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)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>au.edu.rmit.sct</groupId> <!-- Convention is the reverse domain name -->
    <artifactId>numbers</artifactId> <!-- Name of the project -->
    <version>0.1.0-SNAPSHOT</version> <!-- Version of the project -->

    <properties>
        <!-- Java version -->
        <maven.compiler.release>21</maven.compiler.release>
        <junit.version>5.12.2</junit.version>
    </properties>

    <dependencies>
        <!-- JUnit 5 Dependency -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope> <!-- Test scope means it is only used for testing -->
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope> <!-- Test scope means it is only used for testing -->
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Lets use the latest version of the compiler plugin
            https://maven.apache.org/plugins/maven-compiler-plugin/index.html
             -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.14.0</version>
            </plugin>
            <plugin>
                <!-- We need the maven surefire plugin to run the unit tests
                https://maven.apache.org/surefire/maven-surefire-plugin/index.html
                 -->
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.5.3</version>
            </plugin>
        </plugins>
    </build>

</project>
```

You, 2 weeks ago • Added Maven support to ease running tests

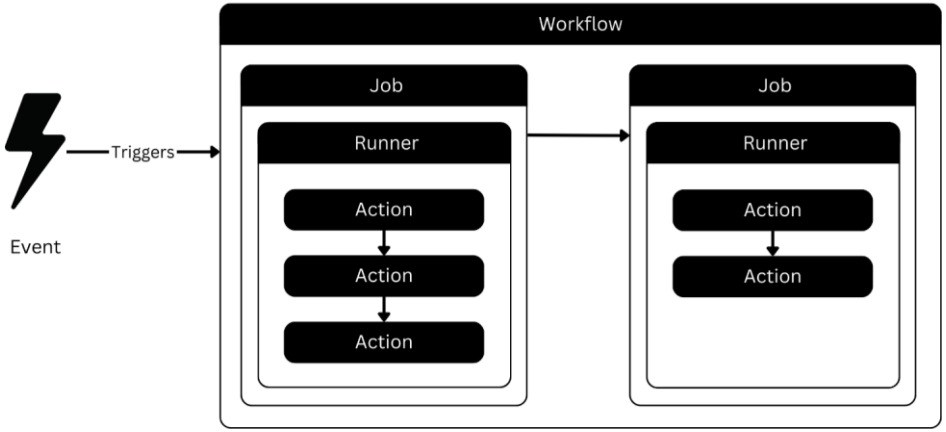
More details about pom.xml file

- 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.**

```yaml
name: GitHub Actions Java test workflow # Workflow name
run-name: Running tests on github actions
on: [push] # Trigger the workflow on push events
jobs:
  java-test: # Job name
    runs-on: ubuntu-latest # Configures the job to run on the latest version of an Ubuntu Linux runner.
    steps:
      - name: Check out repository code
        uses: actions/checkout@v4
        # The uses keyword specifies that this step will run v4 of the actions/checkout action.
        # This is an action that checks out your repository onto the runner,
        #  allowing you to run scripts or other actions against your code (such as build and test tools).
        # You should use the checkout action any time your workflow will use the repository's code.

      - name: Setup java # Install Java on the runner
        uses: actions/setup-java@v4
        with:
          distribution: 'adopt'
          # The distribution of Java to install. In this case, it specifies the AdoptOpenJDK distribution.
          # See https://github.com/actions/setup-java?tab=readme-ov-file#supported-distributions for supported distributions.
          java-version: '21'  # The version of Java to install in the job runner.

      - name: Run tests
        run: mvn test
        # This step runs the command mvn test, which is a Maven command to run the tests in your project.
        # Maven is a build automation tool used primarily for Java projects.
        # The tests are defined in your project's source code and will be executed by Maven.

  another-job:
    runs-on: ubuntu-latest
    steps:
      - name: Check out repository code
        run: ls -la
```

GitHub actions workflow syntax …

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

```
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.

```
jobs:
  java-test: # Job name
    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

```yaml
    steps:
      - name: Check out repository code
        uses: actions/checkout@v4
        # The uses keyword specifies that this step will run v4 of the actions/checkout action.
        # This is an action that checks out your repository onto the runner,
        #  allowing you to run scripts or other actions against your code (such as build and test tools).
        # You should use the checkout action any time your workflow will use the repository's code.

      - name: Setup java # Install Java on the runner
        uses: actions/setup-java@v4
        with:
          distribution: 'adopt'
          # The distribution of Java to install. In this case, it specifies the AdoptOpenJDK distribution.
          # See https://github.com/actions/setup-java?tab=readme-ov-file#supported-distributions for supported distributions.
          java-version: '21'  # The version of Java to install in the job runner.

      - name: Run tests
        run: mvn test
        # This step runs the command mvn test, which is a Maven command to run the tests in your project.
        # Maven is a build automation tool used primarily for Java projects.
        # The tests are defined in your project's source code and will be executed by Maven.
```

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

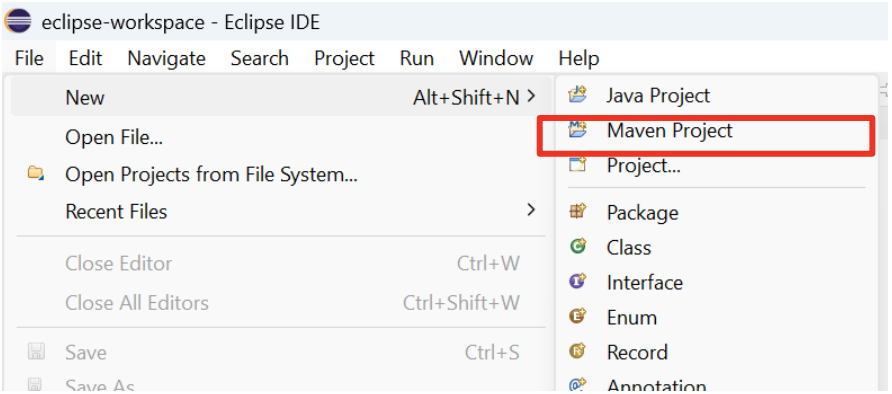
Steps to follow

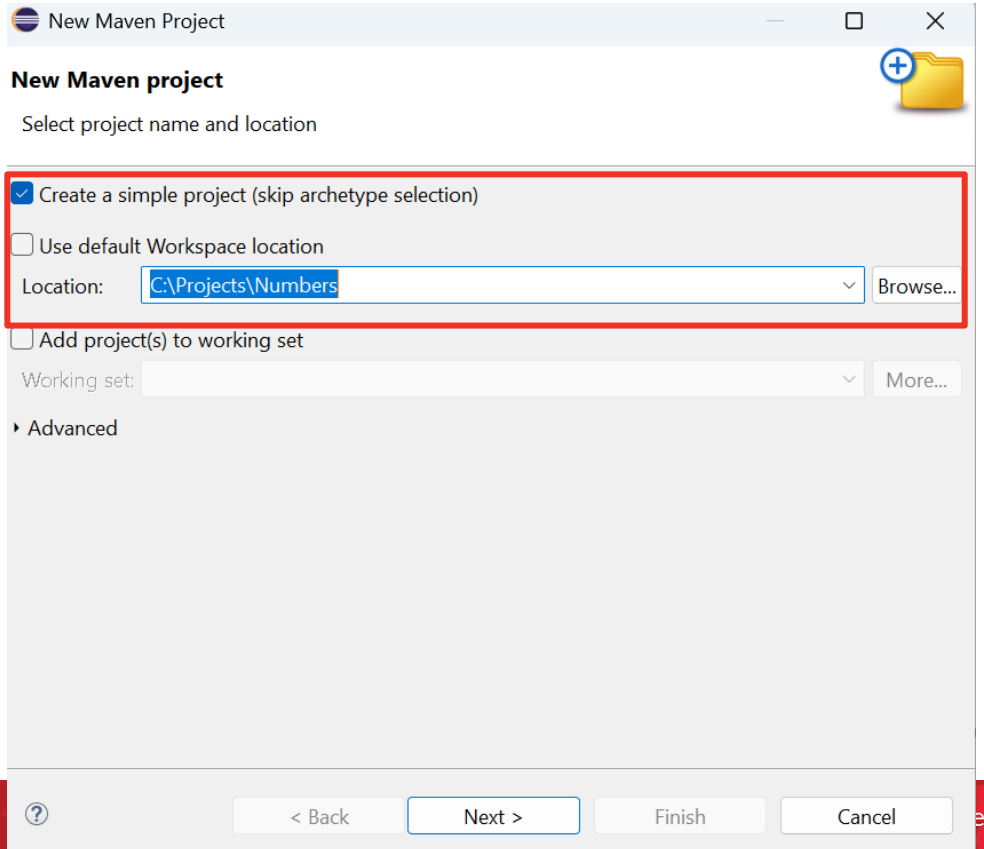
- Create a Maven-based Java project
- Write a Number class that could
  - Give the sum of two given numbers (i+j)
  - Multiply two given numbers (i*j)
  - Subtract numbers: Subtract the second number from the first number (i-j)
  - Divide numbers: Divide the first number by the second number (i/j)
- 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

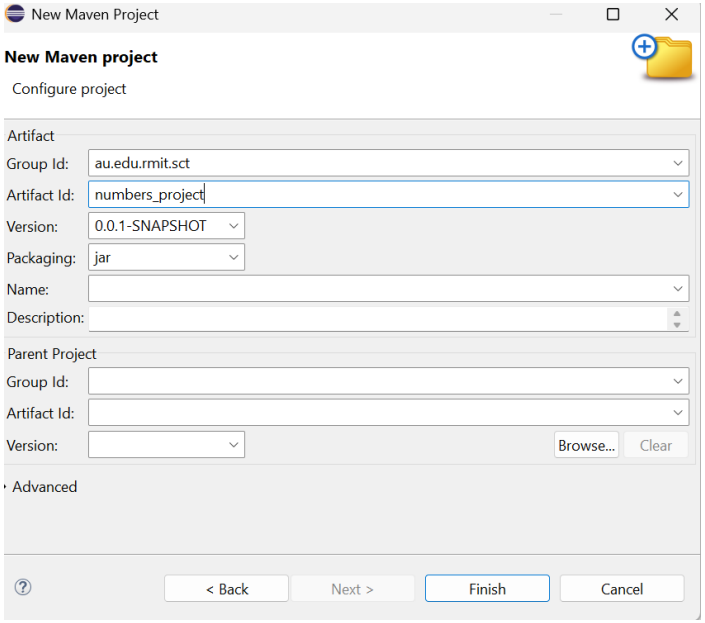
Steps to follow

- **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

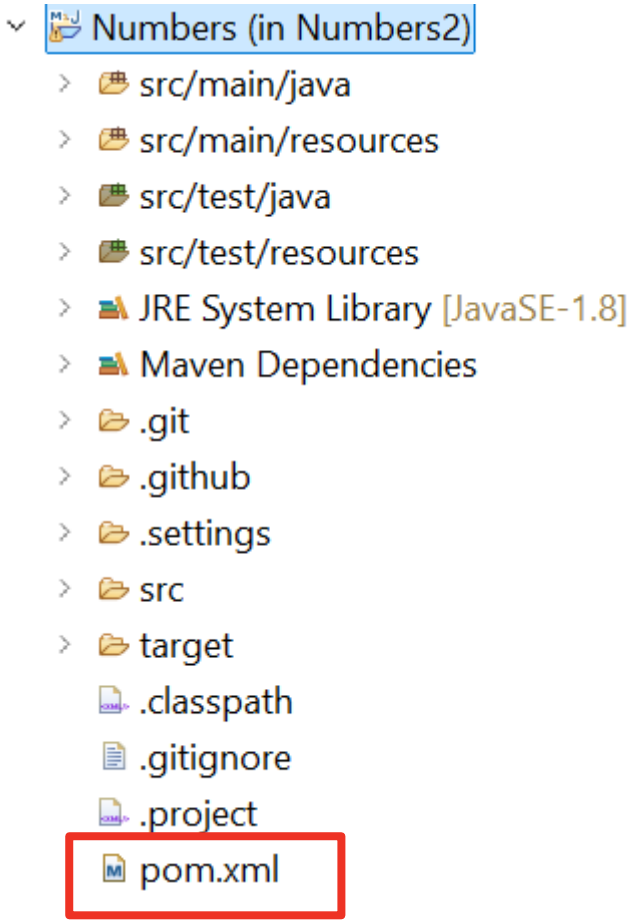
Create a Maven-based Java project







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

```java
public class Numbers {
    public int sumNumbers (int i, int j) {
        return i + j;
    }

    public int multiplyNumbers (int i, int j) {
        return i * j;
    }

    public int subtractNumbers (int i, int j) {
        return i - j;
    }

    public int divideNumbers (int i, int j) {
        if (j == 0) {
            throw new ArithmeticException("Division by zero is not allowed.");
        }
        return i / j;
    }
}
```

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

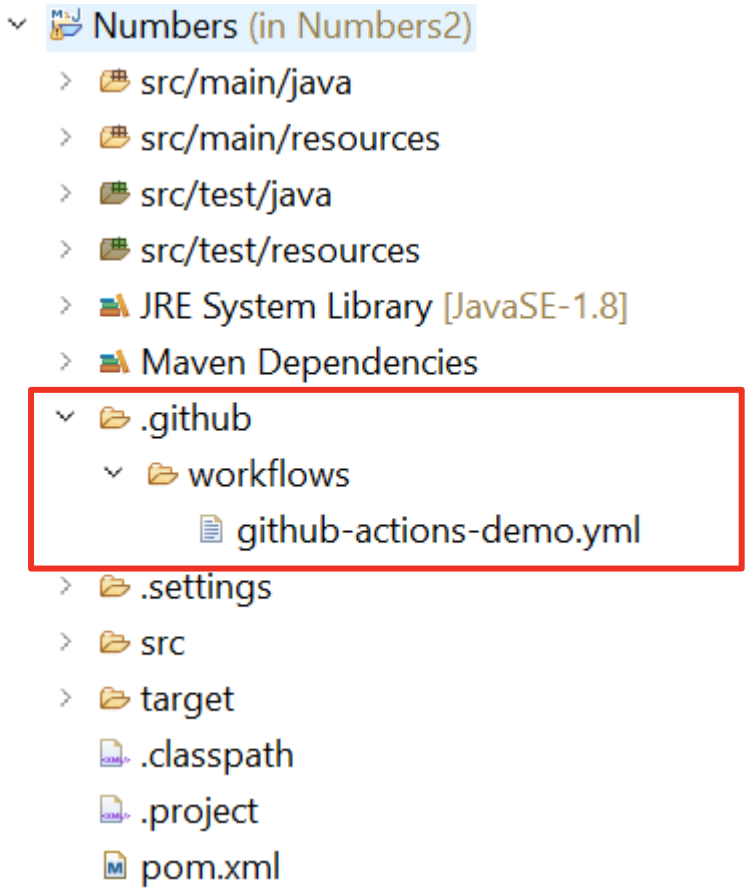
```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class NumbersTest {

    Numbers numbers = new Numbers();

    @Test
    void sumNumbers() {
        assertEquals(5, numbers.sumNumbers(2, 3));
        assertEquals(-1, numbers.sumNumbers(1, -2));
        assertEquals(0, numbers.sumNumbers(0, 0));
    }

    @Test
    void multiplyNumbers() {
        assertEquals(6, numbers.multiplyNumbers(2, 3));
        assertEquals(-2, numbers.multiplyNumbers(1, -2));
        assertEquals(0, numbers.multiplyNumbers(0, 5));
    }

    @Test
    void subtractNumbers() {
        assertEquals(-1, numbers.subtractNumbers(2, 3));
        assertEquals(3, numbers.subtractNumbers(1, -2));
        assertEquals(0, numbers.subtractNumbers(0, 0));
    }

    @Test
    void divideNumbers() {
        assertEquals(2, numbers.divideNumbers(6, 3));
        assertEquals(-2, numbers.divideNumbers(4, -2));
    }

    @Test
    void divideNumbersByZero() {
        assertThrows(ArithmeticException.class, () -> numbers.divideNumbers(5, 0));
    }
}
```

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.



Configure GitHub Actions workflow …

github-actions-demo.yml file

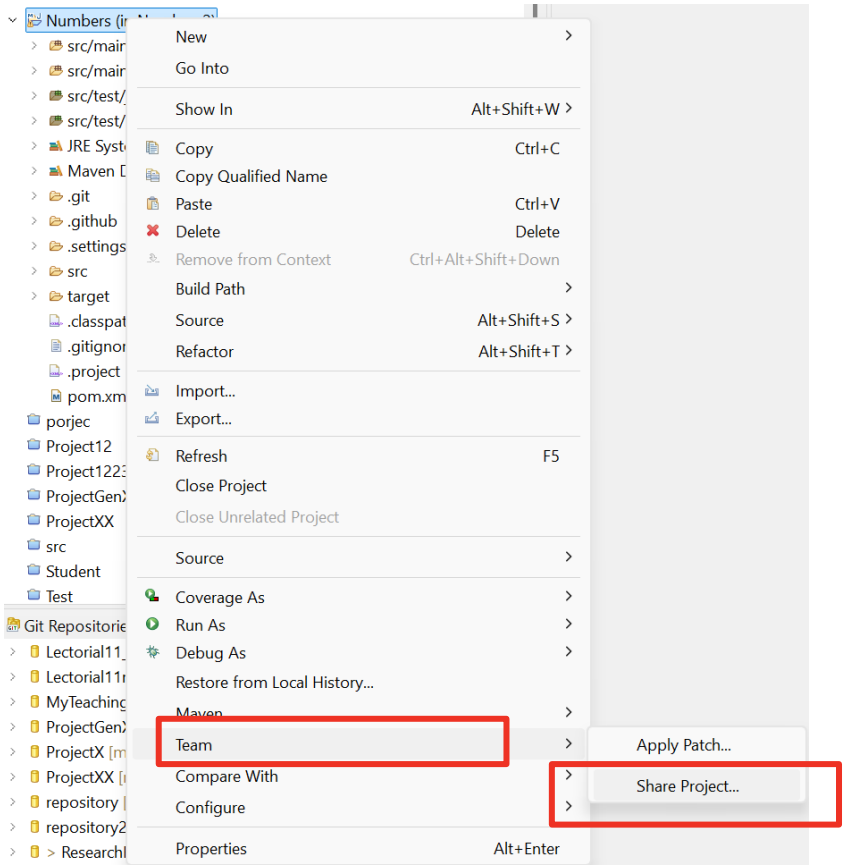
```yaml
name: GitHub Actions Java test workflow # Workflow name
run-name: Running tests on github actions
on: [push] # Trigger the workflow on push events
jobs:
  java-test: # Job name
    runs-on: ubuntu-latest # Configures the job to run on the latest version of an Ubuntu Linux runner.
    steps:
      - name: Check out repository code
        uses: actions/checkout@v4
        # The uses keyword specifies that this step will run v4 of the actions/checkout action.
        # This is an action that checks out your repository onto the runner,
        #  allowing you to run scripts or other actions against your code (such as build and test tools).
        # You should use the checkout action any time your workflow will use the repository's code.

      - name: Setup java # Install Java on the runner
        uses: actions/setup-java@v4
        with:
          distribution: 'adopt'
          # The distribution of Java to install. In this case, it specifies the AdoptOpenJDK distribution.
          # See https://github.com/actions/setup-java?tab=readme-ov-file#supported-distributions for supported distributions.
          java-version: '21'  # The version of Java to install in the job runner.

      - name: Run tests
        run: mvn test
        # This step runs the command mvn test, which is a Maven command to run the tests in your project.
        # Maven is a build automation tool used primarily for Java projects.
        # The tests are defined in your project's source code and will be executed by Maven.

  another-job:
    runs-on: ubuntu-latest
    steps:
      - name: Check out repository code
        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

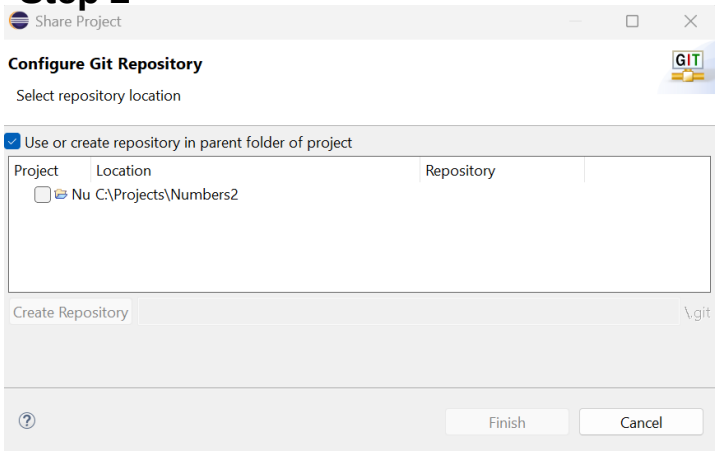
Link your project with the remote GitHub repository

Step 1

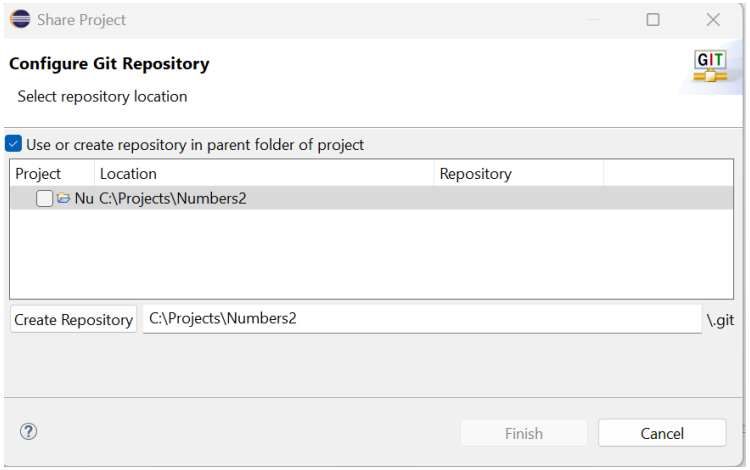


Link your project with the remote GitHub repository

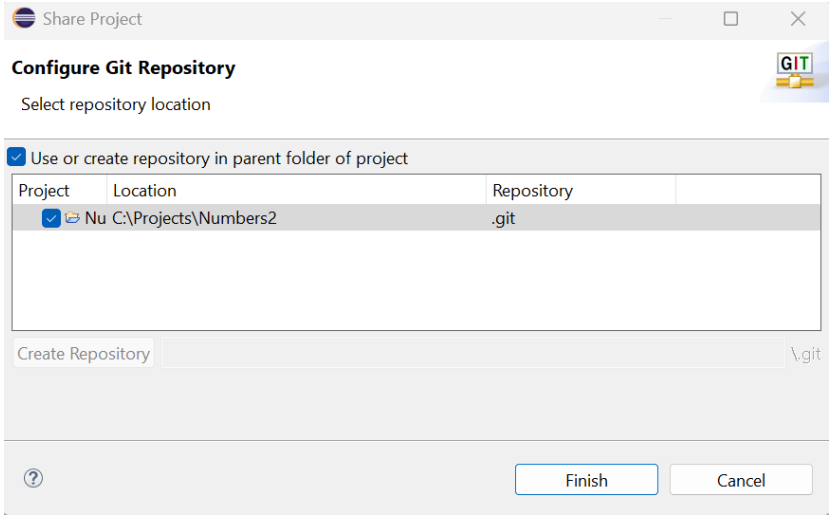
Step 2



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

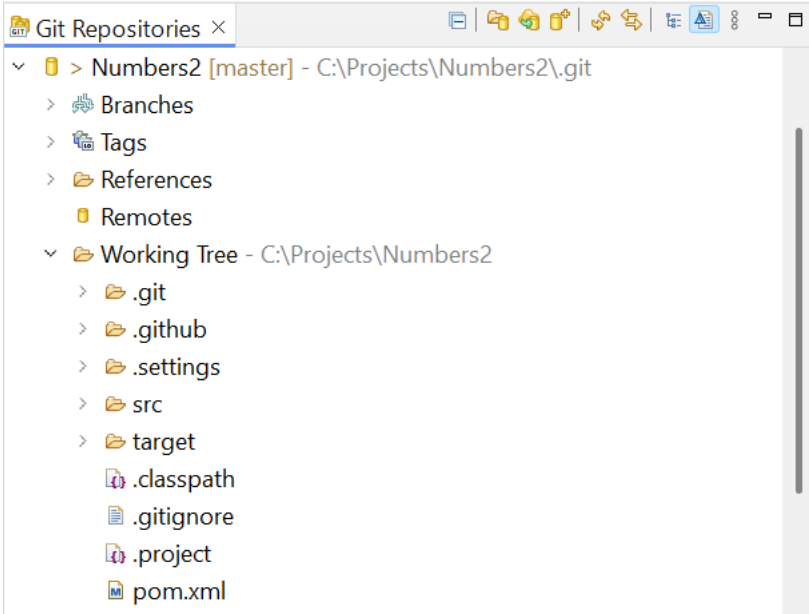


Step 4: Click finish



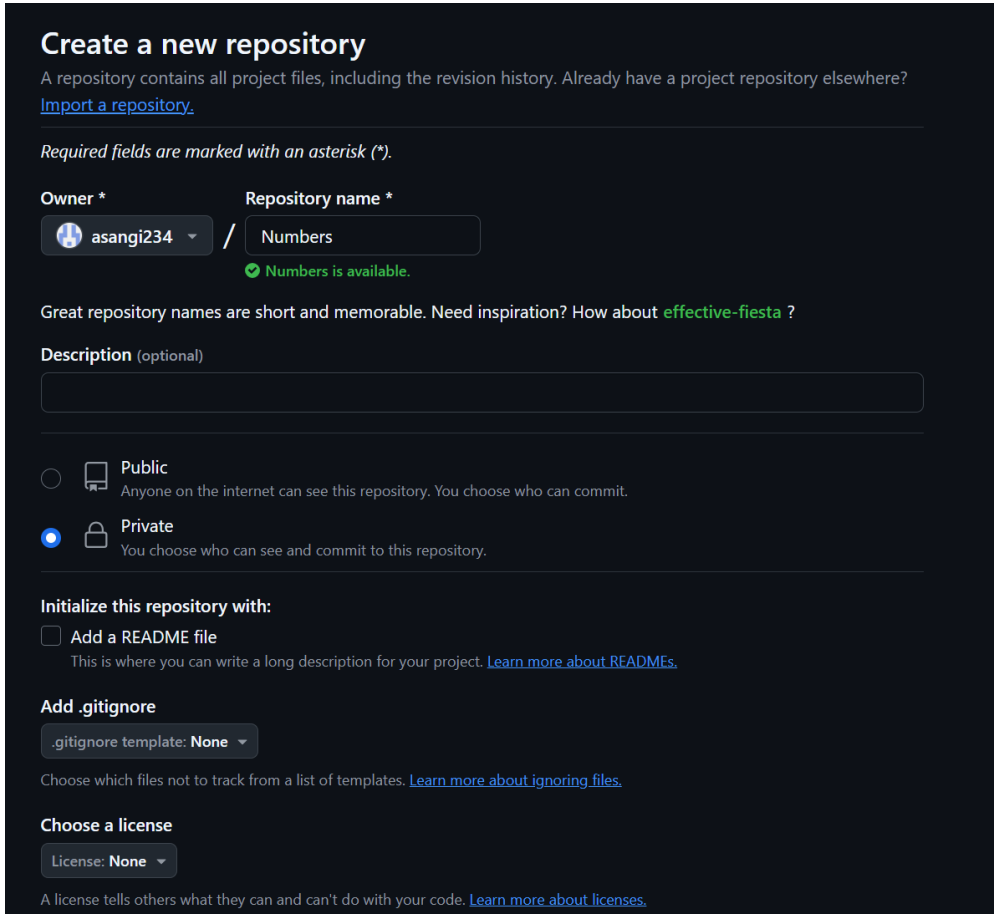
Link your project with the remote GitHub repository …

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.



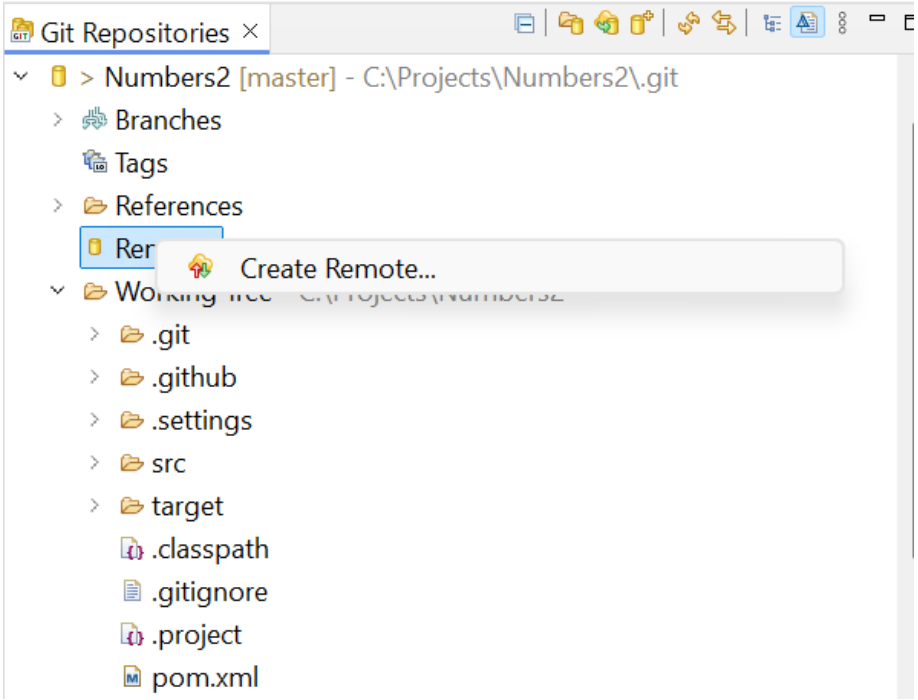
Link your project with the remote GitHub repository …

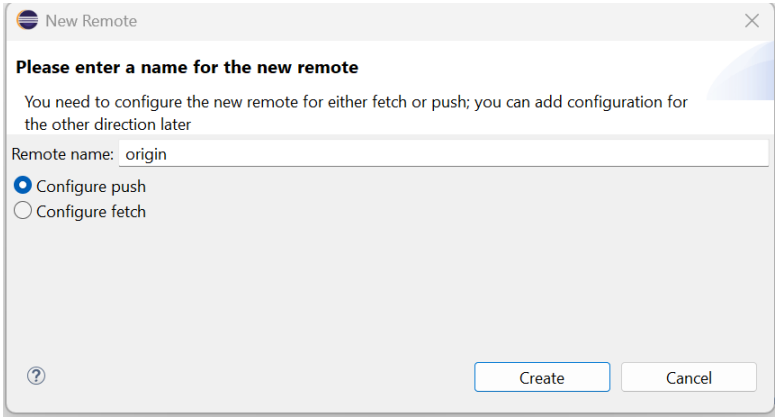
Step 6: Create a new GitHub repository



Link your project with the remote GitHub repository …

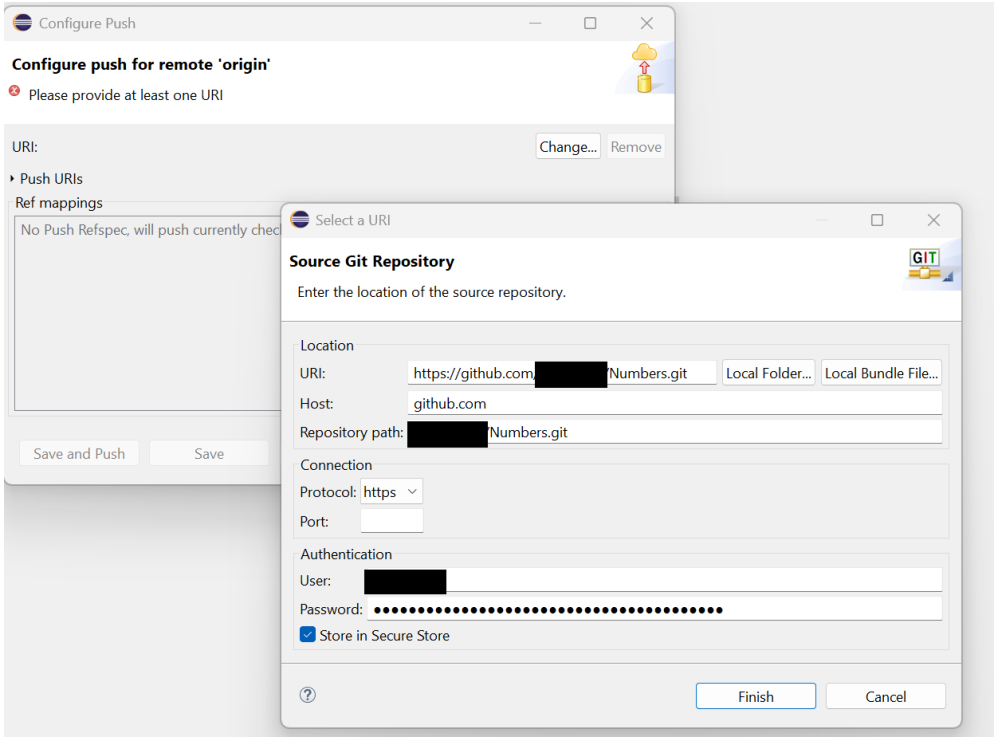
Step 7: Link the local repository to the remote repository

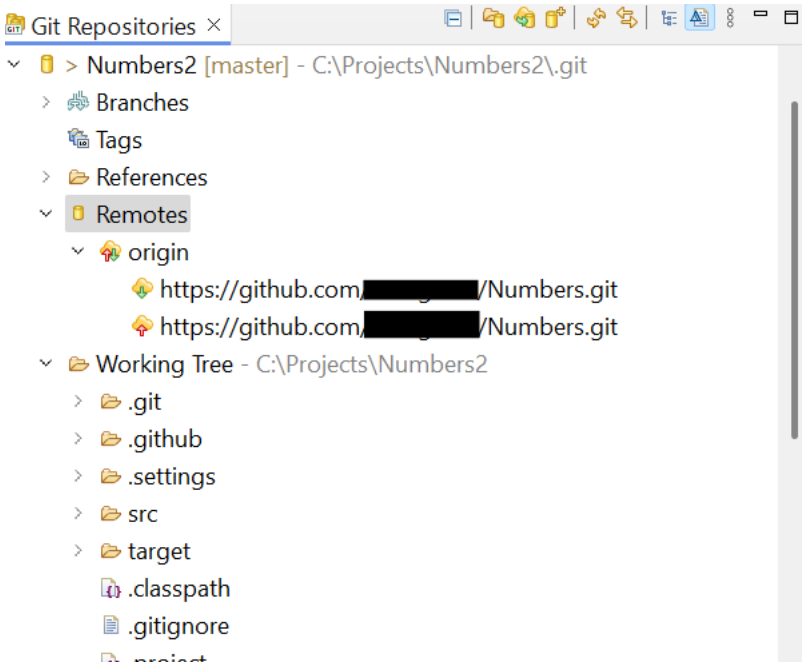




Link your project with the remote GitHub repository …

Step 7: Link the local repository to the remote repository



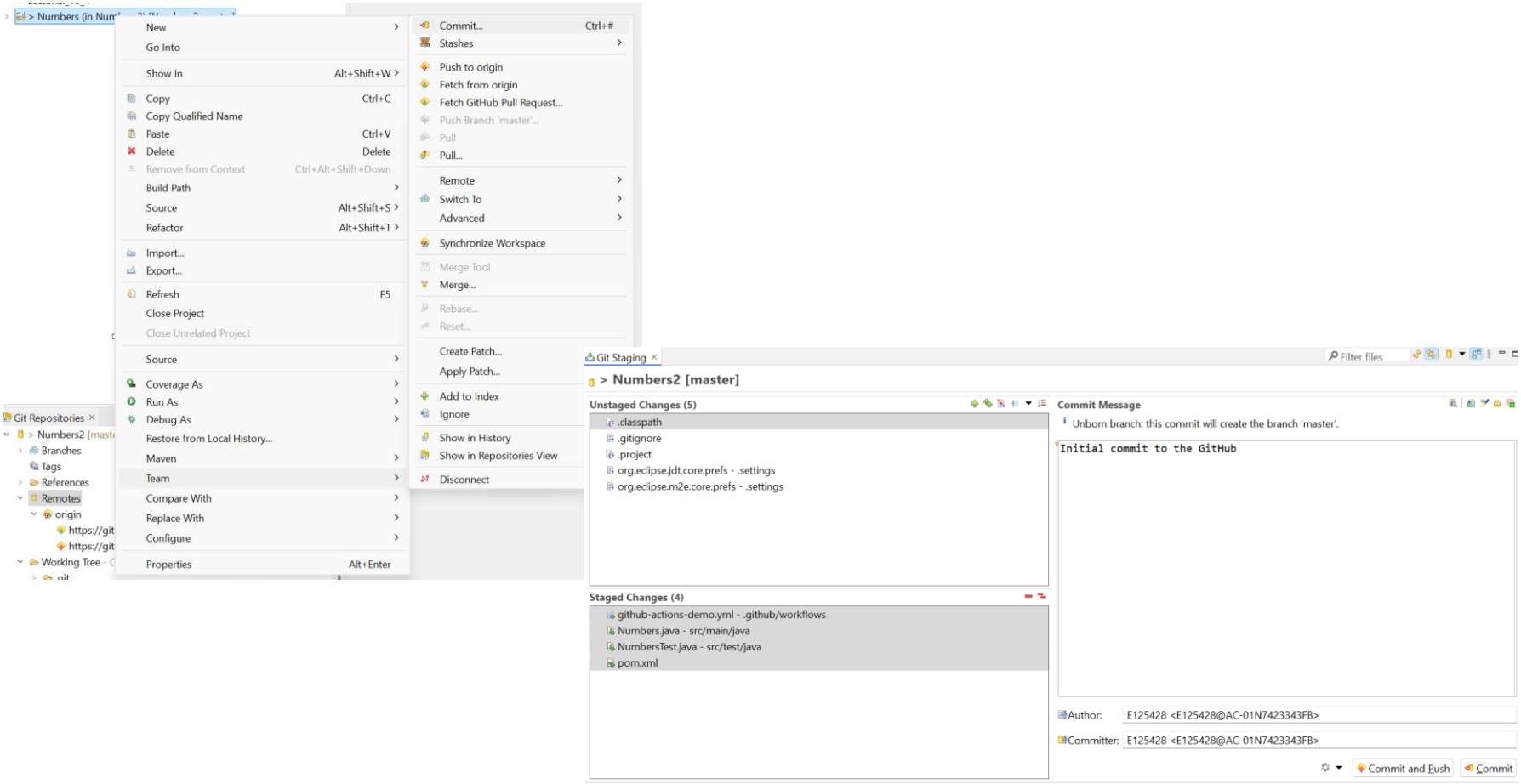


Steps to follow

- 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**

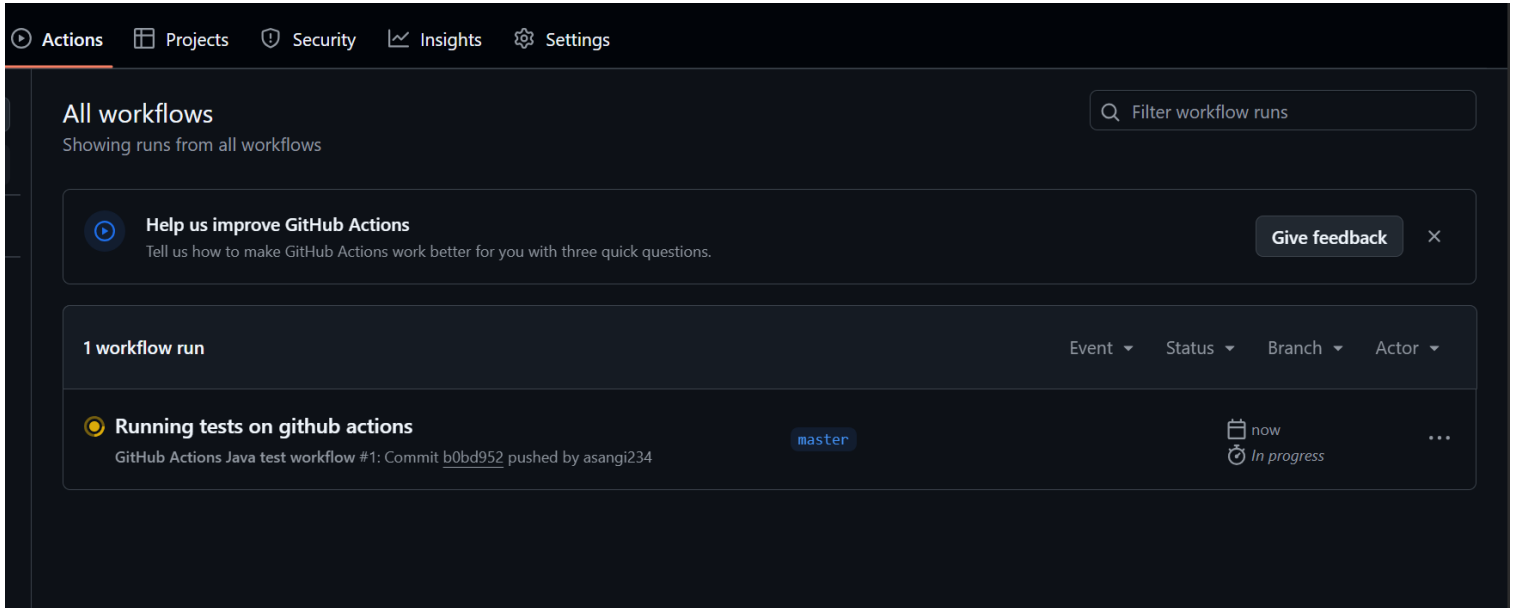
Demonstrate that the GitHub Actions are working as expected

Step 1: Commit and push the files



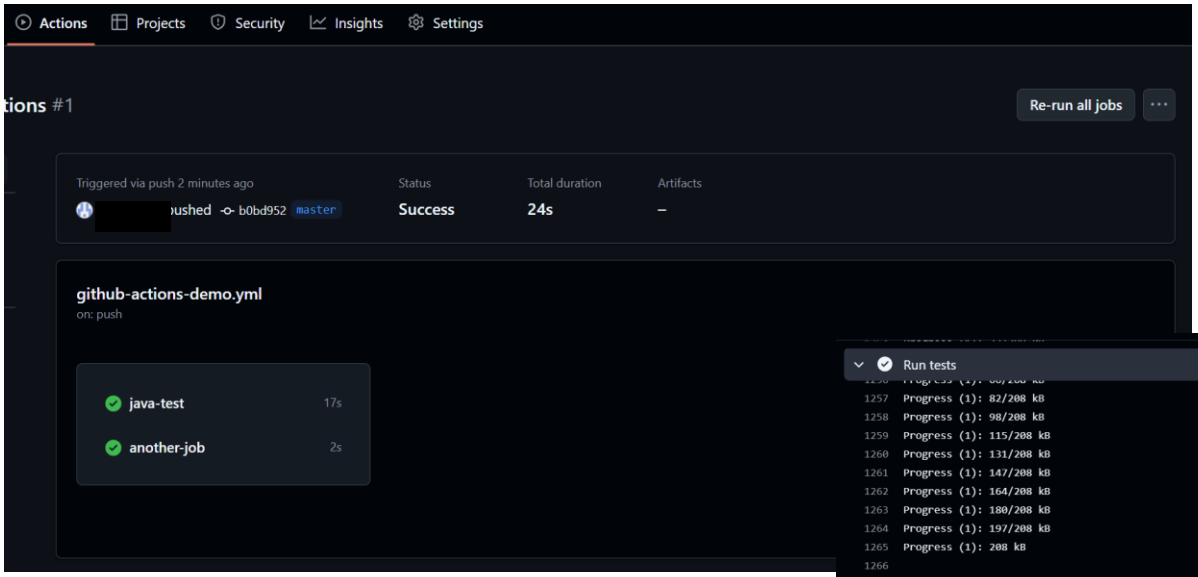
Demonstrate that the GitHub Actions are working as expected

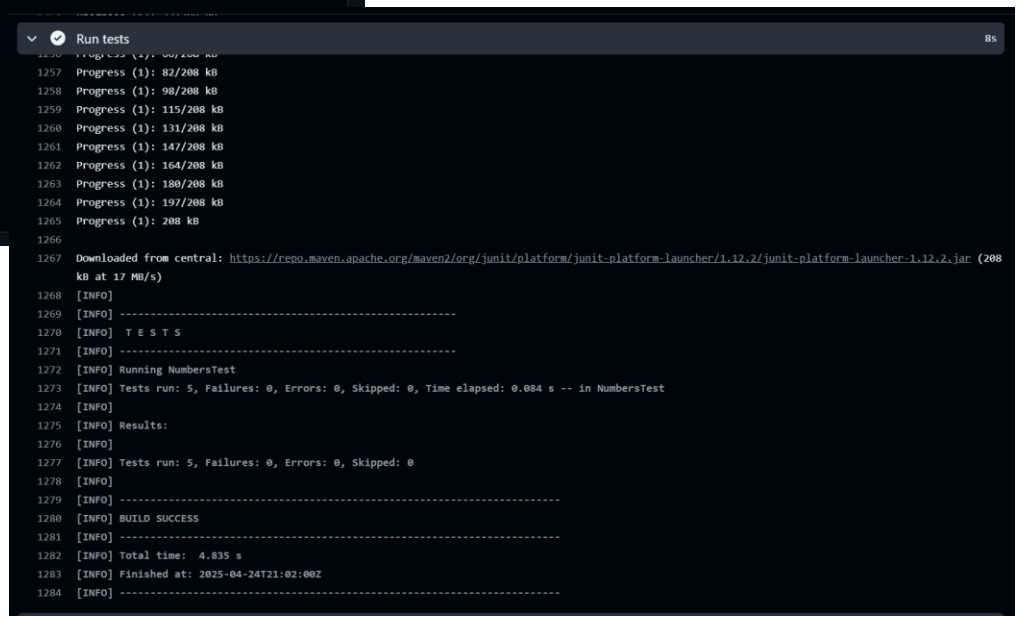
Step 2: Immediately go to GitHub and see whether the pipeline has been triggered as expected



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





Demonstrate that the GitHub Actions are working as expected

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

Actions | Projects | Security | Insights | Settings

All workflows
Showing runs from all workflows

Filter workflow runs

Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.
Give feedback

2 workflow runs
Event | Status | Branch | Actor

Running tests on github actions
GitHub Actions Java test workflow #2: Commit 331a268 pushed by asangi234

Running tests on github actions
GitHub Actions Java test workflow #1: Commit b0bd952 pushed by asangi234

Jobs
java-test
another-job

Run details
Usage
Workflow file

Run tests

```
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-
     kB at 11 MB/s)
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.AssertEquals.assertEquals(AssertEquals.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