

Software Evolution and Maintenance

Asangi Jayatilaka

Week #11: Lecture – part 4

Topics

- **Part 4**
 - **CI/CD**
 - **GitHub workflows**

What is CI/CD in simple terms

- **CI/CD = Continuous Integration + Continuous Deployment/Delivery**
 - **Continuous integration (CI):** Automatically builds, tests, and integrates code changes within a shared repository
 - **Continuous delivery (CD):** Automatically delivers code changes to production-ready environments for approval
 - **Continuous deployment (CD):** Automatically deploys code changes to customers directly

Why CI/CD

- Increase development speed
- Stability and reliability
- Better collaboration
- Innovation and growth for business

Some popular CI/CD tools

- Jenkins
- GitHub Actions
- AWS Code Pipeline
- Etc..



Using built automation tools with CI/CD

- To make it easier to use CI/CD we should consider incorporating built automation tool in our project.
- Build automation tools to support
 - Download dependencies and transitive dependencies automatically
 - Build, test, and package code using a standard lifecycle
 - Integrate with IDEs, CI/CD pipelines

Why use Maven (or a similar tool) as a build automation tool?

- **Dependency Management**

- Declarative configuration: You declare dependencies (and versions) in a **pom.xml**, and Maven resolves them automatically, including transitive dependencies.
- Transitive dependency resolution: If Library A depends on Library B, Maven pulls both

- **Standardised Project Structure**

- There is a standard directory structure that Maven expects to build a project:

- **<root>**

- **POM.xml**

- **src**

- **main**

- **java**

holds program/application code

- **test**

- **java**

holds the unit test code

<https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>

Why use Maven (or a similar tool) as a build automation tool? ...

- **Build Lifecycle Automation**

- Maven provides predefined build phases:
 - validate → compile → test → package → install → deploy
- Example command: **mvn test**

- **Easy Collaboration**

- pom.xml shares everything your team needs: dependencies, plugins, configurations
- Others can clone your project and build it right away

- **CI/CD Friendly**

- Works seamlessly with Jenkins, GitHub Actions, GitLab CI, etc.

Structure of a Basic pom.xml

- **<project>** Root tag
- **<modelVersion>**: POM model version (usually 4.0.0)
- **<groupId>**: Project group. Convention is to use the reverse domain name (e.g., com.company)
- **<artifactId>**: Project/module name
- **<version>**: Current version of the project
- **<properties>**: Maven properties are value placeholders. Their values are accessible anywhere within a POM by using the notation **`${X}`**, where X is the property.
- **<dependencies>**: External libraries used
- **<plugins>**: Any plugins that are needed

Example of a pom.xml file

Root tag and Model version

Project naming and version

Property declarations

Project dependencies

Plugins

```

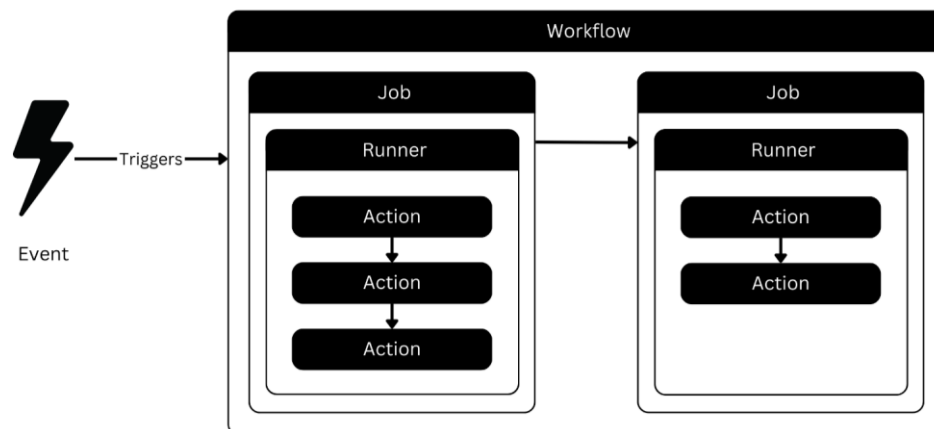
pom.xml
You, 13 seconds ago | 1 author (You)
1 <?xml version="1.0" encoding="UTF-8"?> You, 2 weeks ago • Added Maven support to ease running tests
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>au.edu.rmit.sct</groupId> <!-- Convention is the reverse domain name -->
8     <artifactId>numbers</artifactId> <!-- Name of the project -->
9     <version>0.1.0-SNAPSHOT</version> <!-- Version of the project -->
10
11     <properties>
12         <!-- Java version -->
13         <maven.compiler.release>21</maven.compiler.release>
14         <junit.version>5.12.2</junit.version>
15     </properties>
16
17     <dependencies>
18         <!-- JUnit 5 Dependency -->
19         <dependency>
20             <groupId>org.junit.jupiter</groupId>
21             <artifactId>junit-jupiter-api</artifactId>
22             <version>${junit.version}</version>
23             <scope>test</scope> <!-- Test scope means it is only used for testing -->
24         </dependency>
25         <dependency>
26             <groupId>org.junit.jupiter</groupId>
27             <artifactId>junit-jupiter-engine</artifactId>
28             <version>${junit.version}</version>
29             <scope>test</scope> <!-- Test scope means it is only used for testing -->
30         </dependency>
31     </dependencies>
32
33     <build>
34         <plugins>
35             <!-- Lets use the latest version of the compiler plugin
36             https://maven.apache.org/plugins/maven-compiler-plugin/index.html
37             -->
38             <plugin>
39                 <groupId>org.apache.maven.plugins</groupId>
40                 <artifactId>maven-compiler-plugin</artifactId>
41                 <version>3.14.0</version>
42             </plugin>
43             <plugin>
44                 <!-- We need the maven surefire plugin to run the unit tests
45                 https://maven.apache.org/surefire/maven-surefire-plugin/index.html
46                 -->
47                 <groupId>org.apache.maven.plugins</groupId>
48                 <artifactId>maven-surefire-plugin</artifactId>
49                 <version>3.5.3</version>
50             </plugin>
51         </plugins>
52     </build>
53
54 </project>
  
```

More details about pom.xml file

- <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>
- <https://maven.apache.org/pom.html>

What is GitHub Actions

- GitHub Actions give you the flexibility to build automated software development lifecycle workflows.
- You can write individual tasks, called actions, and combine them to create custom workflows in your repository
- CI/CD is one of the workflows that can be automated through GitHub Actions
 - You can configure a GitHub Actions workflow to be triggered when an event occurs in your repository e.g., **Trigger the workflow on push events**
 - Each job will run inside its own virtual machine runner, or inside a container, and has one or more steps that either run a script that you define or run an action



GitHub actions workflow syntax

- Workflow files use YAML syntax, and must have either a .yml or .yaml file extension.
- You must store workflow files in the `.github/workflows/` directory of your repository.

```
1 name: GitHub Actions Java test workflow # Workflow name
2 run-name: Running tests on github actions
3 on: [push] # Trigger the workflow on push events
4 jobs:
5   java-test: # Job name
6     runs-on: ubuntu-latest # Configures the job to run on the latest version of an Ubuntu Linux runner.
7     steps:
8       - name: Check out repository code
9         uses: actions/checkout@v4
10        # The uses keyword specifies that this step will run v4 of the actions/checkout action.
11        # This is an action that checks out your repository onto the runner,
12        # allowing you to run scripts or other actions against your code (such as build and test tools).
13        # You should use the checkout action any time your workflow will use the repository's code.
14
15       - name: Setup java # Install Java on the runner
16         uses: actions/setup-java@v4
17         with:
18           distribution: 'adopt'
19           # The distribution of Java to install. In this case, it specifies the AdoptOpenJDK distribution.
20           # See https://github.com/actions/setup-java?tab=readme-ov-file#supported-distributions for supported distributions.
21           java-version: '21' # The version of Java to install in the job runner.
22
23       - name: Run tests
24         run: mvn test
25        # This step runs the command mvn test, which is a Maven command to run the tests in your project.
26        # Maven is a build automation tool used primarily for Java projects.
27        # The tests are defined in your project's source code and will be executed by Maven.
28
29   another-job:
30     runs-on: ubuntu-latest
31     steps:
32       - name: Check out repository code
33         run: ls -la
```

GitHub actions workflow syntax ...

- Workflow name:
 - The name of your workflow will be displayed on your repository's actions page

```
1 name: GitHub Actions Java test workflow # Workflow name
```

- On Event
 - The type of event that triggers the workflow

```
on: [push] # Trigger the workflow on push events
```

GitHub actions workflow syntax ...

- Jobs Collection

- A workflow run is made up of one or more jobs identified by a unique job_id (java-test).
- If you have defined multiple jobs then they run in parallel by default
- Each job runs in a fresh instance of the virtual environment specified by runs-on.

```
4 jobs:
5   java-test: # Job name
6     runs-on: ubuntu-latest # Configures the job to run on the latest version of an Ubuntu Linux runner.
```

GitHub actions workflow syntax ...

- Job steps
 - A job contains a sequence of tasks called steps.
 - *Step name*: Specify the label to be displayed for this step in GitHub. It's not required but does improve readability in the logs.
 - *uses*: Specify a pre-built action to run as part of a step in your job. These actions are defined elsewhere and can be shared across multiple workflows. For example, checkout action below is provided by GitHub itself.
 - *with*: A map of the input parameters defined by the action.
 - *run*: Runs command-line programs using the operating system's shell

```
7  steps:
8    - name: Check out repository code
9      uses: actions/checkout@v4
10     # The uses keyword specifies that this step will run v4 of the actions/checkout action.
11     # This is an action that checks out your repository onto the runner,
12     # allowing you to run scripts or other actions against your code (such as build and test tools).
13     # You should use the checkout action any time your workflow will use the repository's code.
14
15    - name: Setup java # Install Java on the runner
16      uses: actions/setup-java@v4
17      with:
18        distribution: 'adopt'
19        # The distribution of Java to install. In this case, it specifies the AdoptOpenJDK distribution.
20        # See https://github.com/actions/setup-java?tab=readme-ov-file#supported-distributions for supported distributions.
21        java-version: '21' # The version of Java to install in the job runner.
22
23    - name: Run tests
24      run: mvn test
25      # This step runs the command mvn test, which is a Maven command to run the tests in your project.
26      # Maven is a build automation tool used primarily for Java projects.
27      # The tests are defined in your project's source code and will be executed by Maven.
28
```

<https://github.com/actions/>

Workflow syntax ...

- More information about workflow syntax can be found here:
 - <https://github.github.io/actions-cheat-sheet/actions-cheat-sheet.pdf>

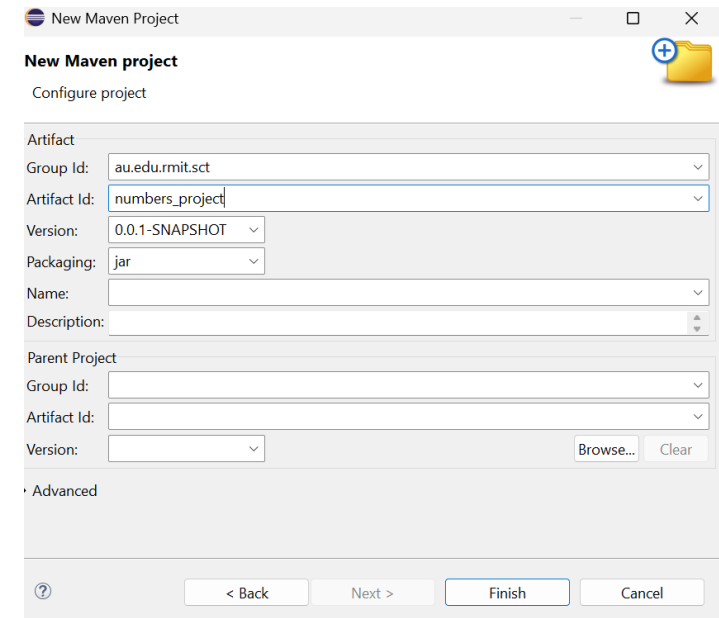
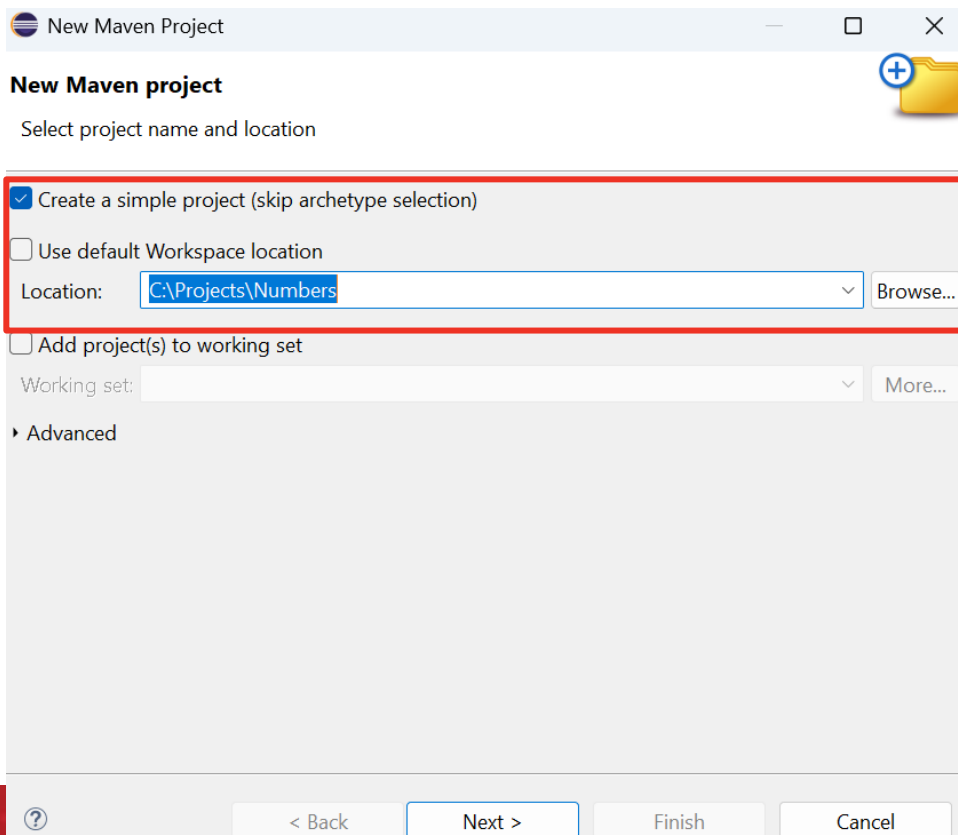
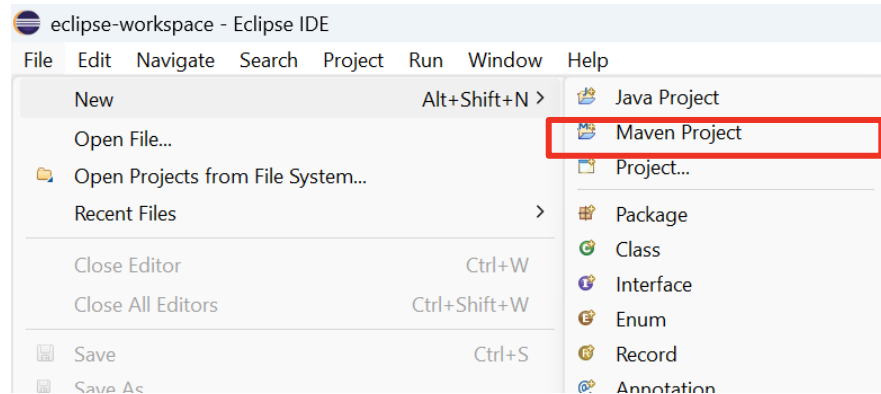
Steps to follow

- Create a Maven-based Java project
- Write a Number class that could
 - Give the sum of two given numbers ($i+j$)
 - Multiply two given numbers ($i*j$)
 - Subtract numbers: Subtract the second number from the first number ($i-j$)
 - Divide numbers: Divide the first number by the second number (i/j)
- Write Unit tests using JUnit
- Add GitHub Actions workflow to run tests once code is pushed to GitHub
- Link your project with the remote GitHub repository
- Demonstrate that the GitHub Actions is working as expected

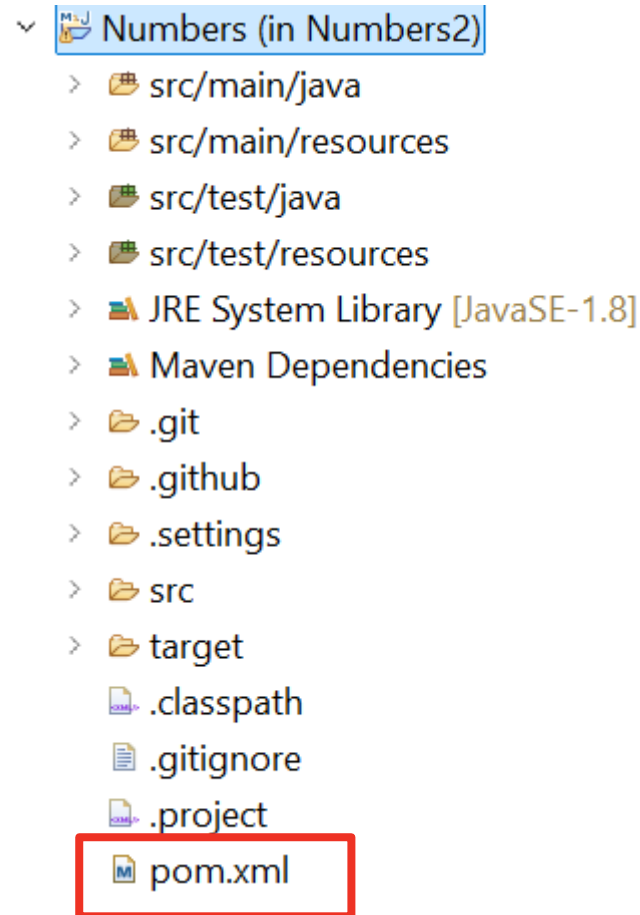
Steps to follow

- **Create a Maven-based Java project**
- Write a Number class that could
 - Give the sum of two given numbers ($i+j$)
 - Multiply two given numbers ($i*j$)
 - Subtract numbers: Subtract the second number from the first number ($i-j$)
 - Divide numbers: Divide the first number by the second number (i/j)
- Write Unit tests using JUnit
- Add GitHub Actions workflow to run tests once code is pushed to GitHub
- Link your project with the remote GitHub repository
- Demonstrate that the GitHub Actions is working as expected

Create a Maven-based Java project



POM.xml file will be at the project root



Steps to follow

- Create a Maven-based Java project
- **Write a Number class that could**
 - **Give the sum of two given numbers ($i+j$)**
 - **Multiply two given numbers ($i*j$)**
 - **Subtract numbers: Subtract the second number from the first number ($i-j$)**
 - **Divide numbers: Divide the first number by the second number (i/j)**
- Write Unit tests using JUnit
- Add GitHub Actions workflow to run tests once code is pushed to GitHub
- Link your project with the remote GitHub repository
- Demonstrate that the GitHub Actions is working as expected

Numbers class

Make sure you have placed the file here: src/main/java

```
Numbers.java x
1 public class Numbers {
2     public int sumNumbers (int i, int j) {
3         return i + j;
4     }
5
6     public int multiplyNumbers (int i, int j) {
7         return i * j;
8     }
9
10    public int subtractNumbers (int i, int j) {
11        return i - j;
12    }
13
14    public int divideNumbers (int i, int j) {
15        if (j == 0) {
16            throw new ArithmeticException("Division by zero is not allowed.");
17        }
18        return i / j;
19    }
20 }
```

Steps to follow

- Create a Maven-based Java project
- Write a Number class that could
 - Give the sum of two given numbers ($i+j$)
 - Multiply two given numbers ($i*j$)
 - Subtract numbers: Subtract the second number from the first number ($i-j$)
 - Divide numbers: Divide the first number by the second number (i/j)
- **Write Unit tests using JUnit**
- Add GitHub Actions workflow to run tests once code is pushed to GitHub
- Link your project with the remote GitHub repository
- Demonstrate that the GitHub Actions is working as expected

Write Unit tests using JUnit

Make sure you have placed the file here: : src/test/java

















```
*NumbersTest.java ×
1 import org.junit.jupiter.api.Test;
2 import static org.junit.jupiter.api.Assertions.*;
3
4 class NumbersTest {
5
6     Numbers numbers = new Numbers();
7
8     @Test
9     void sumNumbers() {
10         assertEquals(5, numbers.sumNumbers(2, 3));
11         assertEquals(-1, numbers.sumNumbers(1, -2));
12         assertEquals(0, numbers.sumNumbers(0, 0));
13     }
14
15     @Test
16     void multiplyNumbers() {
17         assertEquals(6, numbers.multiplyNumbers(2, 3));
18         assertEquals(-2, numbers.multiplyNumbers(1, -2));
19         assertEquals(0, numbers.multiplyNumbers(0, 5));
20     }
21
22     @Test
23     void subtractNumbers() {
24         assertEquals(-1, numbers.subtractNumbers(2, 3));
25         assertEquals(3, numbers.subtractNumbers(1, -2));
26         assertEquals(0, numbers.subtractNumbers(0, 0));
27     }
28
29     @Test
30     void divideNumbers() {
31         assertEquals(2, numbers.divideNumbers(6, 3));
32         assertEquals(-2, numbers.divideNumbers(4, -2));
33     }
34
35     @Test
36     void divideNumbersByZero() {
37         assertThrows(ArithmeticException.class, () -> numbers.divideNumbers(5, 0));
38     }
39 }
```

Steps to follow

- Create a Maven-based Java project
- Write a Number class that could
 - Give the sum of two given numbers ($i+j$)
 - Multiply two given numbers ($i*j$)
 - Subtract numbers: Subtract the second number from the first number ($i-j$)
 - Divide numbers: Divide the first number by the second number (i/j)
- Write Unit tests using JUnit
- **Add GitHub Actions workflow to run tests once code is pushed to GitHub**
- Link your project with the remote GitHub repository
- Demonstrate that the GitHub Actions is working as expected

Configure GitHub Actions workflow

Create a folder structure “.github/workflows” inside your project. This GitHub only recognises workflows if they are in this location.

- ▼  Numbers (in Numbers2)
 - >  src/main/java
 - >  src/main/resources
 - >  src/test/java
 - >  src/test/resources
 - >  JRE System Library [JavaSE-1.8]
 - >  Maven Dependencies
 - ▼  .github
 - ▼  workflows
 -  github-actions-demo.yml
 - >  .settings
 - >  src
 - >  target
 -  .classpath
 -  .project
 -  pom.xml

Configure GitHub Actions workflow ...

github-actions-demo.yml file

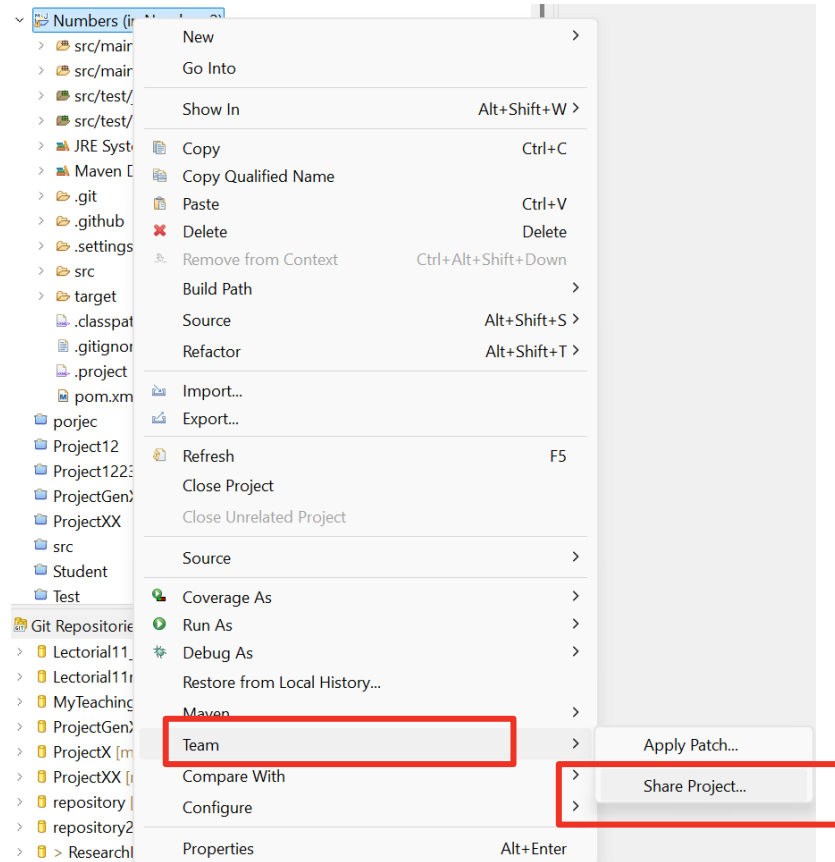
```
1 name: GitHub Actions Java test workflow # Workflow name
2 run-name: Running tests on github actions
3 on: [push] # Trigger the workflow on push events
4 jobs:
5   java-test: # Job name
6     runs-on: ubuntu-latest # Configures the job to run on the latest version of an Ubuntu Linux runner.
7     steps:
8       - name: Check out repository code
9         uses: actions/checkout@v4
10        # The uses keyword specifies that this step will run v4 of the actions/checkout action.
11        # This is an action that checks out your repository onto the runner,
12        # allowing you to run scripts or other actions against your code (such as build and test tools).
13        # You should use the checkout action any time your workflow will use the repository's code.
14
15       - name: Setup java # Install Java on the runner
16         uses: actions/setup-java@v4
17         with:
18           distribution: 'adopt'
19           # The distribution of Java to install. In this case, it specifies the AdoptOpenJDK distribution.
20           # See https://github.com/actions/setup-java?tab=readme-ov-file#supported-distributions for supported distributions.
21           java-version: '21' # The version of Java to install in the job runner.
22
23       - name: Run tests
24         run: mvn test
25        # This step runs the command mvn test, which is a Maven command to run the tests in your project.
26        # Maven is a build automation tool used primarily for Java projects.
27        # The tests are defined in your project's source code and will be executed by Maven.
28
29   another-job:
30     runs-on: ubuntu-latest
31     steps:
32       - name: Check out repository code
33         run: ls -la
```

Steps to follow

- Create a Maven-based Java project
- Write a Number class that could
 - Give the sum of two given numbers ($i+j$)
 - Multiply two given numbers ($i*j$)
 - Subtract numbers: Subtract the second number from the first number ($i-j$)
 - Divide numbers: Divide the first number by the second number (i/j)
- Write Unit tests using JUnit
- Add GitHub Actions workflow to run tests once code is pushed to GitHub
- **Link your project with the remote GitHub repository**
- Demonstrate that the GitHub Actions are working as expected

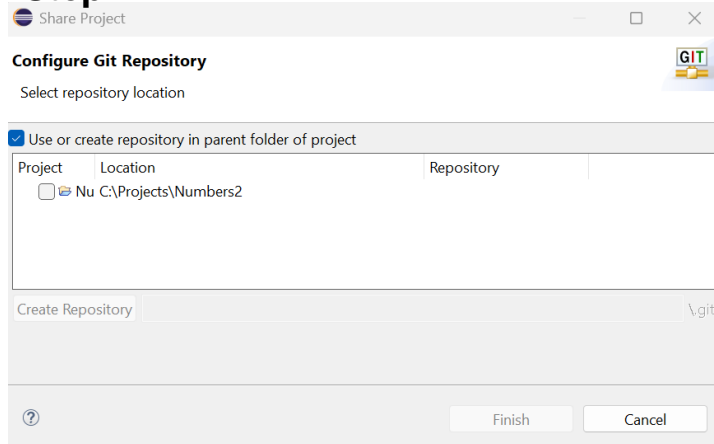
Link your project with the remote GitHub repository

Step 1

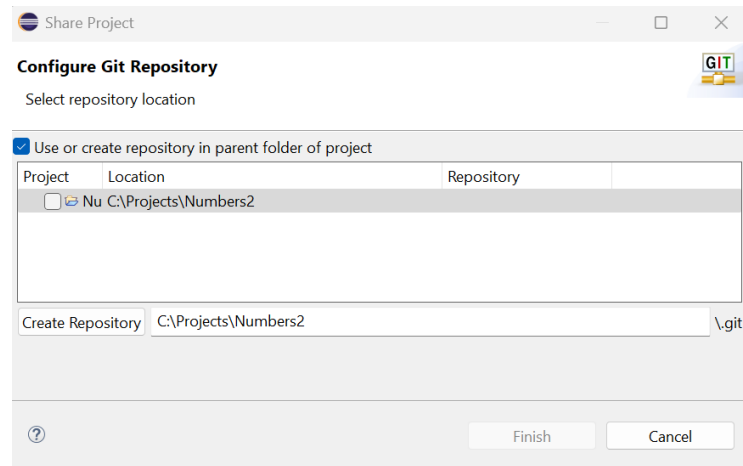


Link your project with the remote GitHub repository

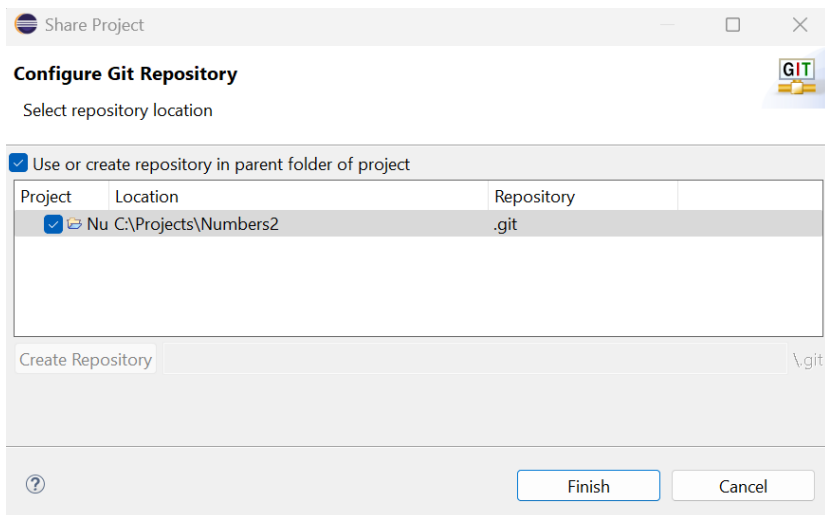
Step 2



Step 3: Select the project, where the create repository button will be enabled. Click create repository.



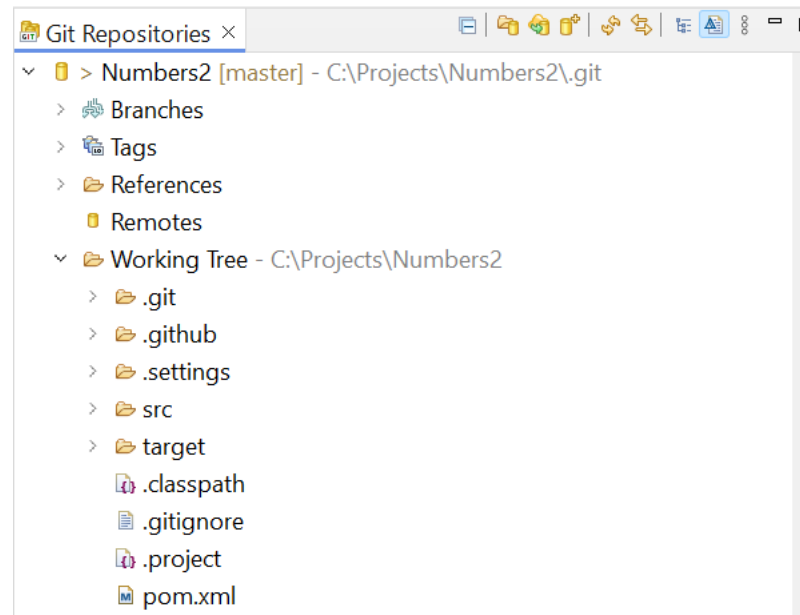
Step 4: Click finish



Link your project with the remote GitHub repository ...



Step 5: Make sure that the local repository is created.
Please note that .github is in the root directory of the working tree.
GitHub only recognises workflows if they are in this location.



Link your project with the remote GitHub repository ...



Step 6: Create a new GitHub repository


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*


Owner *

Repository name *

 asangi234


 /

Numbers


 Numbers is available.

Great repository names are short and memorable. Need inspiration? How about [effective-fiesta](#) ?

Description (optional)

☐  Public

Anyone on the internet can see this repository. You choose who can commit.

☒  Private

You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

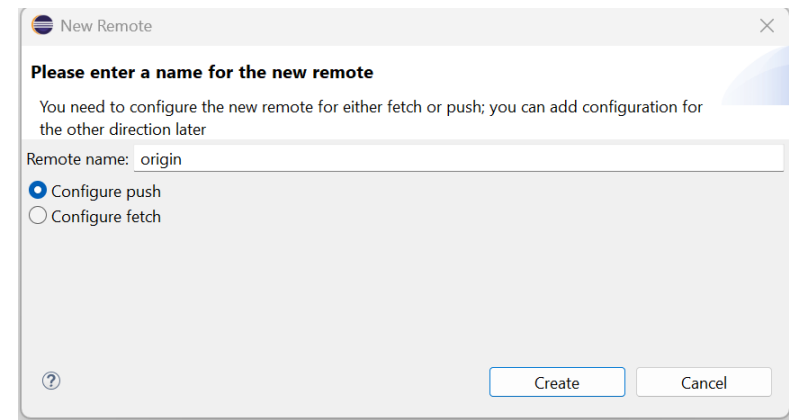
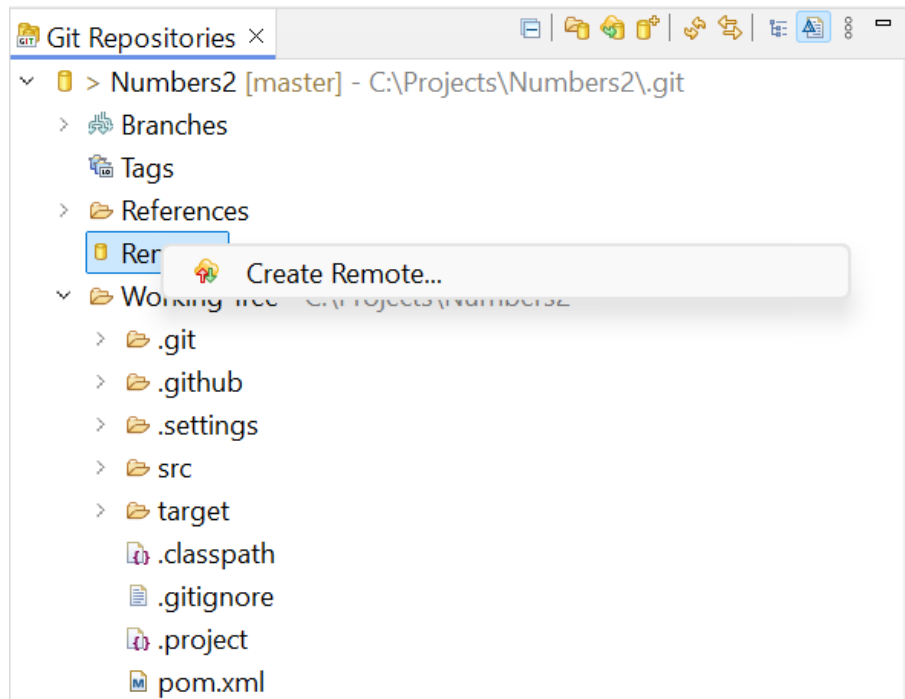
License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

Link your project with the remote GitHub repository ...



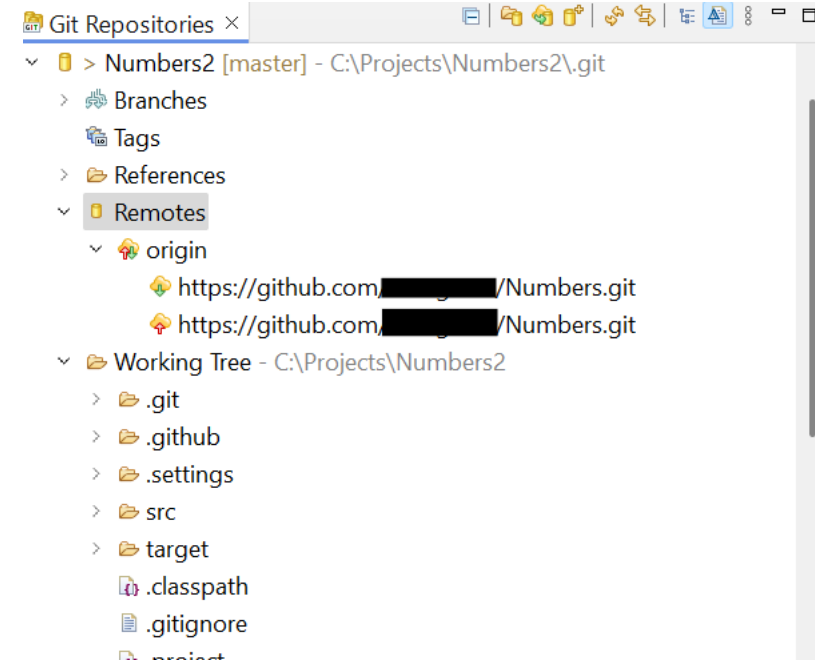
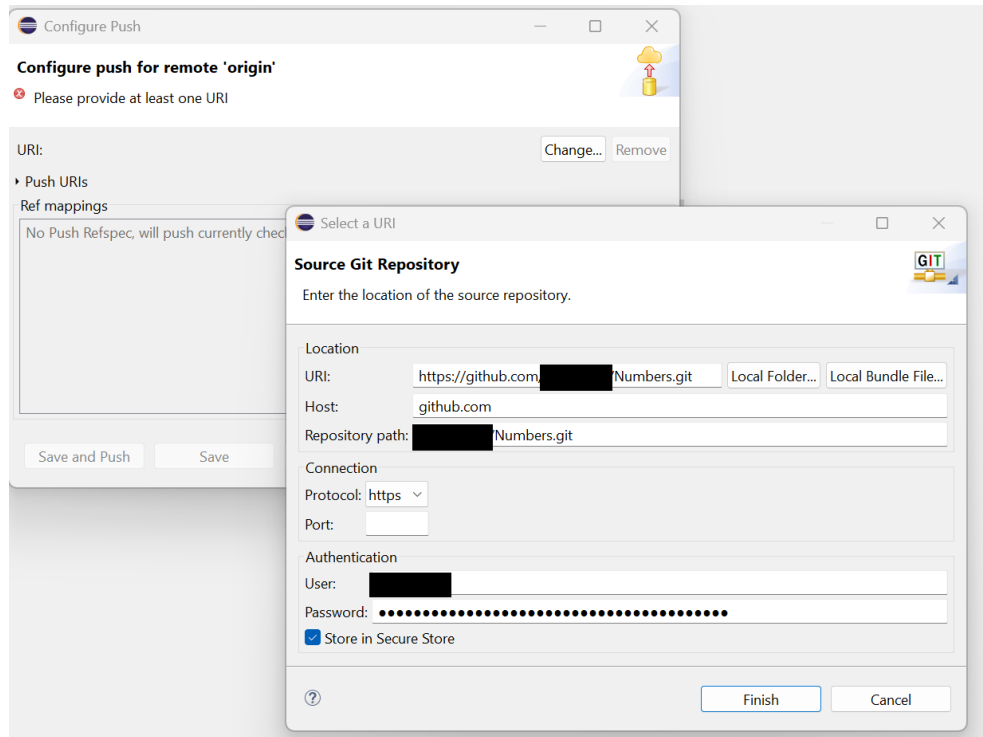
Step 7: Link the local repository to the remote repository



Link your project with the remote GitHub repository ...



Step 7: Link the local repository to the remote repository

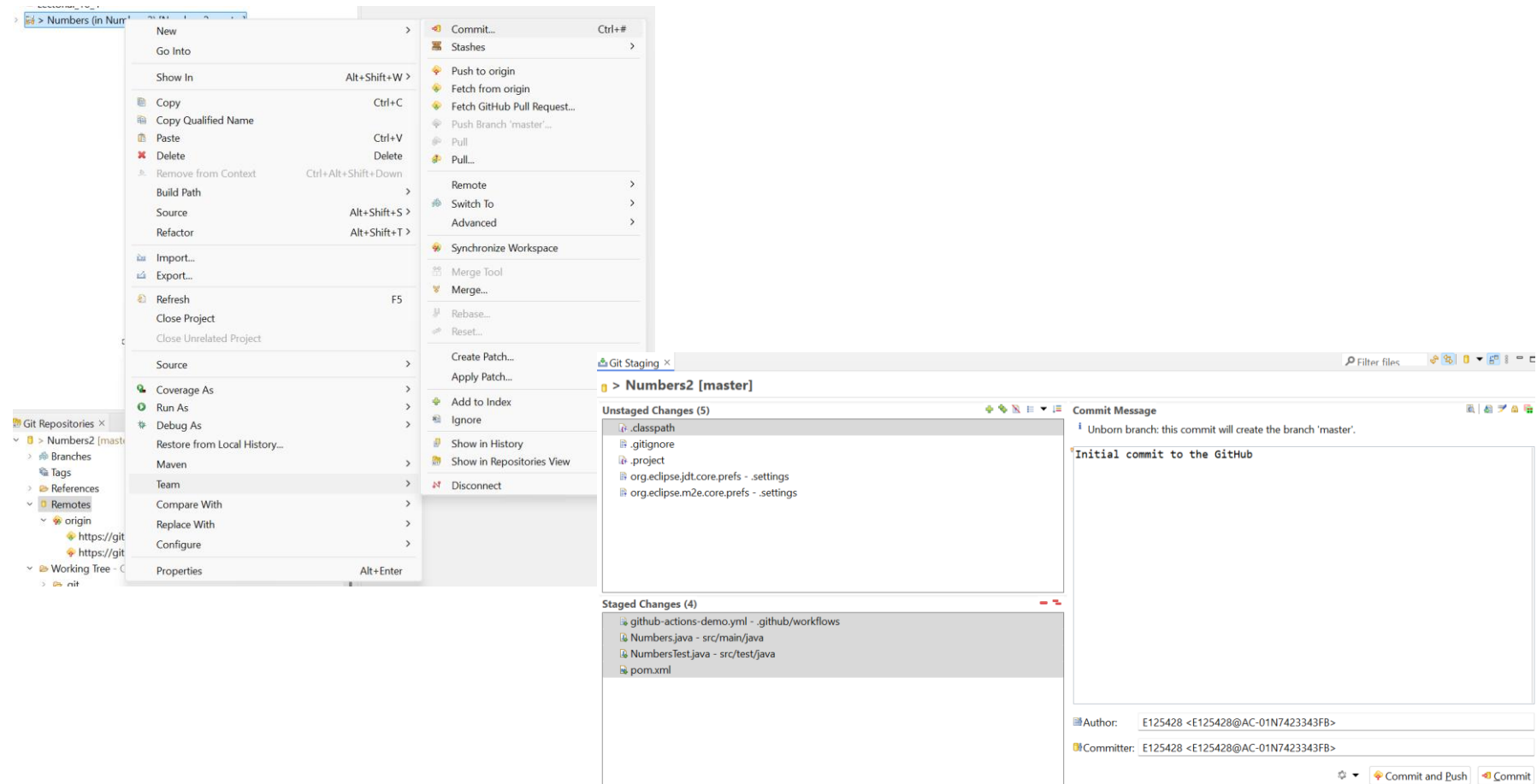


Steps to follow

- Create a Maven-based Java project
- Write a Number class that could
 - Give the sum of two given numbers ($i+j$)
 - Multiply two given numbers ($i*j$)
 - Subtract numbers: Subtract the second number from the first number ($i-j$)
 - Divide numbers: Divide the first number by the second number (i/j)
- Write Unit tests using JUnit
- Add GitHub Actions workflow to run tests once code is pushed to GitHub
- Link your project with the remote GitHub repository
- **Demonstrate that the GitHub Actions are working as expected**

Demonstrate that the GitHub Actions are working as expected

Step 1: Commit and push the files



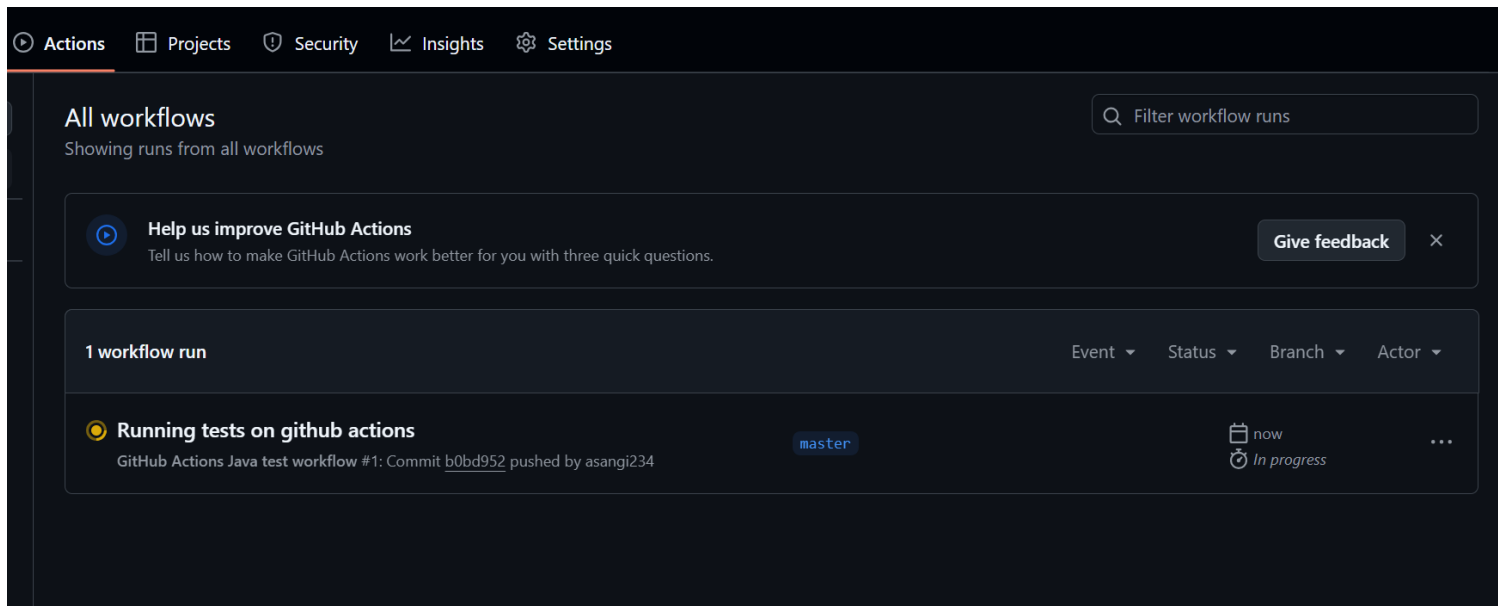
The screenshot illustrates the process of committing and pushing files to GitHub. The IDE's 'Git Staging' window shows the following files:

- Unstaged Changes (5):**
 - .classpath
 - .gitignore
 - .project
 - org.eclipse.jdt.core.prefs
 - org.eclipse.m2e.core.prefs
- Staged Changes (4):**
 - github-actions-demo.yml
 - Numbers.java
 - NumbersTest.java
 - pom.xml

The 'Commit Message' window shows the message: 'Initial commit to the GitHub'. The commit is being pushed to the 'master' branch.

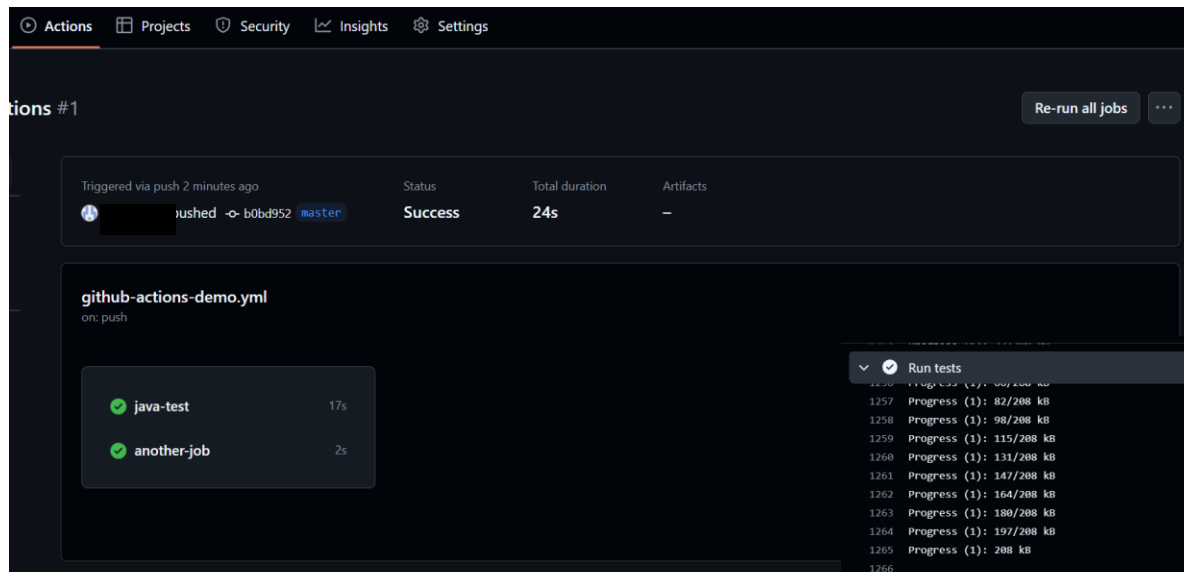
Demonstrate that the GitHub Actions are working as expected

Step 2: Immediately go to GitHub and see whether the pipeline has been triggered as expected



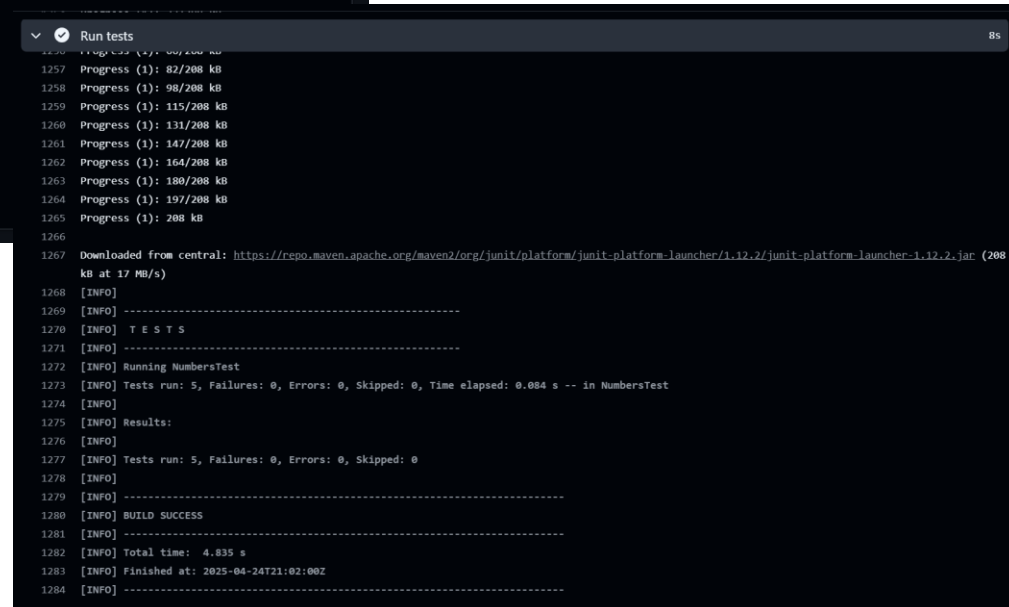
Demonstrate that the GitHub Actions are working as expected

We had two jobs in the workflow, and we can see that both of them have been executed successfully



The screenshot shows the GitHub Actions interface for a workflow named 'github-actions-demo.yml'. The workflow was triggered via a push 2 minutes ago and has a status of 'Success' with a total duration of 24s. Below the workflow summary, two jobs are listed: 'java-test' (17s) and 'another-job' (2s), both marked as successful with green checkmarks.

Job Name	Status	Duration
java-test	Success	17s
another-job	Success	2s

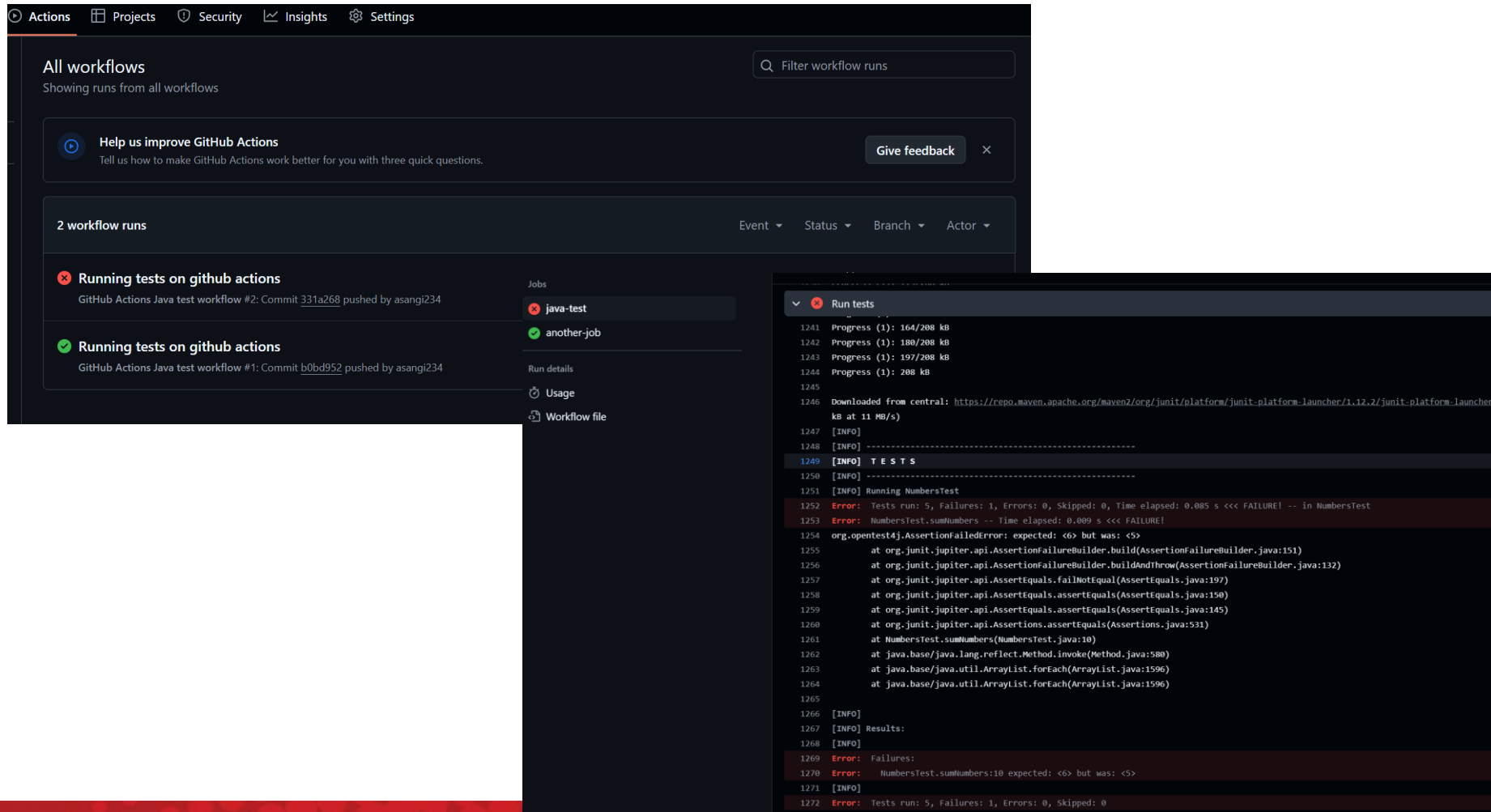


The screenshot shows the log output for the 'Run tests' job. The log displays progress bars for downloading dependencies, test results for 'NumbersTest', and a final 'BUILD SUCCESS' message.

```
1250 Progress (1): 82/208 kB
1251 Progress (1): 82/208 kB
1252 Progress (1): 98/208 kB
1253 Progress (1): 115/208 kB
1254 Progress (1): 131/208 kB
1255 Progress (1): 147/208 kB
1256 Progress (1): 164/208 kB
1257 Progress (1): 180/208 kB
1258 Progress (1): 197/208 kB
1259 Progress (1): 208 kB
1260
1261 Downloaded from central: https://repo.maven.apache.org/maven2/org/junit/platform/junit-platform-launcher/1.12.2/junit-platform-launcher-1.12.2.jar (208
1262 kB at 17 MB/s)
1263 [INFO]
1264 [INFO] -----
1265 [INFO] T E S T S
1266 [INFO] -----
1267 [INFO] Running NumbersTest
1268 [INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.084 s -- in NumbersTest
1269 [INFO]
1270 [INFO] Results:
1271 [INFO]
1272 [INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
1273 [INFO]
1274 [INFO] BUILD SUCCESS
1275 [INFO]
1276 [INFO] -----
1277 [INFO] Total time: 4.835 s
1278 [INFO] Finished at: 2025-04-24T21:02:00Z
1279 [INFO]
1280 [INFO] -----
```

Demonstrate that the GitHub Actions are working as expected

If the job fails you will be able to see this as well.



The screenshot displays the GitHub Actions interface. At the top, there are navigation tabs: Actions, Projects, Security, Insights, and Settings. Below these, the 'All workflows' section shows 'Showing runs from all workflows'. A 'Help us improve GitHub Actions' banner is present. The '2 workflow runs' section lists two runs:

- Running tests on github actions** (Failed): GitHub Actions Java test workflow #2: Commit 331a268 pushed by asangi234. The job 'java-test' is marked as failed.
- Running tests on github actions** (Successful): GitHub Actions Java test workflow #1: Commit b0bd952 pushed by asangi234. The job 'another-job' is marked as successful.

The 'Run details' section for the failed job shows the workflow file. The 'Run tests' job is expanded, showing the following output:

```

1241 Progress (1): 164/208 kb
1242 Progress (1): 180/208 kb
1243 Progress (1): 197/208 kb
1244 Progress (1): 208 kb
1245
1246 Downloaded from central: https://repo.maven.apache.org/maven2/org/junit/platform/junit-platform-launcher/1.12.2/junit-platform-launcher-1.12.2.jar
1247 [INFO]
1248 [INFO] -----
1249 [INFO] T E S T S
1250 [INFO] -----
1251 [INFO] Running NumbersTest
1252 Error: Tests run: 5, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.085 s <<< FAILURE! -- in NumbersTest
1253 Error: NumbersTest.sumNumbers -- Time elapsed: 0.009 s <<< FAILURE!
1254 org.opentest4j.AssertionFailedError: expected: <6> but was: <5>
1255     at org.junit.jupiter.api.AssertionFailureBuilder.build(AssertionFailureBuilder.java:151)
1256     at org.junit.jupiter.api.AssertionFailureBuilder.buildAndThrow(AssertionFailureBuilder.java:132)
1257     at org.junit.jupiter.api.AssertEquals.failNotEqual(AssertEquals.java:197)
1258     at org.junit.jupiter.api.AssertEquals.assertEquals(AssertEquals.java:150)
1259     at org.junit.jupiter.api.Assertions.assertEquals(Assertions.java:145)
1260     at org.junit.jupiter.api.Assertions.assertEquals(Assertions.java:531)
1261     at NumbersTest.sumNumbers(NumbersTest.java:10)
1262     at java.base/java.lang.reflect.Method.invoke(Method.java:580)
1263     at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)
1264     at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)
1265
1266 [INFO]
1267 [INFO] Results:
1268 [INFO]
1269 Error: Failures:
1270 Error: NumbersTest.sumNumbers:10 expected: <6> but was: <5>
1271 [INFO]
1272 Error: Tests run: 5, Failures: 1, Errors: 0, Skipped: 0
  
```


References

- <https://github.github.io/actions-cheat-sheet/actions-cheat-sheet.pdf>
- <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>
- <https://maven.apache.org/pom.html>