LUXOFT TRAINING

Разработка через тестирование Mock & Stub, EasyMock

Test Driven Development

Ivan Dyachenko <IDyachenko@luxoft.com>

Module 5

Содержание

LUXOFT Training Center

EasyMock

LUXOFT Training Center

- Одним из самых популярных инструментов, применяемых при interaction тестировании, является **EasyMock**
- Рассмотрим использование mock-объектов на примере простого клиент-серверного приложения **SimpsonViewer**, используемого для просмотра эпизодов Симпсонов

EasyMock

LUXOFT Training Center

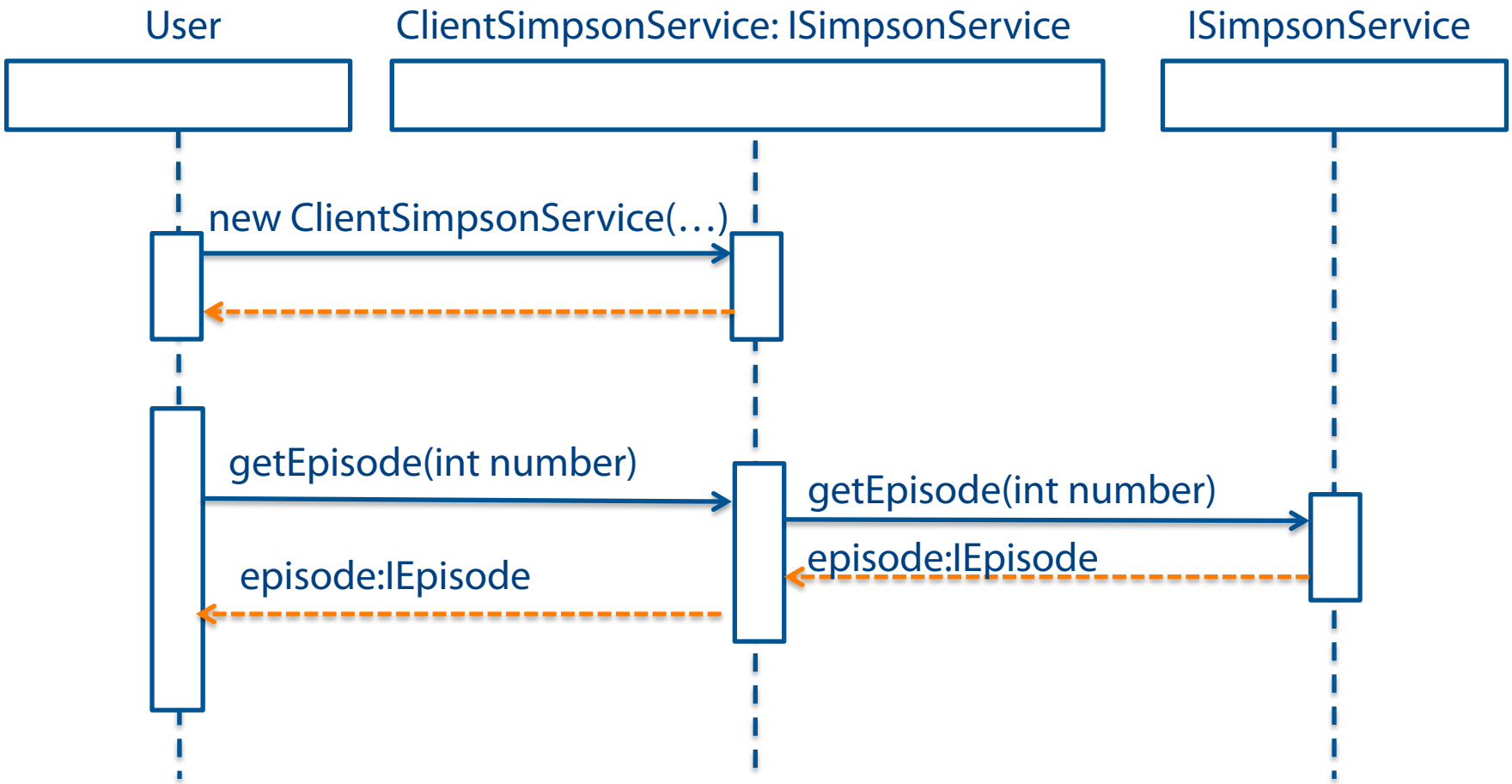
У нас есть web-сервис, возвращающий требуемый эпизод ISimpsonService

```
public interface ISimpsonService {
    IEpisode getEpisode(int number);
}

public interface IEpisode {
    int getNumber();
    String getTitle();
    InputStream getDataAsStream();
}
```

EasyMock

Клиентская часть используется для получения эпизодов с сервера и показывает их пользователю, являясь, по сути, клиентским прокси для web-сервиса



EasyMock

LUXOFT Training Center

```
public class ClientSimpsonService implements ISimpsonService {

    private ISimpsonService remoteSimpsonService;

    public ClientSimpsonService(ISimpsonService remoteSimpsonService) {
        this.remoteSimpsonService = remoteSimpsonService;
    }

    public IEpisode getEpisode(int episodeNumber) {
        return null;
    }
}
```

EasyMock

Напишем первый тест, который инициализирует наш сервис с null-параметром

```
@Test
public void testNullConstructor() {
    try {
        new ClientSimpsonService(null);
        fail("Ожидаемое исключение не получено");
    } catch (IllegalArgumentException e) {
        // expected
    }
}
```

EasyMock

В текущей реализации тест провалится, поэтому изменим наш сервис чтобы тест был успешным

```
public class ClientSimpsonService implements ISimpsonService {
    ...
    public ClientSimpsonService(ISimpsonService remoteSimpsonService){
        this.remoteSimpsonService = remoteSimpsonService;
        throw new IllegalArgumentException();
    }
}
```

EasyMock

LUXOFT Training Center

- Однако конструктор не должен постоянно «кидать» исключение, а только в том случае, когда в качестве параметра передается null
- Напишем тест, который не должен падать, если мы передадим в конструктор реализацию ISimpsonService
- Поскольку у нас нет реализации сервиса (кроме используемой в приложении), будем использовать mock-объект

EasyMock

LUXOFT Training Center

```
// добавляем методы EasyMock
import static org.easymock.EasyMock.*;
// объявляем объект-реализацию нужного интерфейса
private ISimpsonService remoteSimpsonServiceMock;

@BeforeClass
protected void setUp() throws Exception {
    remoteSimpsonServiceMock =
    EasyMock.createMock(ISimpsonService.class);

    // метод createMock() налету создает экземпляр класса,
    // реализующий указанный интерфейс}
```

EasyMock

LUXOFT Training Center

```
@Test
public void testValidConstructor() {
    // и теперь у нас есть валидный экземпляр,
    // который мы можем передать в конструктор

    new ClientSimpsonService(remoteSimpsonServiceMock);
}
```

EasyMock

В текущей реализации тест провалится, поэтому изменим наш сервис чтобы тест был успешным

```
ISimpsonService {
    ...
    public ClientSimpsonService(ISimpsonService remoteSimpsonService) {
        if (remoteSimpsonService == null)
            throw new IllegalArgumentException();
        this.remoteSimpsonService = remoteSimpsonService;
    }
}
```

EasyMock

LUXOFT Training Center

- Перейдём к тестированию метода getEpisode()
- Здесь нам понадобится заглушки как для ISimpsonService, так и для IEpisode

```
@BeforeClass
protected void setUp() throws Exception {
    remoteSimpsonServiceMock =
    EasyMock.createMock(ISimpsonService.class);
    episodeMock = EasyMock.createMock(IEpisode.class);
}
```

EasyMock

LUXOFT Training Center

```
@Test
public void testGetEpisode() throws Exception {
    EasyMock.expect(
        remoteSimpsonServiceMock.getEpisode(17)).andReturn(
            episodeMock);
}
```

- Методом expext(), мы задаем, какие методы объекта и с какими параметрами должны быть вызваны
- Методом andReturn() можно задать возвращаемый результат

EasyMock

LUXOFT Training Center

```
EasyMock.replay(remoteSimpsonServiceMock);
```

- Метод replay() завершает настройку («запись») заглушки и переводит ее в режим использования («воспроизведения»)
- В режиме записи, mock-объект еще не является заглушкой, а лишь «записывает», что он должен делать

EasyMock

LUXOFT Training Center

- После вызова replay(), он начинает работать как заглушка

```
EasyMock.verify(remoteSimpsonServiceMock);
assertEquals(episodeMock, result);
```

- Метод verify() как правило, завершает сценарий использования объекта и проверяет, действительно ли были сделаны требуемые вызовы с нужными параметрами

EasyMock

- Поскольку у нас еще нет реализации метода getEpisode(), тест провалится
- Дополним реализацию нужным методом

```
public class ClientSimpsonService implements ISimpsonService {
    ...
    public IEpisode getEpisode(int episodeNumber) {
        return remoteSimpsonService.getEpisode(
                    episodeNumber);
    }
}
```

EasyMock

LUXOFT Training Center

Т.о. схема использования mock-фреймворка выглядит следующим образом:

- Создаем mock-объект
- Задаем ожидания вызовов и возвращаемые значения
- Переводим mock в режим «воспроизведения»
- Вызываем методы тестируемого объекта
- Проверяем, что наши ожидания соответствуют реальному поведению объекта

EasyMock

LUXOFT Training Center

- Если метод должен выполниться несколько раз, то задается это следующим образом:

```
EasyMock.expect(
    remoteSimpsonServiceMock.getEpisode(17)).andReturn(
        episodeMock).times(3);
```

- Для указания количества вызовов так же могут использоваться setter'ы: **atLeastOnce**(), **anyTimes**(), **times**(from, to)

EasyMock

LUXOFT Training Center

- Стоит помнить, что после вызова EasyMock.verify() для mock-объекта, он считается «отработанным»
- Для того, чтобы снова использовать этот объект нужно вызвать EasyMock.reset() и заново установить поведение mock-объекта

EasyMock - Тестируем исключения:


```
@Test
public void testGetEpisodeException() throws Exception {
    expect(serverSimpsonServiceMock.getEpisode(666)).
        andThrow(new EpisodeNotFoundException());

    replay(serverSimpsonServiceMock);
    ISimpsonService clientSimpsonService = new
        ClientSimpsonService(serverSimpsonServiceMock);

    try {
        clientSimpsonService.getEpisode(666);
    } catch (EpisodeNotFoundException e) {
        fail("Expected EpisodeNotFoundException");
    }
    verify(serverSimpsonServiceMock);
}
```

EasyMock

LUXOFT Training Center

В зависимости от целей тестирования, EasyMock предлагает несколько разновидностей mock-объектов

- Nice mock
  - В отличие от дефолтного mock'а, не вызывает AssertionError на незапланированные вызовы, а возвращает значение по умолчанию (0, null или false)
  - Создается вызовом EasyMock.createNiceMock()

EasyMock

LUXOFT Training Center

В зависимости от целей тестирования, EasyMock предлагает несколько разновидностей mock-объектов

- Strict mock
  - В отличие от дефолтного mock'а, проверяет не только вызовы методов, но и их последовательность
  - Создается вызовом EasyMock.createStrictMock()
  - Переключаться между обычной и strict заглушкой можно «налету», вызовом EasyMock.checkOrder(Object mock, boolean enableOrderCheck)

Вопросы ?

Разработка через тестирование

IDyachenko@luxoft.com

git clone git://github.com/ivan-dyachenko/Trainings.git

https://github.com/ivan-dyachenko/Trainings