Skipping tests

The Jupiter annotation @Disabled (located in the package org.junit.jupiter.api) can be used to skip tests. It can be used at class level or method level. The following example uses the annotation @Disabled at method level and therefore it forces to skip the test:

```
class DisabledTest {

    @Disabled
    @Test
    void skippedTest() {
    }

}
```

In this other example, the annotation @Disabled is placed at the class level and therefore all the tests contained in the class will be skipped. Note that a custom message, typically with the reason of the disabling, can be specified within the annotation:

```
@Disabled("All test in this class will be skipped")
class AllDisabledTest {

    @Test
    void skippedTestOne() {
    }

    @Test
    void skippedTestTwo() {
    }

}
```

Display names

JUnit 4 identified tests basically with the name of the method annotated with @Test. This imposes a limitation on name tests, since these names are constrained by the way of declaring methods in Java.

To overcome this problem, Jupiter provides the ability of declaring a custom display name (different to the test name) for tests. This is done with the annotation @DisplayName. This annotation declares a custom display name for a test class or a test method. This name will be displayed by test runners and reporting tools, and it can contain spaces, special characters, and even emojis.

Take a look at the following example. We are annotating the test class, and also the three test methods declared inside the class with a custom test name using @DisplayName:

```
@DisplayName("A special test case")
class DisplayNameTest {

    @Test
    @DisplayName("Custom test name containing spaces")
    void testWithDisplayNameContainingSpaces() {
    }

    @Test
    @DisplayName("(╯°Д°)╯ ")
    void testWithDisplayNameContainingSpecialCharacters() {
    }

    @Test

    @DisplayName("        ")
    void testWithDisplayNameContainingEmoji() {
    }

}
```

Group of assertions

An important Jupiter assertion is assertAll. This method allows to group different assertions at the same time. In a grouped assertion, all assertions are always executed, and any failures will be reported together.

The method assertAll accepts a vargargs of lambda expressions (Executable…) or a stream of those (Stream<Executable>). Optionally, the first parameter of assertAll can be a String message aimed to label the assertion group.

Let's see an example. In the following test, we are grouping a couple of assertEquals using lambda expressions:

```
class GroupedAssertionsTest {

    @Test
    void groupedAssertions() {
        Address address = new Address("John", "Smith");
        // In a grouped assertion all assertions are executed, and any
        // failures will be reported together.
        assertAll("address", () -> assertEquals("John",
        address.getFirstName()),
            () -> assertEquals("User", address.getLastName()));
    }

}
```

Parametrized test

Parameterized tests make it possible to run a test multiple times with different arguments. They are declared just like regular @Test methods but use the @ParameterizedTest annotation instead. In addition, you must declare at least one *source* that will provide the arguments for each invocation and then *consume* the arguments in the test method.

The following example demonstrates a parameterized test that uses the @ValueSource annotation to specify a String array as the source of arguments.

```
@ParameterizedTest
@ValueSource(strings = { "racecar", "radar", "able was I ere I saw elba" })
void palindromes(String candidate) {
    assertTrue(StringUtils.isPalindrome(candidate));
```

```
}
```

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

import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

class param {
	@ParameterizedTest
	@ValueSource(ints = { 1, 2, 3 })
	void test(int argument) {
		assertTrue(argument > 0 && argument < 4);
	}
	}

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

class testme {
@ParameterizedTest
@ValueSource(strings= {"1","2","3"})

	void test(String  argument) {
	assertTrue(argument.length()>0);
```

```
        }


}
```

Exercise:

Draw the CFG for the following code, convert it to java code and write unit tests. Give display name and use group assertions.

```
1 scanf("%d %d",&x, &y);
2 if (y < 0)
        pow = -y;
        else
        pow = y;
3 z = 1.0;
4 while (pow != 0) {
        z = z * x;
        pow = pow - 1;
5 }
6 if (y < 0)
        z = 1.0 / z;
7 printf ("%f",z)
```

2. Write a java program that accepts students' names, id, mid and final exam results. Calculate the average and determine grades. Store students' names and id into different data structure based on their grades. Write a unit test for your program.