JUnit 5 Assertions

- assertEquals(Object expected, Object actual);
  - Uses the object's equals method (so safe for Strings to compare content)
  - If used on strings will show a visible difference by clicking on the top failure trace

- assertNotEquals(Object unexpected, Object actual)
  - Use when you are not hardcoding a difference to check with equals. For instance, testing an equals method with two non-equal objects.

- assertArrayEquals(array expected, array actual)
  - Makes sure two arrays are equal. If the array contains objects, makes sure that they are "deeply equal", ie the objects in both arrays have the same memory location.

- assertTrue(boolean condition) and assertFalse(boolean condition)
  - Useful for ranges, like checking compare is simply less than or greater than 0 and not caring about a specific value
  - Also useful for just actually testing if booleans are the expected value

- assertNull(Object) and assertNotNull(Object)
  - Useful for checking if Object fields are set to something without necessarily caring about the specific value of that object.

- assertThrows(NameOfException.class, () -> theMethodThatShouldThrowException()));
  - First argument is the exception you are expecting with .class on the end, such as NumberFormatException.class
  - The middle thing is a lambda expression
  - To the right of the lambda is the action you expect to throw an exception.
  - assertThrows returns a Throwable, so you can capture that with Throwable exception = assertThrows(...). You can then use assertEquals("Message expected", exception.getMessage()); to check if the message is what is expected
    - This is useful for testing that a method or constructor calls the super class if that class throws an exception with a descriptive message.

- assertSame(Object expected, Object actual)
  - asserts two objects have the same memory location

- assertAll(lambda stuff, can get complicated)
  - Asserts that everything is executed

- assertTimeout(Durantion timeout, Executable executable) and assertTimeoutPreemptively
  - Asserts that something happens within a given time frame
  - Preemptive choice means it will abort and not finish if timeout is exceeded

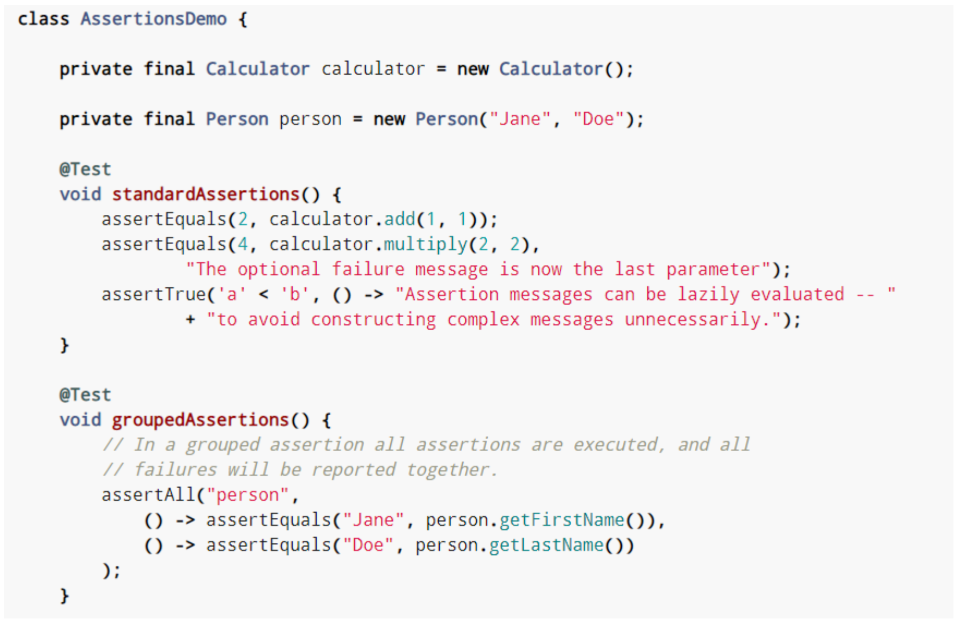
- fail(message or throwable)
  - Will fail the test with a custom message if reached. Useful for try / catch kind of stuff

Why there are so many

- All assertions have 3 forms
  - normal parameters (above) with default message (usually AssertionError or ComparisonFailure)
  - normal parameters followed by a custom String message
    - assertEquals(Object expected, Object actual, “Test message”) will replace AssertionError etc with custom message. Will still include actual differences, like expected 110 actual 0.
  - Normal parameters followed by a message Supplier

- assertEquals works on byte, char, double, float, int, long, Object, and short. Float and double have the option to provide a delta which allows the comparisons to be off by some amount and still return true. Each of these also has the 3 forms above. Same for assertArrayEquals.

Examples stolen from https://junit.org/junit5/docs/current/user-guide/#writing-tests-assertions



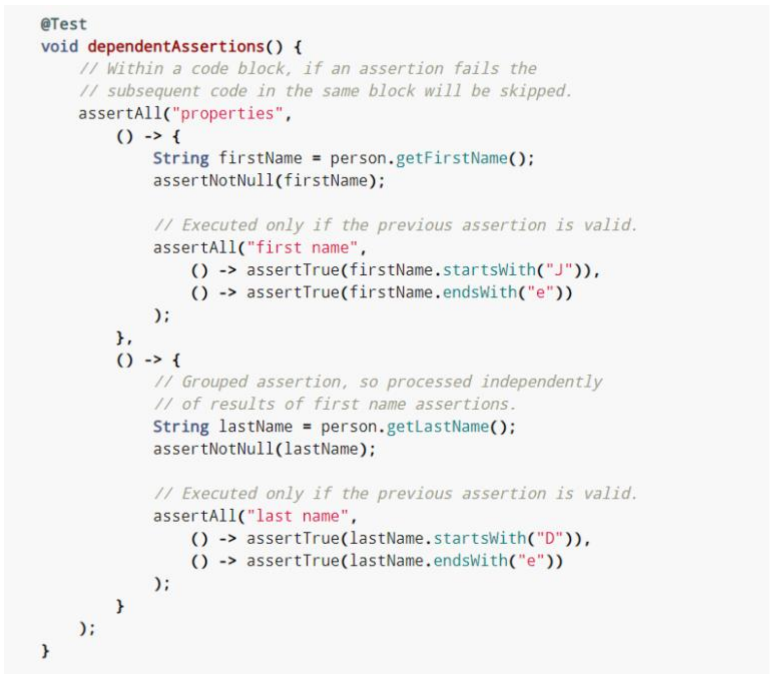
```
class AssertionsDemo {

    private final Calculator calculator = new Calculator();

    private final Person person = new Person("Jane", "Doe");

    @Test
    void standardAssertions() {
        assertEquals(2, calculator.add(1, 1));
        assertEquals(4, calculator.multiply(2, 2),
                "The optional failure message is now the last parameter");
        assertTrue('a' < 'b', () -> "Assertion messages can be lazily evaluated -- "
                + "to avoid constructing complex messages unnecessarily.");
    }

    @Test
    void groupedAssertions() {
        // In a grouped assertion all assertions are executed, and all
        // failures will be reported together.
        assertAll("person",
            () -> assertEquals("Jane", person.getFirstName()),
            () -> assertEquals("Doe", person.getLastName())
        );
    }
```



```
@Test
void dependentAssertions() {
    // Within a code block, if an assertion fails the
    // subsequent code in the same block will be skipped.
    assertAll("properties",
        () -> {
            String firstName = person.getFirstName();
            assertNotNull(firstName);

            // Executed only if the previous assertion is valid.
            assertAll("first name",
                () -> assertTrue(firstName.startsWith("J")),
                () -> assertTrue(firstName.endsWith("e"))
            );
        },
        () -> {
            // Grouped assertion, so processed independently
            // of results of first name assertions.
            String lastName = person.getLastName();
            assertNotNull(lastName);

            // Executed only if the previous assertion is valid.
            assertAll("last name",
                () -> assertTrue(lastName.startsWith("D")),
                () -> assertTrue(lastName.endsWith("e"))
            );
        }
    );
}
```

```java
@Test
void exceptionTesting() {
    Exception exception = assertThrows(ArithmeticException.class, () ->
        calculator.divide(1, 0));
    assertEquals("/ by zero", exception.getMessage());
}

@Test
void timeoutNotExceeded() {
    // The following assertion succeeds.
    assertTimeout(ofMinutes(2), () -> {
        // Perform task that takes less than 2 minutes.
    });
}

@Test
void timeoutNotExceededWithResult() {
    // The following assertion succeeds, and returns the supplied object.
    String actualResult = assertTimeout(ofMinutes(2), () -> {
        return "a result";
    });
    assertEquals("a result", actualResult);
}

@Test
void timeoutNotExceededWithMethod() {
    // The following assertion invokes a method reference and returns an object.
    String actualGreeting = assertTimeout(ofMinutes(2), AssertionsDemo::greeting);
    assertEquals("Hello, World!", actualGreeting);
}
```

```java
    @Test
    void timeoutExceeded() {
        // The following assertion fails with an error message similar to:
        // execution exceeded timeout of 10 ms by 91 ms
        assertTimeout(ofMillis(10), () -> {
            // Simulate task that takes more than 10 ms.
            Thread.sleep(100);
        });
    }

    @Test
    void timeoutExceededWithPreemptiveTermination() {
        // The following assertion fails with an error message similar to:
        // execution timed out after 10 ms
        assertTimeoutPreemptively(ofMillis(10), () -> {
            // Simulate task that takes more than 10 ms.
            new CountDownLatch(1).await();
        });
    }

    private static String greeting() {
        return "Hello, World!";
    }

}
```