Тестирование в разработке
на примере небольших приложений

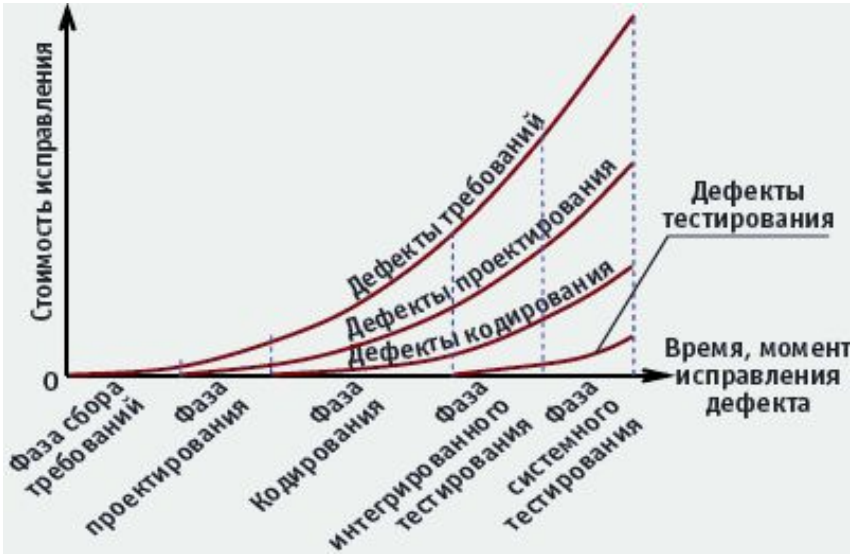
Подготовил:
Студент гр. 550504 Стрельцов Г.Ю.
Руководитель:
Старший преподаватель Искра Н.А.

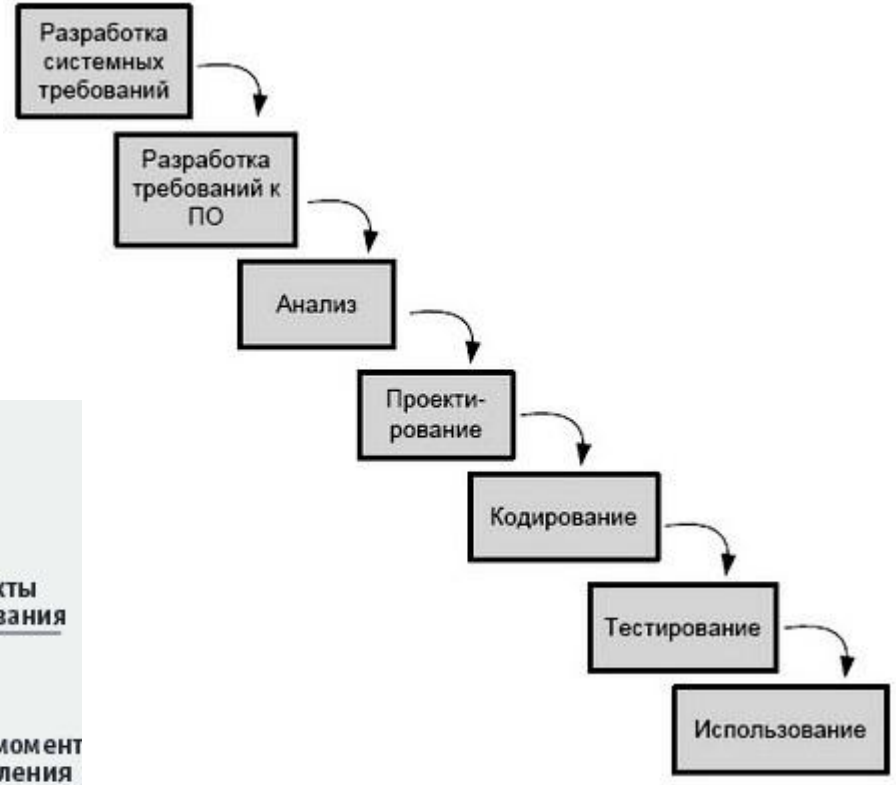
План:

1. Введение
2. Типы тестов для разработчиков
3. Основные стратегии
4. Базовый арсенал
5. JUnit
6. Mockito
7. Примеры
8. Преимущества подхода TDD
9. Недостатки подхода TDD
10. Вывод

Зачем?

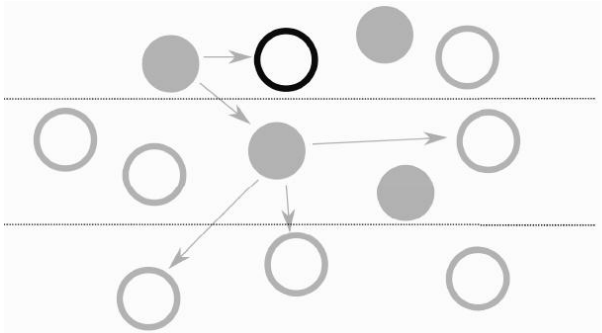
- Быстрее
- Дешевле
- Качественнее

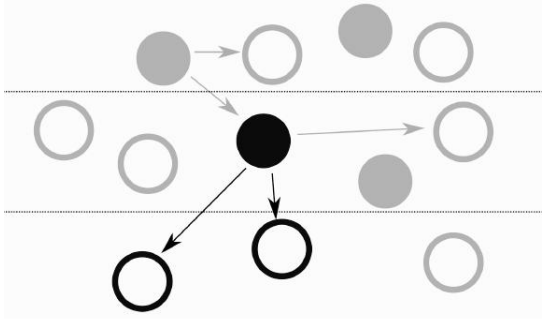


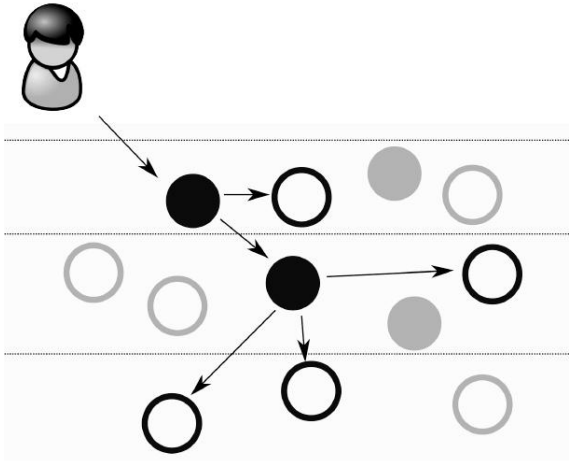


Типы тестов для разработчиков

1. Модульные тесты (Unit tests)
2. Интеграционные тесты (Integration tests)
3. Сквозные тесты (End-to-end tests)

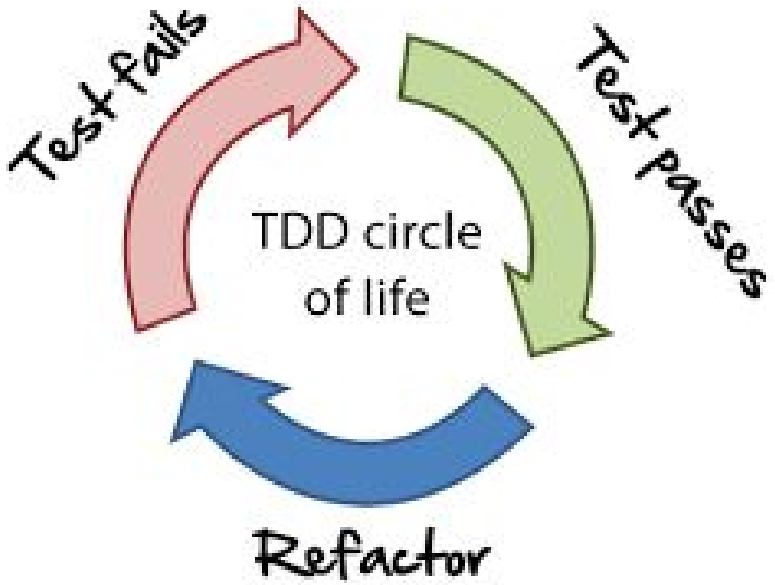






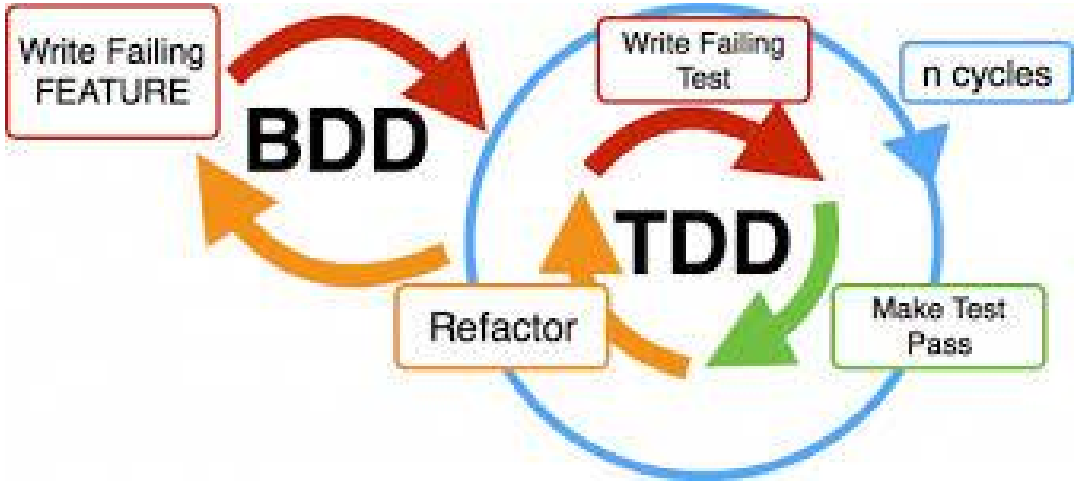
Основные стратегии

TDD (Test-driven development)



Основные стратегии

BDD (Behavior-driven development)



Базовый арсенал

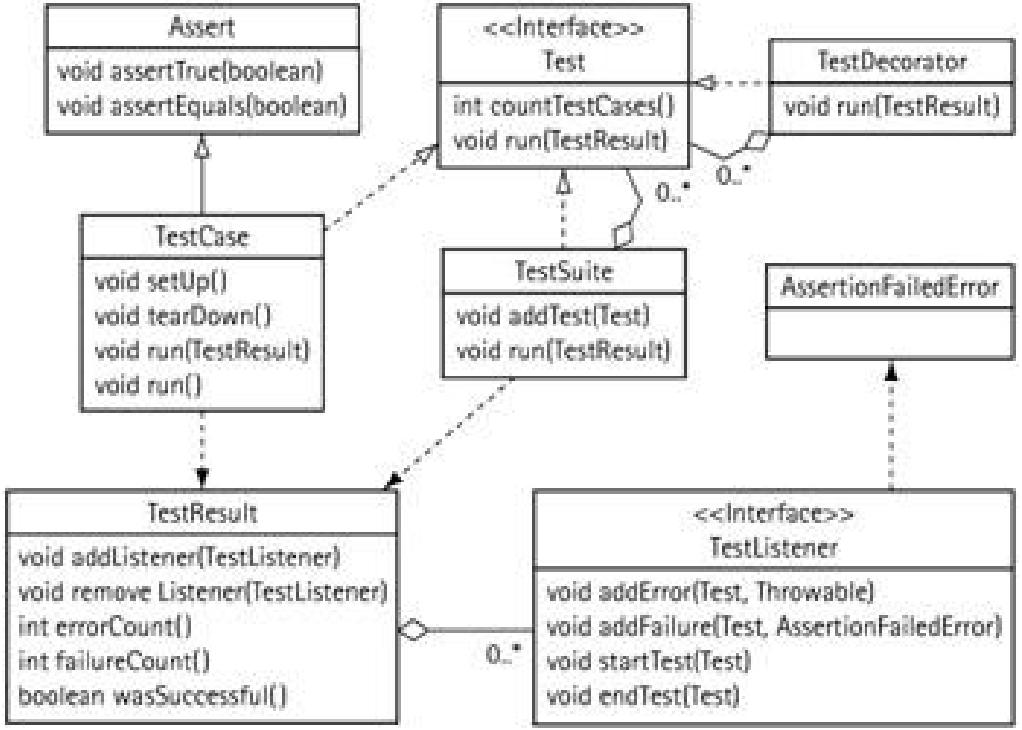
- JAVA
- JUNIT
- MOCKITO
- Build tools (Maven, Gradle…)

Другие инструменты

- EasyTest
- JUnitParams
- Hamcrest
- FEST
- EasyMock
- PowerMock

JUNIT

http://junit.org



Доступные аннотации JUnit

Аннотация	Описание
@Test public void method()	Аннотация **@Test** определяет что метод method() является тестовым.
@Before public void method()	Аннотация **@Before указывает** на то, что метод будет выполняться перед каждым тестируемым методом @Test.
@After public void method()	Аннотация **@After указывает** на то, что метод будет выполняться после каждого тестируемого метода @Test
@BeforeClass public static void method()	Аннотация **@BeforeClass** указывает на то, что метод будет выполняться в начале всех тестов, а точней в момент запуска тестов(перед всеми тестами @Test).
@AfterClass public static void method()	Аннотация **@AfterClass** указывает на то, что метод будет выполнятся после всех тестов.
@Ignore public static void method()	Аннотация **@Ignore** говорит, что метод будет проигнорирован в момент проведения тестирования.
@Test (**expected** = Exception.class) public static void method()	(**expected** = Exception.class) — указывает на то, что в данном тестовом методе вы преднамеренно ожидается Exception.
@Test (**timeout**=100) public static void method()	(**timeout**=100) — указывает, что тестируемый метод не должен занимать больше чем 100 миллисекунд.

Проверяющие методы (Основные)

Метод	Описание
fail(String)	Указывает на то, чтобы тестовый метод завалился при этом выводя текстовое сообщение.
assertTrue([message], boolean condition)	Проверяет, что логическое условие истинно.
assertEquals([String message], expected, actual)	Проверяет, что два значения совпадают. *Примечание*: для массивов проверяются ссылки, а не содержание массивов.
assertNull([message], object)	Проверяет, что объект является пустым **null**.
assertNotNull([message], object)	Проверяет, что объект не является пустым **null**.
assertSame([String], expected, actual)	Проверяет, что обе переменные относятся к одному объекту.
assertNotSame([String], expected, actual)	Проверяет, что обе переменные относятся к разным объектам.

JUNIT

Достоинства:

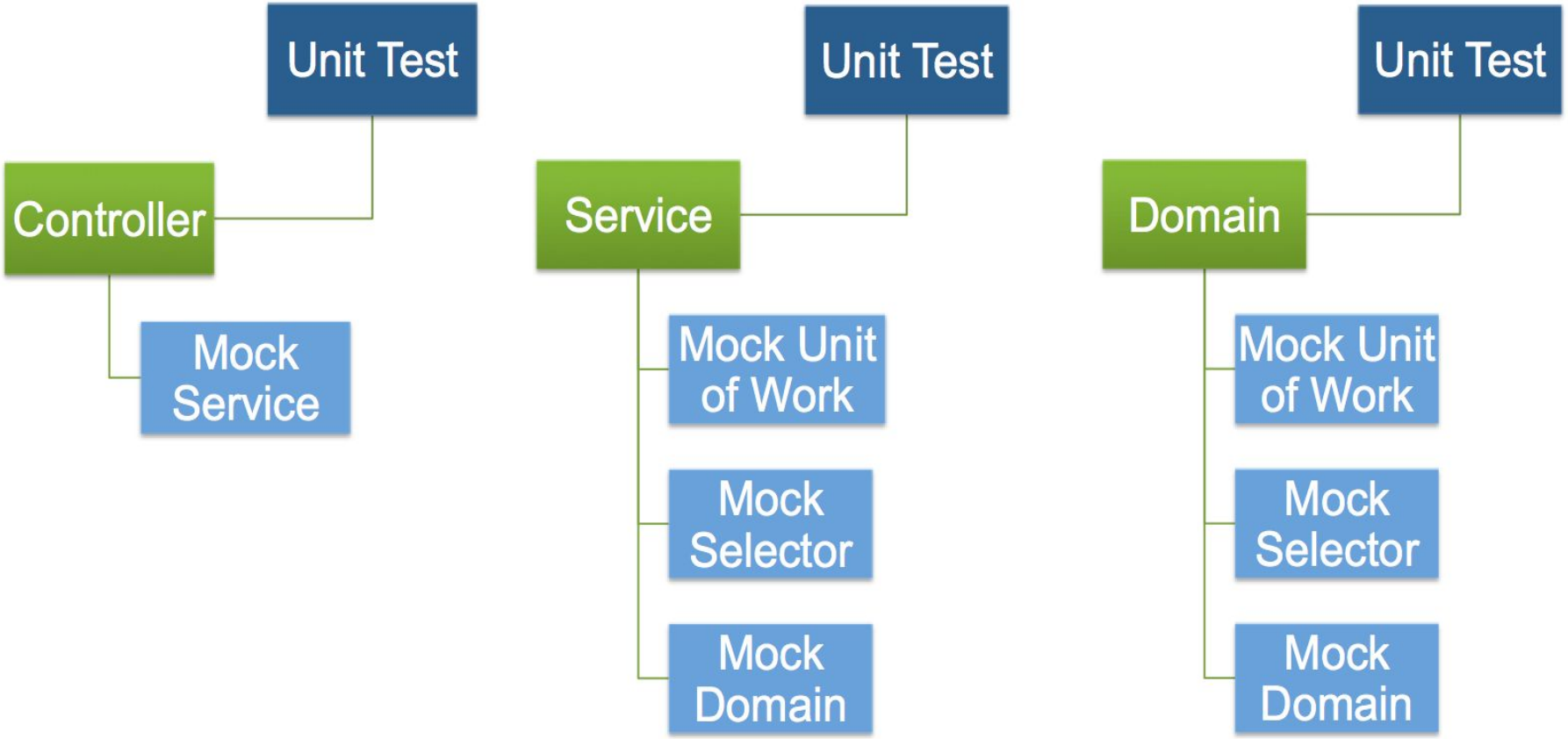
- Простота
- Удобство написания модульных тестов

Недостатки:

- Ограниченная поддержка параметризированных тестов (дополнительное использование JUnitParams)

Mockito

http://site.mockito.org



Mockito

Достоинства:

- Простота
- Отличная документация
- Ряд полезных особенностей (имитирует не только конкретные классы, но и интерфейсы)
- Очень легко и удобно использовать в связке с JUnit

Недостатки:

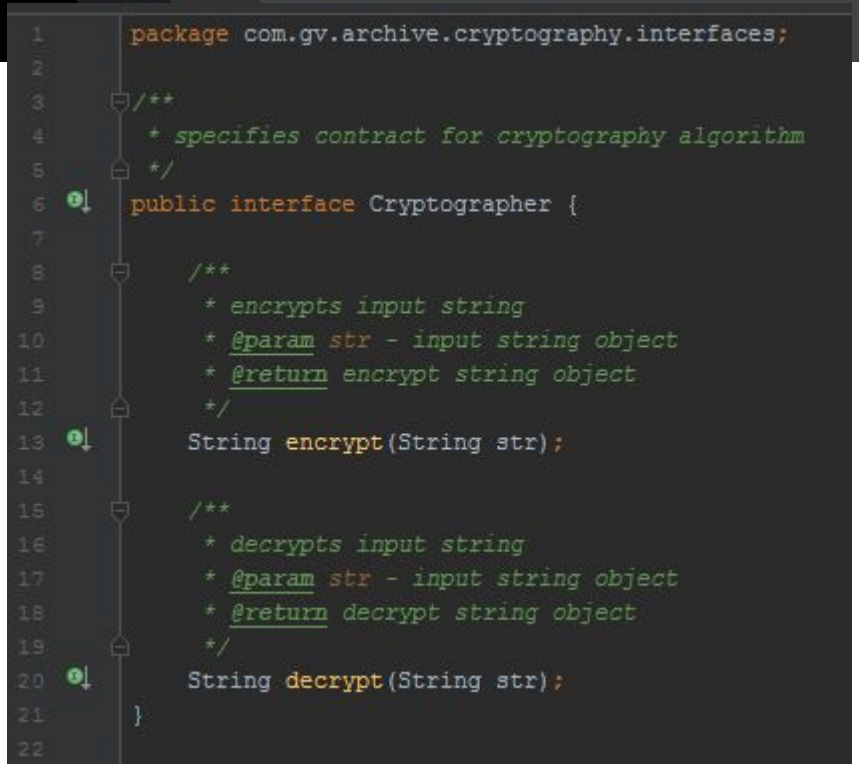
- с точки зрения использования пока не выявлено

Примеры

1) Тестирование бизнес-процессов
2) Тестирование взаимодействия с хранилищем данных

Тестирование бизнес-процессов

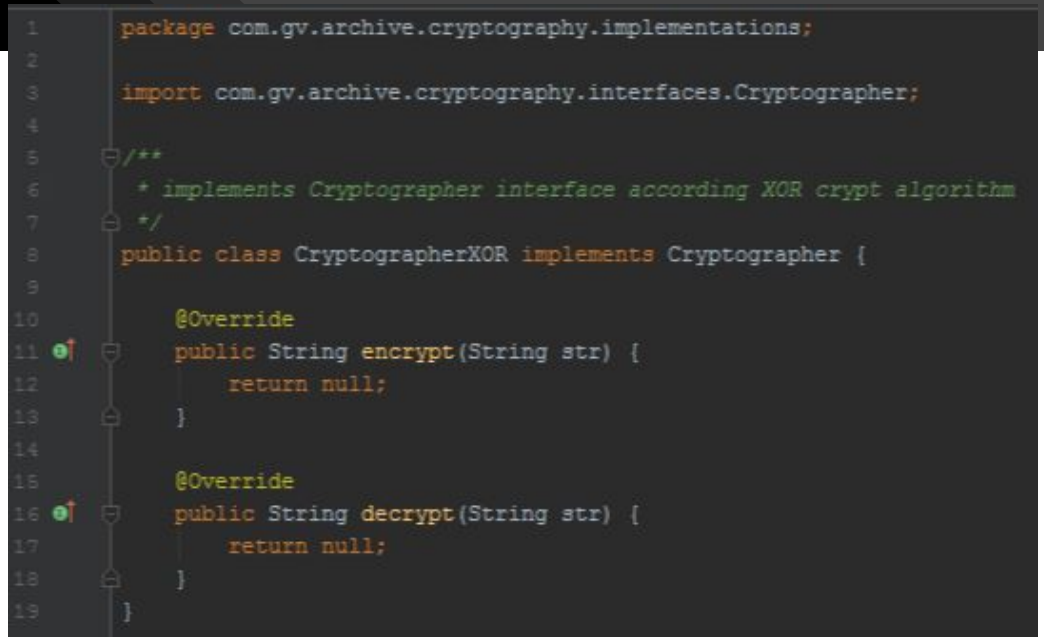
1. Разработка архитектуры разрабатываемого модуля с использованием интерфейса



```java
package com.gv.archive.cryptography.interfaces;

/**
 * specifies contract for cryptography algorithm
 */
public interface Cryptographer {

    /**
     * encrypts input string
     * @param str - input string object
     * @return encrypt string object
     */
    String encrypt(String str);

    /**
     * decrypts input string
     * @param str - input string object
     * @return decrypt string object
     */
    String decrypt(String str);
}
```

Тестирование бизнес-процессов

2. Создание сигнатуры конкретной реализации интерфейса

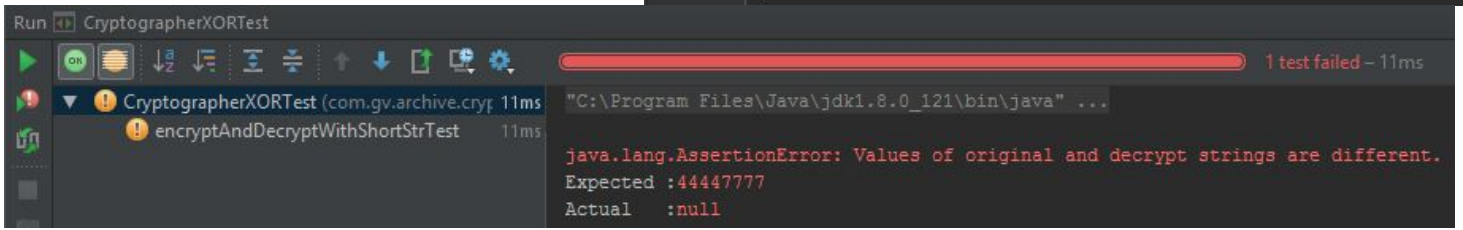


```java
package com.gv.archive.cryptography.implementations;

import com.gv.archive.cryptography.interfaces.Cryptographer;

/**
 * implements Cryptographer interface according XOR crypt algorithm
 */
public class CryptographerXOR implements Cryptographer {

    @Override
    public String encrypt(String str) {
        return null;
    }

    @Override
    public String decrypt(String str) {
        return null;
    }
}
```

Тестирование бизнес-процессов

3. Написание тестов

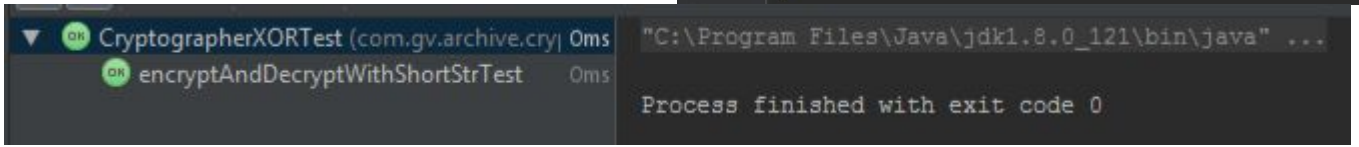
```java
package com.gv.archive.cryptography.implementations;

import com.gv.archive.cryptography.interfaces.Cryptographer;
import org.junit.Assert;
import org.junit.Test;

public class CryptographerXORTest {

    private Cryptographer cryptographer = new CryptographerXOR();
    private String testStr;
    private String encryptStr;

    @Test
    public void encryptAndDecryptWithShortStrTest() throws Exception {
        testStr = "44447777";
        encryptStr = cryptographer.encrypt(testStr);
        String errorMessage = "Values of original and decrypt strings are different.";
        Assert.assertEquals(errorMessage ,testStr, cryptographer.decrypt(encryptStr));
    }
}
```



Тестирование бизнес-процессов

4. Доработка методов тестируемого класса для прохождения тестов

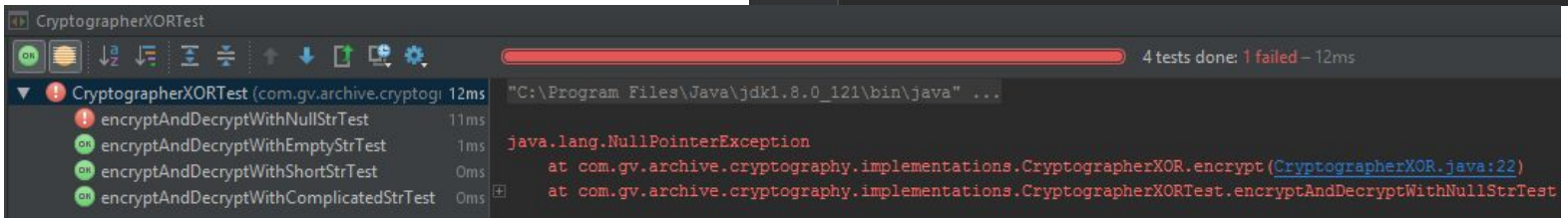
```java
public class CryptographerXOR implements Cryptographer {

    /** object for extracting properties from resource bundle cryptoKey.properties */
    private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle.getBundle("cryptoKey");

    /** name of property in resource bundle that correspond to crypto key */
    private static final String KEY_PROPERTY_NAME = "key";

    @Override
    public String encrypt(String str) {
        byte[] key = RESOURCE_BUNDLE.getString(KEY_PROPERTY_NAME).getBytes();
        byte[] text = str.getBytes();
        byte[] result = new byte[str.length()];
        for(int i = 0; i < text.length; i++){
            result[i] = (byte)(text[i] ^ key[i % key.length]);
        }
        return new String(result);
    }

    @Override
    public String decrypt(String str) {
        return encrypt(str);
    }
}
```



Тестирование бизнес-процессов

5. Больше тестов и доработка кода

```
    @Test
    public void encryptAndDecryptWithEmptyStrTest() throws Exception {
        testStr = "";
        encryptStr = cryptographer.encrypt(testStr);
        String errorMessage = "Result string after decrypting is not empty as original.";
        Assert.assertEquals(errorMessage, testStr, cryptographer.decrypt(encryptStr));
    }

    @Test
    public void encryptAndDecryptWithNullStrTest() throws Exception {
        testStr = null;
        encryptStr = cryptographer.encrypt(testStr);
        String errorMessage = "Result result of encrypting null must be also null.";
        Assert.assertNull(errorMessage, encryptStr);
    }

    @Test
    public void encryptAndDecryptWithComplicatedStrTest() throws Exception {
        testStr = "hello hello hello hello hello 4444 7777!!! Test Test";
        encryptStr = cryptographer.encrypt(testStr);
        String errorMessage = "Values of original and decrypt strings are different.";
        Assert.assertEquals(errorMessage, testStr, cryptographer.decrypt(encryptStr));
    }
}
```



Тестирование взаимодействия с хранилищем данных

- в эпицентре системы

```
public class LoginRequestStrategyTest extends Mockito{

    private RequestStrategy strategy = new LoginRequestStrategy(new DomXMLUserParser());

    private User existedUser = new User( login: "vi4477",  name: "Veronika Sanko", Role.GUEST,  password: "5588");

    XMLUserConverter converter = new XStreamXMLUserConverter();

    @Test
    public void executeLoginRequestWithExistedUserTest() throws Exception {
        Request request = mock(Request.class);

        String requestBodyStr = "vi4477 5588";
        when(request.getRequestBody()).thenReturn(requestBodyStr);

        Response response = strategy.executeRequest(request);
        verify(request).getRequestBody();

        User result = converter.convertXMLStringToUserObject(response.getResponseBody());
        String errorMessage = "Result and existed user objects are different";
        Assert.assertEquals(errorMessage, result, existedUser);
    }
}
```

LoginRequestStrategyTest (com.gv.archive.communication.strategies.implementations)
executeLoginRequestWithExistedUserTest

Преимущества подхода TDD

— ваш код полностью покрыт тестами;

— создавая тесты до написания кода класса, вы заранее задумаетесь об его использовании, что положительно скажется как на качестве внешнего интерфейса класса, так и на архитектуре проекта в целом;

— в общем случае, затрачивается меньше времени на качественную разработку ПО;

— хорошие тесты могут легко заменить документацию, т.к. наглядно демонстрируют использование тестируемого кода (live documentation);

Недостатки подхода TDD

- высокий порог вхождения;
- ошибочный или некорректный тест приводит к написанию такого же ошибочного кода;
- много времени тратится при использовании интеграционных и сквозных тестов;
- необходимо игнорировать слишком простые/сложные ситуации для тестирования;
- существуют ситуации в которых TDD использовать нецелесообразно:
  - невысокая цена ошибки
  - слишком дорого
  - нужно было сделать вчера

Выводы

Материалы: https://github.com/Strelts0v/tdd-conference