Report Prerequisites

Exercise 1

```
/**
 * Do we get the correct delta when moving south?
 */
@Test
void testSouth() {
    Direction south = Direction.valueOf("SOUTH");
    assertThat(south.getDeltaY()).isEqualTo(1);
}

/**
 * Do we get the correct delta when moving west?
 */
@Test
void testWest() {
    Direction west = Direction.valueOf("WEST");
    assertThat(west.getDeltaX()).isEqualTo(-1);
}

/**
 * Do we get the correct delta when moving east?
 */
@Test
void testEast() {
    Direction east = Direction.valueOf("EAST");
    assertThat(east.getDeltaX()).isEqualTo(1);
}
```

Exercise 2

2.7

```
/**
 * Resets the unit under test.
 */
@BeforeEach
void setUp() {
    unit = new BasicUnit();
    square = new BasicSquare();
}

/**
 * Asserts that a unit has no square to start with.
 */
@Test
void noStartSquare() {
    Assertions.assertFalse(unit.hasSquare());
}

/**
 * Tests that the unit indeed has the target square as its base after
 * occupation.
 */
@Test
void testOccupy() {
   unit.occupy(square);
   assertThat(unit.getSquare()).isEqualTo(square);
}

/**
 * Test that the unit indeed has the target square as its base after
 * double occupation.
 */
@Test
void testReoccupy() {
    unit.occupy(square);
    Assertions.assertTrue(unit.hasSquare());
    unit.leaveSquare();
    Assertions.assertFalse(unit.hasSquare());
    unit.occupy(square);
    assertThat(unit.getSquare()).isEqualTo(square);
}
```

2.8

The invariant checks if the board contains a square that is null. The test for the squareAt method returns the exception "Initial grid cannot contain null square".

```
@Test
    public void gridTest() {
        Square[][] sg = new Square[1][1];
        sg[0][0] = new BasicSquare();
        board = new Board(sg);

        assertEquals(board.getHeight(), 1);
        assertEquals(board.getWidth(), 1);
    }
```

Testing SquareAt method:

```
@Test
    public void squareAtTest() {
        Square[][] sg = new Square[1][1];
        sg[0][0] = null;
        board = new Board(sg);

        assertEquals(board.squareAt(0,0), null);
    }
```

`-ea` Stands for enable assertions. By leaving out this tag assertions errors won't show up because assertions that are part of the code will be ignored

GitLab has assertions checking enabled because the testing stage fails for that particular test

2.9

- `<module name="LineLength">` A line may only contain 100 characters or less
- `<module name="MethodName">` The name of a method must obey the following RegEx `^[a-z][a-zA-Z0-9_]*$`
- `<module name="TodoComment">` It checks if a comment contains TODO and highlight it if so

Exercise 3

We can't exhaustively test our entire software project, because that would mean that would mean that we have to create an enormous amount of test which would be greater than our lifetime. Instead of that we should test many different aspects of the project.

Exercise 4

Pesticide paradox is about the more you test your software, the more it becomes immune to those tests. For software testers it implies they should create more different kinds of tests, and also change and improve those tests by using different techniques and methods.

Exercise 5

We should automate tests as much as possible because it is less time consuming as the tests can be executed multiple times, it frees up the time of the testers so they can focus more on interesting scenarios and exploratory testing. Secondly continuous integration provides a quick feedback on your application.