JUNIT 5.0 with Maven

Kent Beck

Erich Gamma

Topics

- Test-Driven Development
- Overview of Test-driven Development
- The JUnit Solution
- Test, code, refactor, repeat

Test Driven Development

- TDD starts with developing test for each one of the features.
- The test might fail as the tests are developed even before the development.
- Development team then develops and refactors the code to pass the test.
  - *test-first as part of extreme programming concepts.*

Testing phases

- ***Unit Testing***
  - on individual units of source code (mostly methods)
- ***Integration Testing***
  - on groups of individual software modules
- ***System testing***
  - on a complete end-to-end system

Benefits of TDD

- Much less debug time
- Code proven to meet requirements
- Tests become Safety Net
- Near zero defects
- Shorter development cycles

What is unit testing

- **Unit**
  - A Method , a class, a package, or a subsystem.

- **Can Test :**
  - an entire object
  - part of an object – a method or some interacting methods
  - interaction between several objects

- Helps discover failures in the logic and improve the quality of their code.

- Used to ensure that the code work as expected in case of future changes.

When to Write Test

During Development

- To add new functionality
- When adding new features,

During Debugging

- **While a defect is discovered in code to demonstrates the defect.**

- **write unit tests to *thoroughly test a single class***
- **write tests *as develop* (even before you implement)**
- **write tests for *every new piece of functionality***

JUNIT

Testing Tools -JUNIT

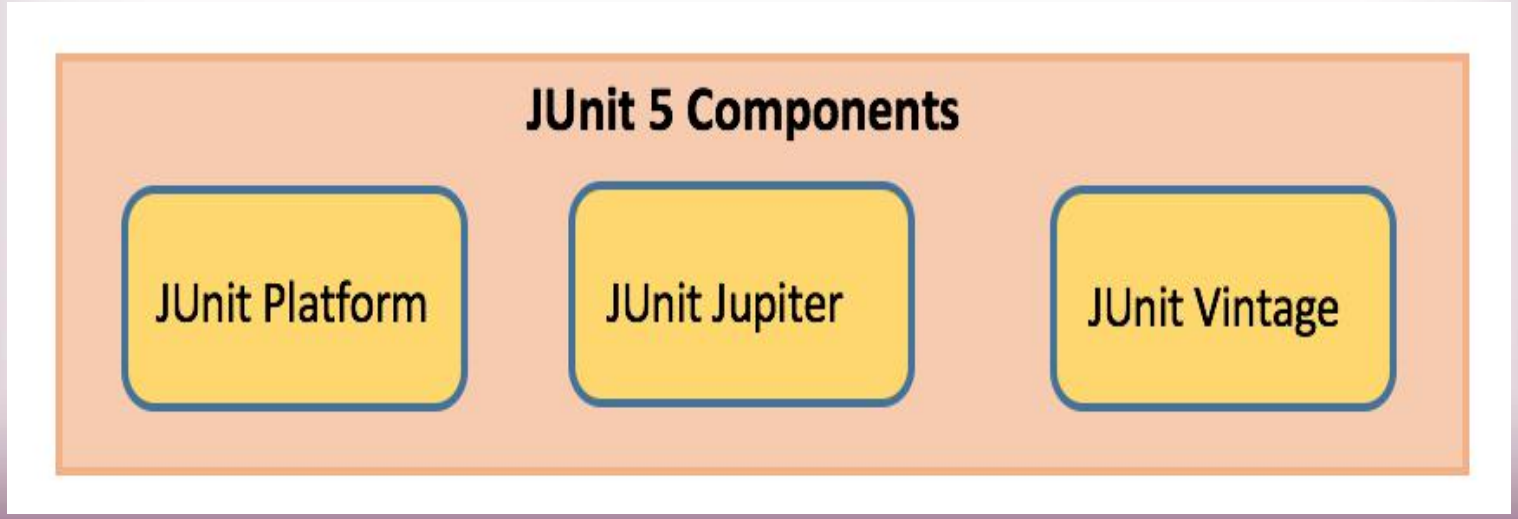
- Open source for automated test ,developed by **Kent Beck** and **Erich Gamma**
- Widely used in industry and can be used as stand alone or within an IDE such as Eclipse.
- Part of a family known as the xUnit
- Linked as a JAR at compile-time and can be used to write repeatable tests

Features

- Assertions for testing expected results.
- Test features for sharing common test data.
- Test suites for easily organizing and running tests.
- Graphical and textual test runners.

Junit 5

- Composed of modules from three different sub-projects.
- Requires Java 8 (or higher) at runtime.
- Can also test code that has been compiled with previous versions of the JDK.



JUnit Platform

- The Foundation for launching testing frameworks on the JVM.
- Defines the API for developing a testing framework that runs on the platform.
- Provides a Console Launcher to launch the platform from the command line
- Platform also exists in popular IDEs

Junit Jupiter

- **JUnit Jupiter**
  - Combination of the new programming model and extension model for writing tests
  - Provides a TestEngine for running Jupiter based tests on the platform.

- **JUnit Vintage**
  - A TestEngine for running JUnit 3 and JUnit 4 based tests on the platform.

Installation

- Download latest version of JUnit jar file from http://www.junit.org.
- Downloaded Jar File is placed in the classpath
- Can Create a Stand Alone Java Application to Execute the tests
- Can Also use Popular IDE Like Eclipse, NetBeans to run tests

Test Classes and Methods

- **Test Class**
  - Top-level class, *Must* ***NOT be abstract***
  - Static member class,
  - @Nested class that contains at least one test method.
  - **Can** declare custom display name with @DisplayName

- **Test Method:**
  - **Instance method** annotated with **@Test**
  - ***Method*** *must* ***be void***
  - Need not be Public but **should not be private**
  - **Can** declare custom display name with @DisplayName
  - Can also have following annotations
    - @RepeatedTest, @ParameterizedTest

Write Unit Test

```
public class Greeting {


public String getMessage() {

    return "Welcome to Java Programming";

 }

}
```

Assertions & Assumptions

- **Assertions**
  - Static methods in **org.junit.jupiter.api.Assertions** class.
  - Used to support asserting conditions in a Test Method
- **Assumptions**
  - Static methods in the **org.junit.jupiter.api.Assumptions** class.
  - Execute a test only when the specified condition met
    - If not met Test will be aborted.
    - The aborted test will not cause build failure.
      - But Throws ***TestAbortedException*** and the test is skipped.

Assertions

- **assertEquals(expected, actual)**
  - Fails when expected does not equal actual
- **assertNull(actual)**
  - Fails when actual is not null
- **assertAll()**
  - Group many assertions
  - Every assertion is executed even if one or more of them fails
- **assertThrows()**
  - Class to be tested is expected to throw an exception

Writing a Junit Test – Assert Equals

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

class TestGreeting {

private  Greeting grtObj =  new Greeting();

@Test
void testGetMessageLength() {

   int actual = grtObj.getMessage().length();

      assertEquals(5,actual);
  }
}
```

Example

```
public String findResult(int mark){

   String result =null;
   if(mark<60)
   {
         result="B";
   }
   if(mark>60 && mark<80)
   {
       result ="C";
   }
   return result;

   }
```

Writing a Junit Test – Assert Not Null

```
@Test
@DisplayName("Test for Method  Should Not Throw  Null Value")
void testFindResultForNotNull() {

    String actual = grtObj.findResult(95);

        assertNotNull(actual);
}
```

Writing a Junit Test – Assert All

```
@Test
@DisplayName("Using Assert All")
void testFindUsingAssertAll() {

assertAll("Testing Cases",
  () -> {
          String expected = grtObj.getMessage();
          assertNotNull(expected);
        },
      () -> assertEquals("Hello World",grtObj.getMessage())
      );
}
  }
```

Example

```
public String checkUserId(String ... values)  {

  String message = "invalid";
    try {

  int id = Integer.parseInt(values[1]);
    message="valid";

} catch (NumberFormatException e) {

      System.err.println(e.getMessage());
}
   return message;
}
```

Writing a Junit Test – Assert Throws

```
@Test
@DisplayName("Test For Number Format Exception ")
void testForException() {
Throwable exception =
        assertThrows(NumberFormatException.class, ()->
            grtObj.checkUserId("fourTwenty"));
    assertEquals("Invalid Number",exception.getMessage());
    }
```

Disabling Tests

- @Disabled
  - Entire test classes or individual test methods can be disabled
  - Can also be declared without providing a reason

```
@Disabled("Disabled until bug #560 has been fixed")
class DisabledClassDemo {

    @Test
    void testWillBeSkipped() {
    }

}
```

Lifecycle Method

- **@BeforeAll**
  - Denotes that the annotated method should be executed once before all Test Methods
  - ***Methods must be static***
  - Used for expensive common operation like database connection or the startup of a server.
- **@AfterAll**
  - Denotes that the annotated method should be executed after all the Test Methods
  - ***Methods must be static***

Lifecycle Method

- **@BeforeEach**
  - Denotes that the annotated method should be executed **before EACH Test Method**
  - To execute some common code before running a test
- **@AfterEach**
  - Denotes that the annotated method should be executed **after EACH Test Method**

Writing a Junit Test Life Cycle Methods

```
@BeforeEach
void setUp(TestInfo info) throws Exception {

System.out.println("BEFORE EACH Called on "+
                info.getDisplayName());
}

@AfterEach
void tearDown(TestInfo info) throws Exception {

System.out.println("AFTER EACH Called on"+
                        info.getDisplayName());

}
```

Example - Timeouts

```
    public String getMessage() {

   try {

   Thread.sleep(5000);

} catch (InterruptedException e) {
       e.printStackTrace();
}
    return  "Welcome to Java Programming";
}
```

Test Timeouts

```
import static java.time.Duration.ofMillis;
import static org.junit.jupiter.api.Assertions.assertTimeout;


    @Test
    @DisplayName("Testing For Timeout")
      void timeoutNotExceededWithMethod() {
        String actualGreeting = assertTimeout(ofMillis(1000),() ->
                                grtObj.getMessage());
        assertEquals("Hello, World!", actualGreeting);
      }
```

ParameterizedTest

- **@ParameterizedTest**
  - Denotes that a method is a parameterized test.
  - When executing the parameterized test method
  - Each invocation will be passed with arguments and reported separately.

- **@ValueSource**
  - Used to pass an array of values to the test method.
  - Values can be int,byte,short ,double,char,String
  - Can Pass Only one Argument to the method.

ParameterizedTest

```
@DisplayName("Testing For elements in even position should not be null")
@ParameterizedTest
@ValueSource(ints = {0,2,4,6})
    void checkingForNullInList(int idxPos) {


assertNotNull(grtObj.findElement(idxPos));
    }
```

ParameterizedTest

```
public String findElement(int idxPos) {


    List<String> names =


Arrays.asList("Ramesh","Suresh",null,"magesh",null,"Rajesh","S
iva");
        return names.get(idxPos);
    }
```

Assumptions

- **assumeTrue**
  - Execute the body of lambda when the positive condition hold else test will be skipped
- **assumeFalse**
  - Execute the body of lambda when the negative condition hold else test will be skipped
- **assumingThat**
  - Flexible, If condition is true then executes
  - Else do not abort test continue rest of code in test.

Assumptions vs Assertions

Assertions

- Assertions is used to write testing scenarios for test methods.
- Assertions fail, the test fails.

Assumptions

- Assumptions are used to validate favorable conditions for test cases.
- Assumptions fails then test method is skipped.

Assumptions -AssumeTrue

```
@DisplayName("Testing For elements in even position should not be null")
@ParameterizedTest
@ValueSource(ints = {0,2,4,6})

    void checkingForNullInList(int idxPos) {

     LocalDateTime dt = LocalDateTime.now();

     assumeTrue(dt.getDayOfWeek().getValue() == 6);

assertNotNull(grtObj.findElement(idxPos));
    }
```

Assumptions -AssumingThat

```
System.setProperty("env", "test");

    assumingThat("test".equals(System.getProperty("os.name")),
        () -> {
            assertEquals(10, 10);
            System.out.println("perform below assertions only on the test env");
            });

    assertEquals(20, 20);
    System.out.println("perform below assertions on all env");
}
```