Модульное тестирование JUnit

V-модель разработки

Стандартным подходом к созданию программных продуктов является т. н. V-модель.

Она разделяет всю работу на этапы и позволяет провести проверку каждого из них.

Для этого применяются разные виды тестирования.

Тестовый сценарий

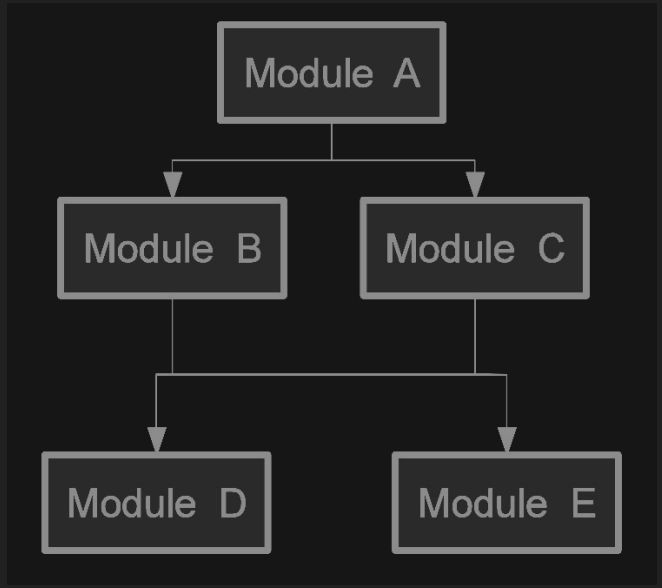
№	Начальное состояние	Ввод	Действие системы	Вывод	Конечное состояние
1	Готов	Пользователь вставляет карточку	Успешное чтение карточки	Приглашение “введите pin”	Ожидание pin-кода
2	Ожидание pin-кода	Вводим **верный** pin-код	Проверка pin-кода	Приглашение к выбору транзакции	Ожидание выбора транзацкии
3	Ожидание выбора транзацкии	Выбор выдачи 5000 euro	Проверка баланса, возможности выдачи	Деньги	Выдача денег
4	Выдача денег	Пользователь берет деньги и карточку	Завершение выдачи	Благодарность за использование	Готов

Тестовый сценарий состоит из тестовых случаев (тест кейсов).

Тестовый случай состоит из входных данных (значение или действие), предусловия, условия выполнения, постусловия и ожидаемый результат (должен быть определен до запуска теста).

Тестовый случай должен быть повторяемым и автоматизируемым. Он должен учитывать состояния (при наличии, например переходы между состояниями). Тестовый сценарий описывает типичное использование системы.

Модуль и unit testing



Модульное тестирование - тестирование отдельным компонентов программы: метод, класс, программный модуль.

Для тестирования модуля надо изолировать его от остальной части программы/системы.

Плюсы: можно тестировать отдельно взятую часть программы не ожидая завершения остальных компонентов программы, модульные тесты отображают идеальный сценарий использования, выявление проблем на более ранних этапах

Минусы: не все проблемы в коде можно выявить модульными тестами (проблема недостаточного покрытия и ограниченность одним модулем), нужно уделить время для изоляции модулей друг от друга, изменение существующего кода требуют изменения тестов.

JUnit:

JUnit - простой фреймворк для реализации модульного тестирования в Java.

Поставляется в виде jar-файла, то есть в виде библиотеки.

Позволяет писать модульные тесты более наглядно и быстро, предоставляя в то же время дополнительные функции, например анализ покрытия (coverage).

Coverage утверждение насколько полное покрытие тестами у вашего кода (основывается на проверке был ли вызвал тот или иной блок вашего кода в ходе тестирования или нет).

Подключение junit: тут

Junit: аннотации и методы

```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions;

public class CarArrayListTest {
   private Car car1;
   private Car car2;

   @BeforeEach
   public void setUp() {
      this.car1 = new Car("bmw", 1915);
      this.car2 = new Car("polo", 1915);
   }

   @Test
   public void testCount() {
       CarArrayList garage = new CarArrayList();
       Assertions.assertEquals(0, garage.count);

       garage.add(this.car1);
       garage.add(this.car2);

       Assertions.assertEquals(2, garage.count);
   }
}
```

`@Test` - аннотация для обозначения метода-теста

`@BeforeEach` - аннотация для обозначения метода вызываемого перед каждым тестом

`@AfterEach` - аннотация для обозначения метода вызываемого после каждого теста

`@BeforeClass` - аннотация для обозначения метода вызываемого перед всеми другими методами класса тестирования

`@AfterClass` - аннотация для обозначения метода вызываемого после всех других методов класса тестирования

`Assertions.assertEquals` - метод для проверки равенства одного объекта другому

Домашнее задание

Написать тест (см. предыдущий слайд или Pokemon.java и PokemonTest.java) для нескольких методов какого нибудь вашего класса из последних домашек (CurrencyConverter или Pokemon или 17 домашка с разными объектами)

отчетность: гитхаб + скриншот с успешным запуском с отображением тестового покрытия

по желанию №1: используйте по возможности все рассмотренные аннотации
по желанию №2: напишите тесты с покрытием не меньше 80% вашего проекта (можно выбрать любой)

The end