16   build:
17     # The type of runner that the job will run on
```

# Behind the Abstraction, Some Nuance

Most tools are abstractions of common commands

- Typically operated via GUI and/or a DSL
- Obvious for GitHub Actions: just read the Yaml
  - Script-like languages are common
  - Involving a vocabulary of “targets”
  - E.g., ``mvn site``

Abstraction can also “trap” us

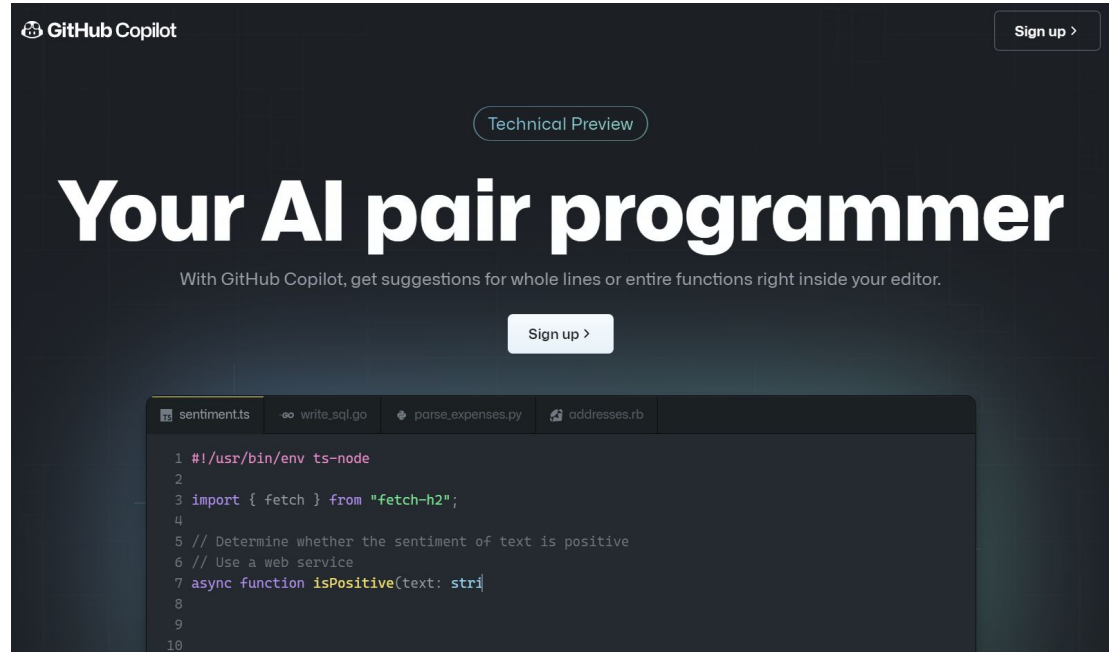
- When/how do we leave the abstraction?
- Command-line comes built into IDEs for a reason
- Non-trivial in general! May require switching/“patching” libraries
  - E.g., Maven → Gradle for more unusual build routines

# Abstraction, Reuse, and Programming Tools

- For each in {IDE, Build systems, libraries, CI}:
  - What is it today?
  - What is under the hood?
- **What is next?** (*any guesses?*)

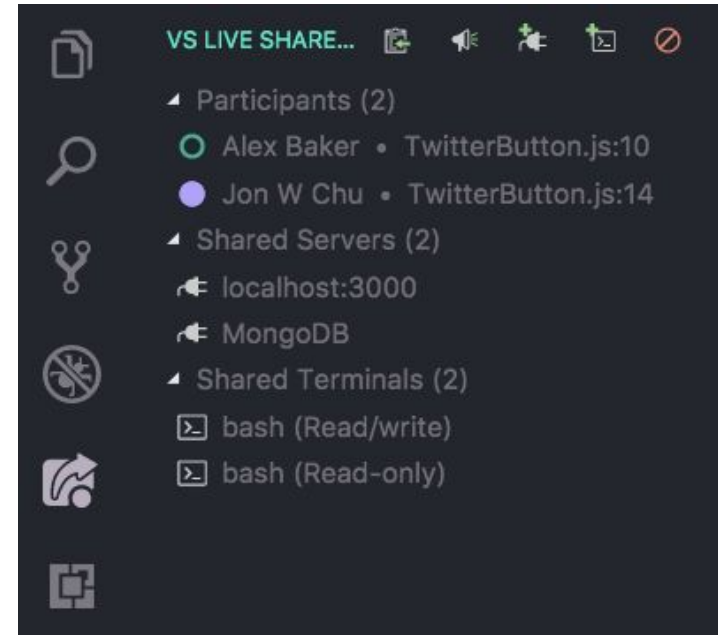
# What's Next: AI Powered Programming

- Easier in Web IDEs
  - Which are themselves “next”



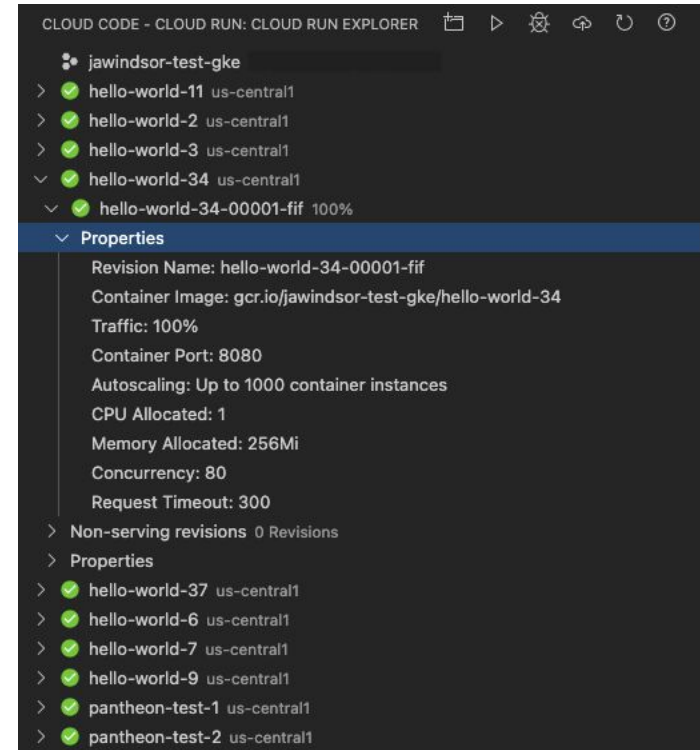
# What's Next: Collaborative online coding

- Think: Google Docs for code
- E.g. VS Live Share
- How will this change “commits”?



# What's Next: Tighter IDE-to-cloud integration

- Google Cloud is pushing on this with VSCode
- We will (lightly) touch on Containers & Clouds in this course



# Summary

- Programming Tools are abundant, and rapidly evolving
  - Learn multiple; you will have to inevitably
- They rely on abstractions through interfaces to facilitate reuse
  - Which come in many shapes: GUI, API, DSL
  - And can be a limitation -- choose wisely
- Your HW1 toolchain sets you up for all homeworks
  - With modest variations (frameworks, new build targets)
  - Self-discovery is a big asset
  - Recitation should be helpful!