JACOCO
Java Code Coverage

Introduction to JaCoCo

- **What is JaCoCo?**

  JaCoCo is a popular code coverage tool for Java applications. It integrates with build tools like Maven and Gradle to measure how much of your code is tested by your unit tests.

- **Purpose of JaCoCo:**

  Provides insights into how well your tests cover your codebase. It measures instruction coverage, branch coverage, and other metrics that indicate the effectiveness of the tests.

▼ How JaCoCo Works

- **Instrumentation —** JaCoCo instruments Java bytecode to record which parts of the code are executed during the test run.
- **Integration —** JaCoCo integrates seamlessly with Maven, Gradle, Ant, and even within CI/CD pipelines like GitHub Actions and Jenkins.
- JaCoCo doesn't change your source code directly. Instead, it tracks code execution at runtime and then compares it to the test suite.

▼ Setting Up JaCoCo with Maven

- **Add Plugin —** In your `pom.xml`, add the JaCoCo Maven plugin under the `<build>` section.

Example:

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.7</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

- **Run the Coverage Report —** After writing your tests, run:

```
mvn clean test jacoco:report
```

▼ Understanding the JaCoCo Report

ExamRegistry

Element	Missed Instructions	Cov.	Missed Branches	Cov.
enrol(Student, Exam)		100%		100%
recordScore(Student, Exam, int)		100%		83%
isEnrolled(Student, Exam)		100%		75%
getScore(Student, Exam)		100%		n/a
ExamRegistry()		100%		n/a
Total	0 of 116	100%	2 of 16	87%

- **Coverage Dashboard:**

  After running JaCoCo, navigate to `target/site/jacoco/index.html`. This will show the coverage summary for each class and method.

- **Reading the Report:**
  - Green = Fully covered
  - Red = Missed code or branches
  - Yellow = Partially covered branches.

▼ JaCoCo and Continuous Integration (CI)

- **Why Use JaCoCo in CI?**

  Ensures that each commit and pull request maintains a standard level of code quality and test coverage.

- **JaCoCo in CI/CD Pipelines:**

  JaCoCo can be integrated with GitHub Actions, Jenkins, Travis CI, etc., to automate the generation of reports after each test run.

- **Enforcing Coverage Thresholds:**

  You can enforce minimum coverage thresholds to ensure that your tests reach a certain percentage of coverage.

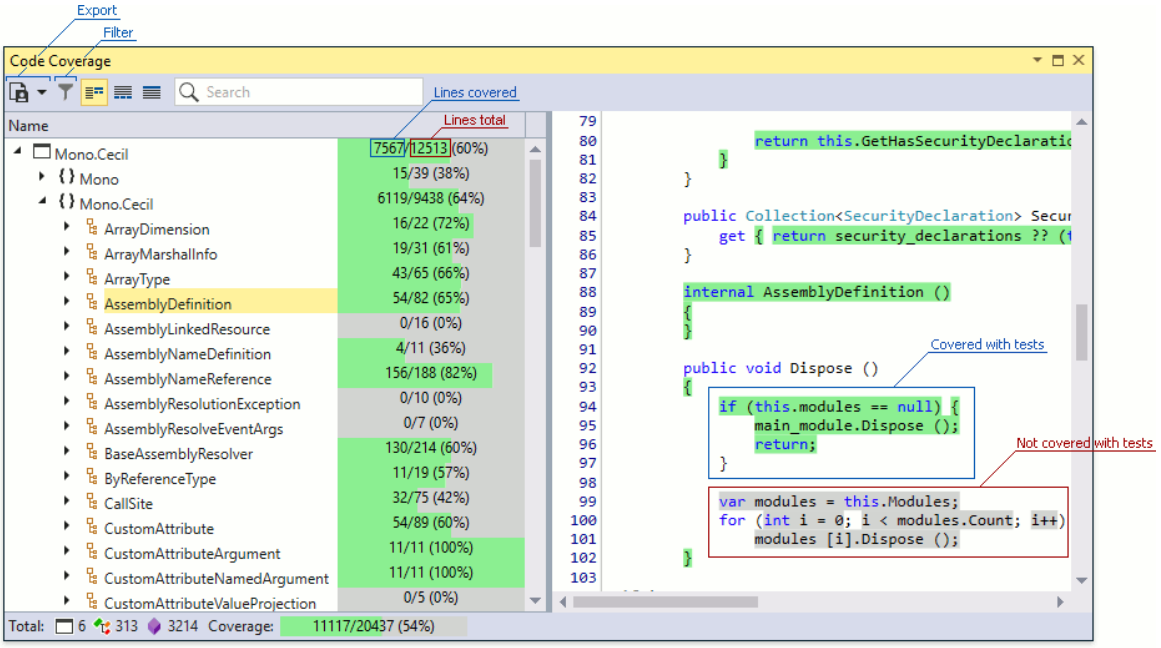
▼ JaCoCo Limitations

- **Coverage vs. Quality:**

  100% code coverage does not guarantee bug-free code. JaCoCo measures execution, but you still need to write meaningful tests.

- **Mocking Challenges:**

Testing certain parts of the code (e.g., integration with external services) may require mocks or stubs, which don't fully reflect real-world usage.



▼ Best Practices for Using JaCoCo Effectively

- **Balance Coverage with Testing Value:**

  Not every line of code needs testing—focus on complex and critical sections.

- **Set Realistic Coverage Goals:**

  Aiming for **80%–90%** coverage is a reasonable target, but don't force 100% at the expense of test quality.

- **Refactor Poorly Covered Code:**

  Use JaCoCo to identify areas where code can be refactored for better maintainability and testability.

▼ Integrating JaCoCo in GitHub Actions CI/CD

Step 1: Add a `.github/workflows/main.yml` file

- Create a workflow file that sets up the CI pipeline to run tests and generate the code coverage report using **JaCoCo**.
- Here's an example of a GitHub Actions workflow file:

```yaml
name: CI Pipeline

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up JDK 17
        uses: actions/setup-java@v4
        with:
          distribution: 'temurin'
          java-version: '17'
          cache: 'maven'

      - name: Cache Maven dependencies
        uses: actions/cache@v4
        with:
          path: ~/.m2/repository
          key: ${{ runner.os }}-maven-${{ hashFiles('**/
pom.xml') }}
          restore-keys: |
            ${{ runner.os }}-maven-
      - name: Build with Maven
        run: mvn clean install

      - name: Run Tests and Generate JaCoCo Report
        run: mvn test jacoco:report
```

```
      - name: Upload JaCoCo coverage report
        uses: actions/upload-artifact@v4
        with:
          name: jacoco-report
          path: target/site/jacoco/index.html
```

Step 2: View Coverage Report in GitHub Actions

- Once the pipeline runs, you can navigate to **Actions** in the GitHub repository to view the pipeline run. There, the **JaCoCo coverage report** will be uploaded as an artifact.
- You can download this artifact and open the `index.html` file locally to view the detailed coverage report.
