S04 Test-Driven Development

Test-driven development (TDD)
JUnit

Types of Tests

- **Unit tests**: testing individual “units” / “modules”
  - In OO a unit is at the level of a **method** or **class**
  - Check the “building blocks” are functioning correctly
- **Integration tests**: the integration of multiple modules
  - Expose problems with interface of modules and interactions between them
- **System tests**: end-to-end complete system
  - Checking it meets its requirements

Test Driven Development (TDD)

TDD “red, green, refactor”

1. Create test that defines new requirements
2. Ensure test fails
3. Write code to support new requirement
4. Run tests to ensure code is *correct****
5. Then refactor and improve
6. Repeat

Key element of *agile programming*

What Makes **Good** Unit Tests?

- Isolate behaviour / reduce dependencies
- Common path / usage
- Edge cases
- Touch on all branches
- Deterministic
- Limit false positives (test fails for correct code)
- Coverage

JUnit

Unit testing for Java

- Developed by Kent Beck
  - Father of extreme programming movement
- Integrated into IntelliJ
- Useful for:
  - TDD (Test driven development)
  - Bug isolation and regression testing
    - Precisely identify the bug with a unit test
    - Use test to ensure that the bug is not reintroduced

JUnit

- Methods marked with `@Test` will be tested
- When JUnit is called on a class, all tests are run and a report is generated (a failed test does not stop execution of subsequent tests).
- JUnit has a rich set of annotations that can be used to configure the testing environment, including:
- `@Test`, `@Ignore`, `@BeforeEach`, `@BeforeClass`, `@AfterEach`, `@AfterClass`, `@Timeout`
- JUnit can check that an exception is thrown if that is expected in a certain case
  -
```
Assertions.assertThrows(
    ArithmeticException.class,
    () -> myMethod());
```