

# Trabajo Práctico 9

Materia: Ingeniería de Software 3

Alumno: Vietto Santiago

Docente: Fernando Bono

Institución: UCC

Año: 2022

## Ejercicio 1:

1)\_ En el archivo pom.xml del proyecto spring-boot, tenemos la dependencia spring-boot-starter-test, en donde esta es la dependencia principal que contiene la mayoría de los elementos necesarios para nuestras pruebas, es decir, proporciona dependencias de alcance de "prueba" como JUnit, Hamcrest y Mockito. Esta, provee las siguientes bibliotecas:

- JUnit 4: el estándar de facto para la prueba unitaria de aplicaciones Java.
- Spring Test y Spring Boot Test: utilidades y soporte de prueba de integración para aplicaciones Spring Boot.
- AssertJ: una biblioteca de aserciones fluida.

2)\_ Analizando el método HelloWorldServiceTest, tenemos a la notación @Test (perteneciente a JUnit 4) que identifica un método como un método de prueba. Luego vemos que se crea una instancia de la clase HelloWorldServiceTest y mediante la declaración AssertEquals comprueba que el mensaje ingresado a través del metodo getHelloMessage() sea igual o retorne "Spring boot says hello from a Docker container", y si esto se cumple se ejecuta correctamente.

\_ A continuación, ejecutamos el test en el IDE Eclipse y vemos que se ejecuta correctamente (para verificar el porque, accedemos al archivo HelloWorldService.java y vemos que la clase getHelloMessage retorna el mensaje preconfigurado):

The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The Package Explorer on the left shows the project structure with 'JUnit' selected. The main editor displays the 'HelloWorldServiceTest.java' file with the following code:

```
1 package sample.actuator;
2
3 import static org.junit.Assert.assertEquals;
4
5
6
7 public class HelloWorldServiceTest {
8
9     @Test
10    public void expectedMessage() {
11        HelloWorldService helloWorldService = new HelloWorldService();
12        assertEquals("Expected correct message", "Spring boot says hello from a Docker container", helloWorldService.getHelloMessage());
13    }
14
15 }
16
```

The Package Explorer shows 'sample.actuator.HelloWorldServiceTest' with a green bar indicating successful execution. The bottom status bar shows 'Problems', 'Javadoc', 'Declaration', 'Console', and 'Servers'. The Console output displays: '<terminated> HelloWorldServiceTest [JUnit] /usr/lib/jvm/jdk-17.0.4.1/bin/java (Oct 8, 2022, 5:05:02 PM - 5:05:03 PM) [pid: 12901]'.

### Ejercicio 3:

\_ Analizando el test `ExampleInfoContributorTest`, vemos que como primera medida tenemos a la notación `@Test` (perteneciente a JUnit 4) que identifica un método como un método de prueba. Luego se crea un objeto mock que es una implementación ficticia para una interfaz o una clase, en donde permite definir la salida de ciertas llamadas a métodos, por lo general, registran la interacción con el sistema y las pruebas pueden validarlo, y esto le permite simplificar la configuración de la prueba. En este caso el objeto mock llamado `builder` es una implementación ficticia para la clase `Info.Builder.class`, luego se le añade o asigna información con el método `contribute()` de la clase `ExampleInfoContributor`, y por último al ejecutarse verifica que el mock contenga algún tipo de información o dato agregado, en donde si es así se el test corre correctamente.

\_ Lo ejecutamos en el IDE Eclipse para verificar su correcta ejecución (para verificar el porque, accedemos al archivo `ExampleInfoContributor.java` y vemos que la clase `contribute` posee un método `builder` con valores previamente cargados):

```
eclipse-workspace - spring-boot-sample-actuator/src/test/java/sample/actuator/ExampleInfoContributorTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit x
Finished after 0.685 seconds
Runs: 1/1 Errors: 0 Failures: 0
sample.actuator.ExampleInfoContributorTest

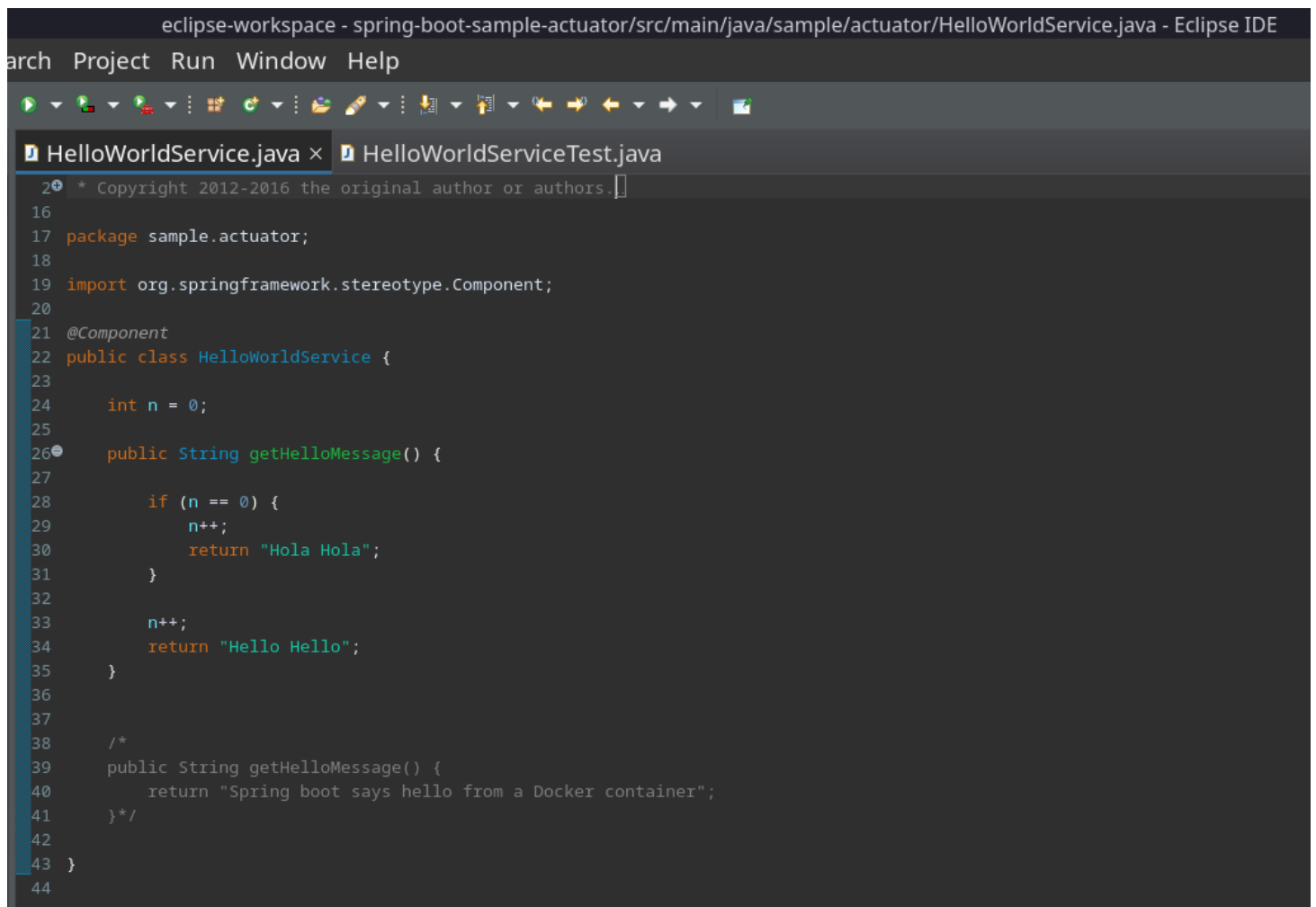
ExampleInfoContributorTest.java x
1 package sample.actuator;
2
3 import static org.mockito.ArgumentMatchers.any;
4
5
6
7
8
9
10 public class ExampleInfoContributorTest {
11
12     @Test
13     public void infoMap() {
14         Info.Builder builder = mock(Info.Builder.class);
15
16         ExampleInfoContributor exampleInfoContributor = new ExampleInfoContributor();
17         exampleInfoContributor.contribute(builder);
18
19         verify(builder).withDetail(any(), any());
20     }
21 }
22

Problems Javadoc Declaration Console x Servers
<terminated> ExampleInfoContributorTest [JUnit] /usr/lib/jvm/jdk-17.0.4.1/bin/java (Oct 8, 2022, 7:15:33 PM - 7:15:35 PM) [pid: 14519]
```

## Ejercicio 4:

1)\_ A continuación realizamos cambios en el archivo HelloWorldService.java para que cuando se llame por primera vez al método getHelloMessage, retorne "Hola Hola", y cuando se llame por segunda vez al método getHelloMessage, retorne "Hello Hello".

\_ Analizando un poco el código, vemos que primero tenemos un contador inicializado en cero, luego entramos al método get y vemos que se compara si el contador es igual a cero, primero se incrementa el contador ya que se llamó al método por primera vez y retorna el mensaje "Hola Hola", luego ante una segunda llamada del método, el contador que ya no vale mas cero porque se incremento en la primer llamada (valor igual a 1), se incrementa nuevamente (valor igual a 2) y el método retorna "Hello Hello".



```
eclipse-workspace - spring-boot-sample-actuator/src/main/java/sample/actuator/HelloWorldService.java - Eclipse IDE
arch Project Run Window Help

HelloWorldService.java x HelloWorldServiceTest.java

20 * Copyright 2012-2016 the original author or authors.
16
17 package sample.actuator;
18
19 import org.springframework.stereotype.Component;
20
21 @Component
22 public class HelloWorldService {
23
24     int n = 0;
25
26     public String getHelloMessage() {
27
28         if (n == 0) {
29             n++;
30             return "Hola Hola";
31         }
32
33         n++;
34         return "Hello Hello";
35     }
36
37
38     /*
39     public String getHelloMessage() {
40         return "Spring boot says hello from a Docker container";
41     }*/
42
43 }
44
```

\_ Luego construimos el test en el archivo HelloWorldServiceTest.java, en donde en primer lugar creamos una instancia de la clase helloWorldService, y mediante la declaración assertEquals en primer lugar comparamos que el mensaje que retorne en la primer llamada del método get de la instancia helloWorldService (mensaje real) sea igual a "Hola Hola" (mensaje esperado), y luego sucede lo mismo cuando se llama nuevamente al método los que esta vez comparamos que el mensaje que retorne en la segunda llamada del método get de la instancia helloWorldService (mensaje real) sea igual a "Hello Hello" (mensaje

esperado). En caso de que sea correcto el test se ejecuta correctamente, y en caso de que no, se devolverá un mensaje. A continuación vemos el código:

```
eclipse-workspace - spring-boot-sample-actuator/src/test/java/sample/actuator/HelloWorldServiceTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

HelloWorldServiceTest.java x
1 package sample.actuator;
2
3 import static org.junit.Assert.assertEquals;
4
5
6
7 public class HelloWorldServiceTest {
8
9
10 @Test
11 public void HelloMessageTest() {
12     HelloWorldService helloWorldService = new HelloWorldService();
13     assertEquals("Expected correct message", "Hola Hola", helloWorldService.getHelloMessage());
14     assertEquals("Expected correct message", "Hello Hello", helloWorldService.getHelloMessage());
15 }
16
17 /*
18 @Test
19 public void expectedMessage() {
20     HelloWorldService helloWorldService = new HelloWorldService();
21     assertEquals("Expected correct message", "Spring boot says hello from a Docker container", helloWorldService.getHelloMessage());
22 } */
23
24 }
25
```

\_ Ejecutamos el test y vemos que se ejecuta correctamente:

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays the package structure, with 'sample.actuator.HelloWorldServiceTest' selected. The JUnit test runner shows 'Finished after 0.02 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. The main editor shows the source code of 'HelloWorldServiceTest.java'. At the bottom, the 'Coverage' tab is active, displaying a table with the following data:

Element	Coverage	Covered Inst	Missed Inst
spring-boot-sample-actuator	16.2 %	43	223

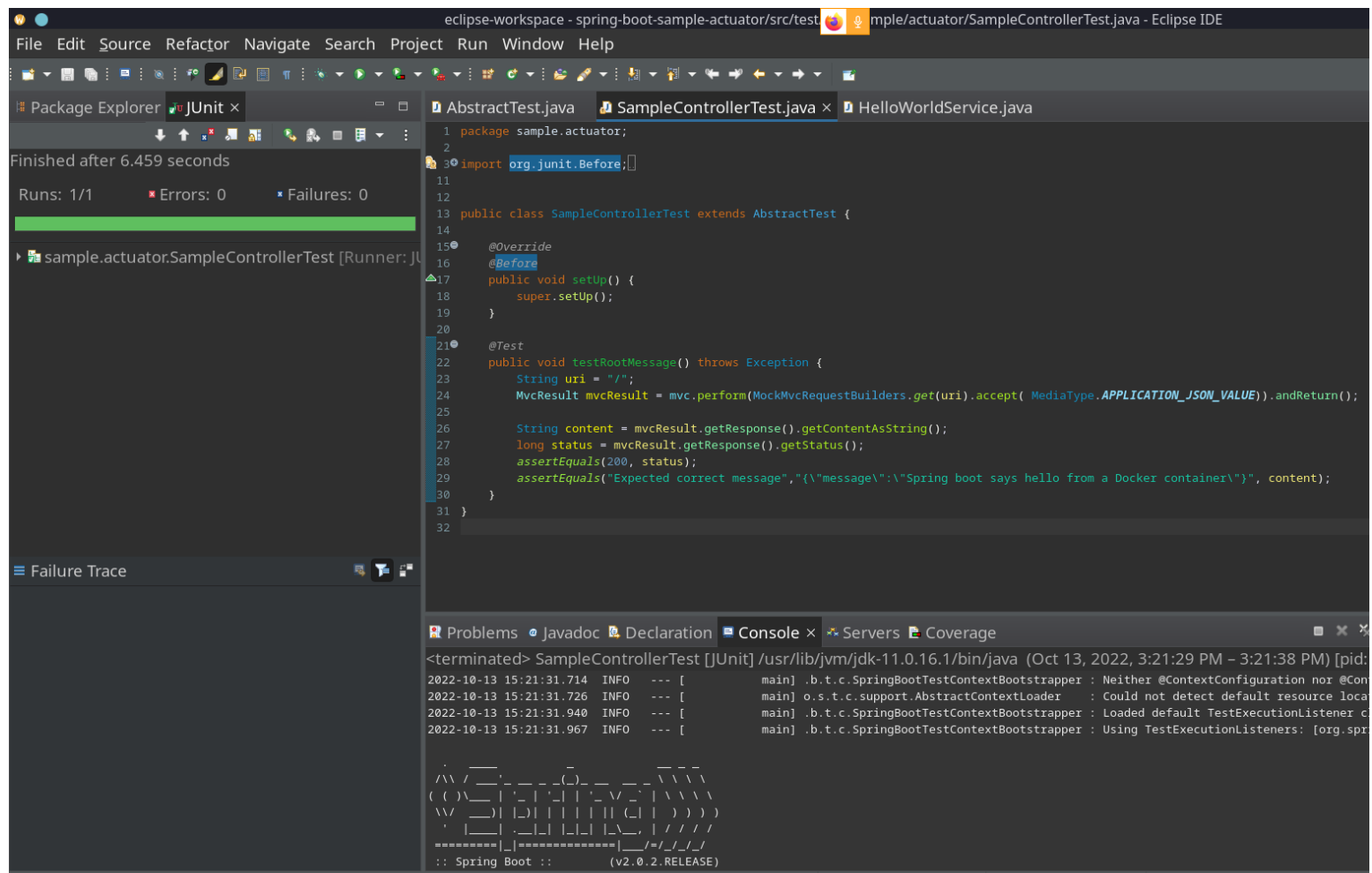
2)\_ Como primera medida creamos la clase de tipo JUnit llamada AbstractTest. En esta clase se instancia una variable mvc de tipo MockMvc, en donde este ofrece una “api fluida” que permite hacer una llamada web a través de una URL con todos los parámetros que sean necesarios, así como validar la corrección de las respuestas. Por otro lado, vemos que utiliza la notación @Autowired que es una de las anotaciones más habituales cuando se trabaja con Spring Framework ya que permite inyectar unas dependencias con otras dentro de Spring, y luego se crea el método setup que construye la variable mvc, y vemos que posee un WebApplicationContext que se utiliza para crear aplicaciones web.

```
AbstractTest.java x SampleControllerTest.java HelloWorldService.java
1 package sample.actuator;
2
3 import org.junit.runner.RunWith;
11
12 @RunWith(SpringJUnit4ClassRunner.class)
13 @SpringBootTest(classes = SampleActuatorApplication.class)
14 @WebAppConfiguration
15 public abstract class AbstractTest {
16     protected MockMvc mvc;
17
18     @Autowired
19     WebApplicationContext webApplicationContext;
20
21     protected void setUp() {
22         mvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();
23     }
24 }
25
```

\_ Por otro lado creamos la clase de tipo JUnit llamada SampleControllerTest. Esta clase hereda de la clase AbstractTest, en donde como primera medida recibe el método setUp que se va a ejecutar antes del test que se declara a continuación. Luego se construye un método TestRootMessage que es un test debido a su notación @Test, mediante el cual a través de una petición o request http a la uri "/", se guarda el response en una variable mvcResult, se verifica con un status 200 que se establece la conexión y que se obtuvo el response, y por último se compara el contenido del response con el contenido del archivo HelloWorldService.java.

```
AbstractTest.java SampleControllerTest.java x HelloWorldService.java
1 package sample.actuator;
2
3 import org.junit.Before;
11
12
13 public class SampleControllerTest extends AbstractTest {
14
15     @Override
16     @Before
17     public void setUp() {
18         super.setUp();
19     }
20
21     @Test
22     public void testRootMessage() throws Exception {
23         String uri = "/";
24         MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri).accept(MediaType.APPLICATION_JSON_VALUE)).andReturn();
25
26         String content = mvcResult.getResponse().getContentAsString();
27         long status = mvcResult.getResponse().getStatus();
28         assertEquals(200, status);
29         assertEquals("Expected correct message","{\"message\":\"Spring boot says hello from a Docker container\"}", content);
30     }
31 }
```

\_A continuación corremos el test y vemos que se ejecuta correctamente:



## Ejercicio 6:

\_ En github actions creamos un workflow llamado build\_maven\_test.yml, a través del cual añadimos el siguiente [script](#):

43 lines (36 sloc) | 1.12 KB

```
1  # This is a basic workflow to help you get started with Actions
2
3  name: Java CI with Maven
4
5  # Controls when the workflow will run
6  on:
7    # Triggers the workflow on push or pull request events but only for the master branch
8    push:
9      paths:
10       - 'spring-boot/**'
11    branches: [ main ]
12    pull_request:
13      paths:
14       - 'spring-boot/**'
15    branches: [ main ]
16
17  # Allows you to run this workflow manually from the Actions tab
18  workflow_dispatch:
19
20  # A workflow run is made up of one or more jobs that can run sequentially or in parallel
21  jobs:
22    # This workflow contains a single job called "build"
23    build:
24      # The type of runner that the job will run on
25      runs-on: ubuntu-latest
26
27      # Steps represent a sequence of tasks that will be executed as part of the job
28      steps:
29        # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
30        - uses: actions/checkout@v2
31
32        # Install Java JDK with maven
33        - name: Set up JDK 8
34          uses: actions/setup-java@v2
35          with:
36            java-version: '8'
37            distribution: 'adopt'
38            cache: maven
39
40        # Compile the application
41        - name: Build with Maven
42          run: |
43            mvn -B package --file pom.xml
```



santi2019 / spring-bootPublic

Pin

Unwatch1

Fork0

Star0

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

spring-boot / .github / workflows / build\_maven\_test.ymlin main

Cancel changesStart commit

<> Edit new file

Preview

Spaces2No wrap

```
13 paths:
14   - 'spring-boot/**'
15 branches: [ main ]
16
17 # Allows you to run this workflow manually from the Actions tab
18 workflow_dispatch:
19
20 # A workflow run is made up of one or more jobs that can run sequentially or in parallel
21 jobs:
22   # This workflow contains a single job called "build"
23   build:
24     # The type of runner that the job will run on
25     runs-on: ubuntu-latest
26
27     # Steps represent a sequence of tasks that will be executed as part of the job
28     steps:
29       # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
30       - uses: actions/checkout@v2
31
32       # Install Java JDK with maven
33       - name: Set up JDK 8
34         uses: actions/setup-java@v2
35         with:
36           java-version: '8'
37           distribution: 'adopt'
38           cache: maven
39
40       # Compile the application
41       - name: Build with Maven
42         run: |
43           mvn -B package --file pom.xml
```

Use `Control` + `Space` to trigger autocomplete in most situations.

MarketplaceDocumentation

Search Marketplace for Actions

Featured Actions

Cache

By actions

Cache artifacts like dependencies and build outputs to improve workflow execution time

3.2k

Setup Node.js environment

By actions

Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH

2.4k

Setup Go environment

By actions

Setup a Go environment and add it to the PATH

882

Download a Build Artifact

By actions

Download a build artifact that was previously uploaded in the workflow by the upload-artifact action

740

Setup .NET Core SDK

By actions

Used to build and publish .NET source. Set up a specific version of the .NET and authentication to private NuGet repository

606

Featured categories

\_ Le damos start commit, y luego procedemos a correr manualmente el workflow (pipeline) creado:

santi2019 / spring-bootPublic

Pin

Unwatch1

Fork0

Star0

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

Actions

New workflow

All workflows

CI

Docker

Java CI with Maven

Java CI with Maven

build\_maven\_test.yml

Filter workflow runs

0 workflow runs

EventStatusBranchActor

This workflow has a workflow\_dispatch event trigger.

Run workflow

Java CI with Maven

Java CI with Maven #1: Manually run by santi2019

5 minutes ago

31s

Java CI with Maven

Java CI with Maven #1

Re-run all jobs

Summary

Jobs

build

Run details

Usage

Workflow file

build

succeeded 4 minutes ago in 23s

Search logs

Set up job1s

Run actions/checkout@v22s

Set up JDK 82s

Build with Maven17s

Post Set up JDK 88s

Post Run actions/checkout@v28s

Complete job8s

\_ Lo que hacemos ahora es modificar el archivo HelloWorldService.java a proposito para chequear que la ejecución del test falle y por ende el workflow falle también:

The screenshot shows a GitHub Actions workflow run for 'Java CI with Maven #2'. The workflow is in a failed state, indicated by a red 'X' icon. The left sidebar shows the workflow steps: Summary, Jobs, and Run details. The 'Jobs' section is expanded, showing a list of jobs: build (failed), Set up job, Run actions/checkout@v2, Set up JDK 8, Build with Maven (failed), Post Set up JDK 8, Post Run actions/checkout@v2, and Complete job. The 'build' job is selected, and its details are shown. The job failed 24 seconds ago in 23s. The logs show the following steps: Set up job (2s), Run actions/checkout@v2 (2s), Set up JDK 8 (1s), Build with Maven (17s), Post Set up JDK 8 (0s), Post Run actions/checkout@v2 (0s), and Complete job (1s). The 'Build with Maven' step is highlighted in red, indicating failure. The logs for this step show the following output:

```
164 at org.apache.maven.surefire.booter.SurefireStarter.invokeProvider(SurefireStarter.java:172)
165 at org.apache.maven.surefire.booter.SurefireStarter.runSuitesInProcessWhenForked(SurefireStarter.java:104)
166 at org.apache.maven.surefire.booter.ForkedBooter.main(ForkedBooter.java:76)
167 2022-10-13 19:57:07.545 INFO 1670 --- [ Thread-2] o.s.w.c.s.GenericWebApplicationContext : Closing org.springframework.web.context.support.GenericWebApplicationContext@565b064f: startup date
[Thu Oct 13 19:57:04 UTC 2022]; root of context hierarchy
168
169 Results :
170
171 Failed tests: expectedMessage(sample.actuator.HelloWorldServiceTest): Expected correct message expected:<...om a Docker containe[r]> but was:<...om a Docker containe[]>
172 testRootMessage(sample.actuator.SampleControllerTest): Expected correct message expected:<...om a Docker containe[r]> but was:<...om a Docker containe[]>
173
174 Tests run: 3, Failures: 2, Errors: 0, Skipped: 0
175
176 [INFO] -----
177 [INFO] BUILD FAILURE
178 [INFO] -----
179 [INFO] Total time: 11.120 s
180 [INFO] Finished at: 2022-10-13T19:57:07Z
181 [INFO] -----
182 Error: Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.9:test (default-test) on project spring-boot-sample-actuator: There are test failures.
183 Error:
184 Error: Please refer to /home/runner/work/spring-boot/spring-boot/target/surefire-reports for the individual test results.
185 Error: -> [Help 1]
186 Error:
187 Error: To see the full stack trace of the errors, re-run Maven with the -e switch.
188 Error: Re-run Maven using the -X switch to enable full debug logging.
189 Error:
190 Error: For more information about the errors and possible solutions, please read the following articles:
191 Error: [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
192 Error: Process completed with exit code 1.
```

The screenshot shows a GitHub Actions workflow run for 'Java CI with Maven #2'. The workflow is in a failed state, indicated by a red 'X' icon. The left sidebar shows the workflow steps: Summary, Jobs, and Run details. The 'Jobs' section is expanded, showing a list of jobs: build (failed), Set up job, Run actions/checkout@v2, Set up JDK 8, Build with Maven (failed), Post Set up JDK 8, Post Run actions/checkout@v2, and Complete job. The 'build' job is selected, and its details are shown. The job failed 24 seconds ago in 23s. The logs show the following steps: Set up job (2s), Run actions/checkout@v2 (2s), Set up JDK 8 (1s), Build with Maven (17s), Post Set up JDK 8 (0s), Post Run actions/checkout@v2 (0s), and Complete job (1s). The 'Build with Maven' step is highlighted in red, indicating failure. The logs for this step show the following output:

```
164 at org.apache.maven.surefire.booter.SurefireStarter.invokeProvider(SurefireStarter.java:172)
165 at org.apache.maven.surefire.booter.SurefireStarter.runSuitesInProcessWhenForked(SurefireStarter.java:104)
166 at org.apache.maven.surefire.booter.ForkedBooter.main(ForkedBooter.java:76)
167 2022-10-13 19:57:07.545 INFO 1670 --- [ Thread-2] o.s.w.c.s.GenericWebApplicationContext : Closing org.springframework.web.context.support.GenericWebApplicationContext@565b064f: startup date
[Thu Oct 13 19:57:04 UTC 2022]; root of context hierarchy
168
169 Results :
170
171 Failed tests: expectedMessage(sample.actuator.HelloWorldServiceTest): Expected correct message expected:<...om a Docker containe[r]> but was:<...om a Docker containe[]>
172 testRootMessage(sample.actuator.SampleControllerTest): Expected correct message expected:<...om a Docker containe[r]> but was:<...om a Docker containe[]>
173
174 Tests run: 3, Failures: 2, Errors: 0, Skipped: 0
175
176 [INFO] -----
177 [INFO] BUILD FAILURE
178 [INFO] -----
179 [INFO] Total time: 11.120 s
180 [INFO] Finished at: 2022-10-13T19:57:07Z
181 [INFO] -----
182 Error: Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.9:test (default-test) on project spring-boot-sample-actuator: There are test failures.
183 Error:
184 Error: Please refer to /home/runner/work/spring-boot/spring-boot/target/surefire-reports for the individual test results.
185 Error: -> [Help 1]
186 Error:
187 Error: To see the full stack trace of the errors, re-run Maven with the -e switch.
188 Error: Re-run Maven using the -X switch to enable full debug logging.
189 Error:
190 Error: For more information about the errors and possible solutions, please read the following articles:
191 Error: [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
192 Error: Process completed with exit code 1.
```

\_ Y por último corregimos el archivo para que compile correctamente el test y por ende el workflow también:

The screenshot shows a GitHub Actions workflow run for 'Java CI with Maven #2'. The workflow is in a failed state, indicated by a red 'X' icon. The left sidebar shows the workflow steps: Summary, Jobs, and Run details. The 'Jobs' section is expanded, showing a list of jobs: build (failed), Set up job, Run actions/checkout@v2, Set up JDK 8, Build with Maven (failed), Post Set up JDK 8, Post Run actions/checkout@v2, and Complete job. The 'build' job is selected, and its details are shown. The job failed 24 seconds ago in 23s. The logs show the following steps: Set up job (2s), Run actions/checkout@v2 (2s), Set up JDK 8 (1s), Build with Maven (17s), Post Set up JDK 8 (0s), Post Run actions/checkout@v2 (0s), and Complete job (1s). The 'Build with Maven' step is highlighted in red, indicating failure. The logs for this step show the following output:

```
164 at org.apache.maven.surefire.booter.SurefireStarter.invokeProvider(SurefireStarter.java:172)
165 at org.apache.maven.surefire.booter.SurefireStarter.runSuitesInProcessWhenForked(SurefireStarter.java:104)
166 at org.apache.maven.surefire.booter.ForkedBooter.main(ForkedBooter.java:76)
167 2022-10-13 19:57:07.545 INFO 1670 --- [ Thread-2] o.s.w.c.s.GenericWebApplicationContext : Closing org.springframework.web.context.support.GenericWebApplicationContext@565b064f: startup date
[Thu Oct 13 19:57:04 UTC 2022]; root of context hierarchy
168
169 Results :
170
171 Failed tests: expectedMessage(sample.actuator.HelloWorldServiceTest): Expected correct message expected:<...om a Docker containe[r]> but was:<...om a Docker containe[]>
172 testRootMessage(sample.actuator.SampleControllerTest): Expected correct message expected:<...om a Docker containe[r]> but was:<...om a Docker containe[]>
173
174 Tests run: 3, Failures: 2, Errors: 0, Skipped: 0
175
176 [INFO] -----
177 [INFO] BUILD FAILURE
178 [INFO] -----
179 [INFO] Total time: 11.120 s
180 [INFO] Finished at: 2022-10-13T19:57:07Z
181 [INFO] -----
182 Error: Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.9:test (default-test) on project spring-boot-sample-actuator: There are test failures.
183 Error:
184 Error: Please refer to /home/runner/work/spring-boot/spring-boot/target/surefire-reports for the individual test results.
185 Error: -> [Help 1]
186 Error:
187 Error: To see the full stack trace of the errors, re-run Maven with the -e switch.
188 Error: Re-run Maven using the -X switch to enable full debug logging.
189 Error:
190 Error: For more information about the errors and possible solutions, please read the following articles:
191 Error: [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
192 Error: Process completed with exit code 1.
```