학습 테스트 진행 방식

- import한 저장소의 src/test/java 학습 테스트를 위한 패키지를 생성한다.
    - 예를 들어 study
- 단위 테스트를 위한 클래스 명명 규칙("테스트 대상 클래스명 + Test")에 따라 클래스를 생성한다.
    - 예를 들어 String에 대한 학습 테스트 클래스는 StringTest
- 다음 요구사항을 구현하면서 JUnit과 AssertJ 사용법을 익힌다.

String 클래스에 대한 학습 테스트

요구사항 1

- "1,2"을 `,`로 split 했을 때 1과 2로 잘 분리되는지 확인하는 학습 테스트를 구현한다.
- "1"을 `,`로 split 했을 때 1만을 포함하는 배열이 반환되는지에 대한 학습 테스트를 구현한다.

힌트

- 배열로 반환하는 값의 경우 assertj의 contains()를 활용해 반환 값이 맞는지 검증한다.
- 배열로 반환하는 값의 경우 assertj의 containsExactly()를 활용해 반환 값이 맞는지 검증한다.

요구사항 2

- "(1,2)" 값이 주어졌을 때 String의 substring() 메소드를 활용해 `()` 을 제거하고 "1,2"를 반환하도록 구현한다.

요구사항 3

- "abc" 값이 주어졌을 때 String의 charAt() 메소드를 활용해 특정 위치의 문자를 가져오는 학습 테스트를 구현한다.
- String의 charAt() 메소드를 활용해 특정 위치의 문자를 가져올 때 위치 값을 벗어나면 StringIndexOutOfBoundsException이 발생하는 부분에 대한 학습 테스트를 구현한다.
- JUnit의 @DisplayName을 활용해 테스트 메소드의 의도를 드러낸다.

힌트

- AssertJ Exception Assertions 문서 참고

```
import static org.assertj.core.api.Assertions.*;

assertThatThrownBy(() -> {
    // ...
}).isInstanceOf(IndexOutOfBoundsException.class)
  .hasMessageContaining("Index: 2, Size: 2");
```

```
import static org.assertj.core.api.Assertions.assertThatExceptionOfType;

assertThatExceptionOfType(IndexOutOfBoundsException.class)
  .isThrownBy(() -> {
      // ...
}).withMessageMatching("Index: \\d+, Size: \\d+");
```

- 자주 발생하는 Exception의 경우 Exception별 메서드 제공하고 있음
  - assertThatIllegalArgumentException()
  - assertThatIllegalStateException()
  - assertThatIOException()
  - assertThatNullPointerException()

Set Collection에 대한 학습 테스트

- 다음과 같은 Set 데이터가 주어졌을 때 요구사항을 만족해야 한다.

```
public class SetTest {
    private Set<Integer> numbers;

    @BeforeEach
    void setUp() {
        numbers = new HashSet<>();
        numbers.add(1);
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
    }

    // Test Case 구현
}
```

요구사항 1

- Set의 size() 메소드를 활용해 Set의 크기를 확인하는 학습테스트를 구현한다.

요구사항 2

- Set의 contains() 메소드를 활용해 1, 2, 3의 값이 존재하는지를 확인하는 학습테스트를 구현하려한다.
- 구현하고 보니 다음과 같이 중복 코드가 계속해서 발생한다.
- JUnit의 ParameterizedTest를 활용해 중복 코드를 제거해 본다.

```
    @Test
    void contains() {
        assertThat(numbers.contains(1)).isTrue();
        assertThat(numbers.contains(2)).isTrue();
        assertThat(numbers.contains(3)).isTrue();
    }
```

힌트

- Guide to JUnit 5 Parameterized Tests

```
@ParameterizedTest
@ValueSource(strings = {"", "  "})
void isBlank_ShouldReturnTrueForNullOrBlankStrings(String input) {
    assertTrue(Strings.isBlank(input));
}
```

요구사항 3

- 요구사항 2는 contains 메소드 결과 값이 true인 경우만 테스트 가능하다. 입력 값에 따라 결과 값이 다른 경우에 대한 테스트도 가능하도록 구현한다.
- 예를 들어 1, 2, 3 값은 contains 메소드 실행결과 true, 4, 5 값을 넣으면 false 가 반환되는 테스트를 하나의 Test Case로 구현한다.

힌트

- Guide to JUnit 5 Parameterized Tests 문서에서 @CsvSource를 활용한다.

```
@ParameterizedTest
@CsvSource(value = {"test:test", "tEst:test", "Java:java"}, delimiter = ':')
void toLowerCase_ShouldGenerateTheExpectedLowercaseValue(String input, String expected) {
    String actualValue = input.toLowerCase();
    assertEquals(expected, actualValue);
}
```

assertj 활용

- Introduction to AssertJ 문서 참고해 assertj의 다양한 활용법 익힌다.