LUXOFT TRAINING

Разработка через тестирование Integration Tests

Test Driven Development

Ivan Dyachenko <IDyachenko@luxoft.com>

Module 6

Для кого этот тренинг?

LUXOFT Training Center

1. **Beginner**
   Хорошая точка входа
2. **Intermediate**
   Поможет лучше всё структурировать в голове и объяснять коллегам
3. **Advanced**
   Можно использовать для обучения и проверки других

Содержание

LUXOFT Training Center

Интеграционное тестирование

Интеграционное тестирование — одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе.

Тестирование архитектуры системы

Интеграционное тестирование называют еще тестированием архитектуры системы.

Результаты выполнения интеграционных тестов – один из основных источников информации для процесса улучшения и уточнения архитектуры системы, межмодульных и межкомпонентных интерфейсов. Т.е. с интеграционные тесты проверяют корректность взаимодействия компонент системы.

Итеративный процесс

Интеграционное тестирование, как правило, представляет собой итеративный процесс, при котором проверяется функциональность все более и более увеличивающейся в размерах совокупности модулей.

Пример

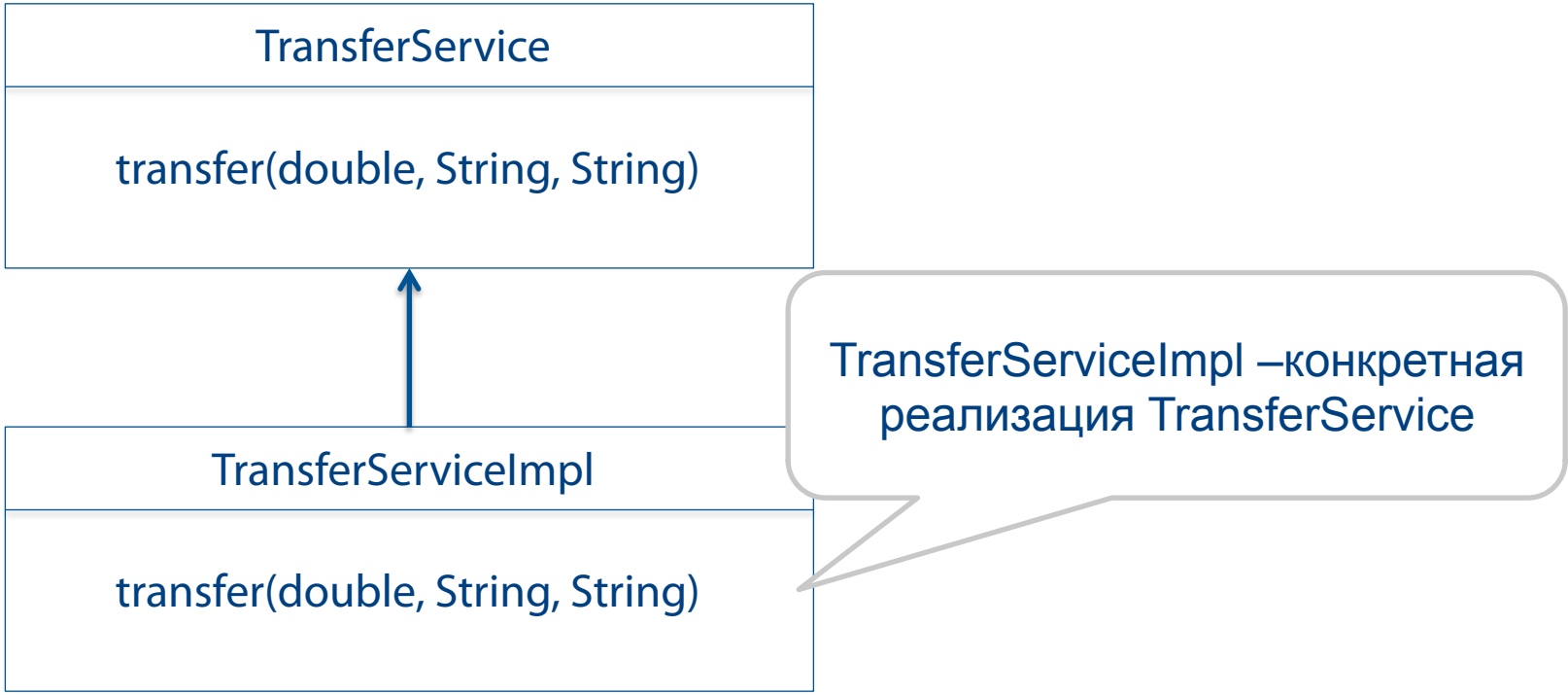
LUXOFT Training Center

TransferService
transfer(double, String, String)

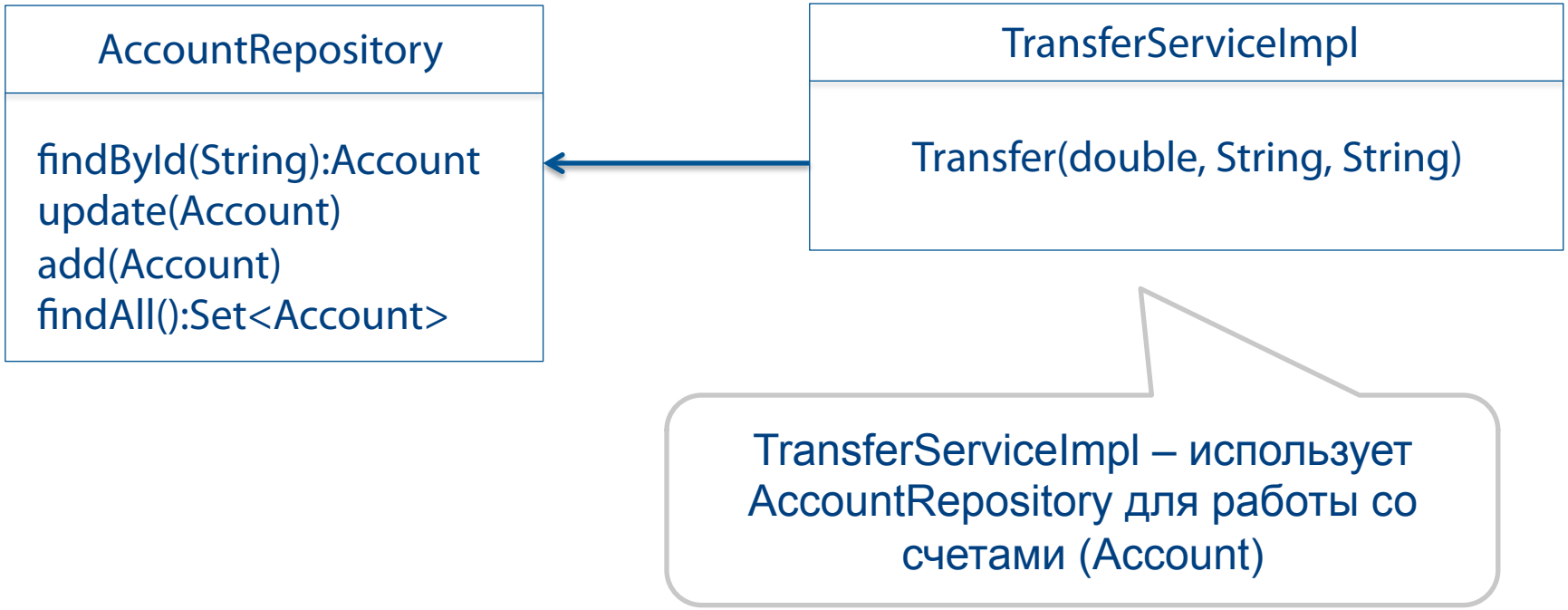
TransferService – сервис для перевода средств с одного счета на другой.

springsource community

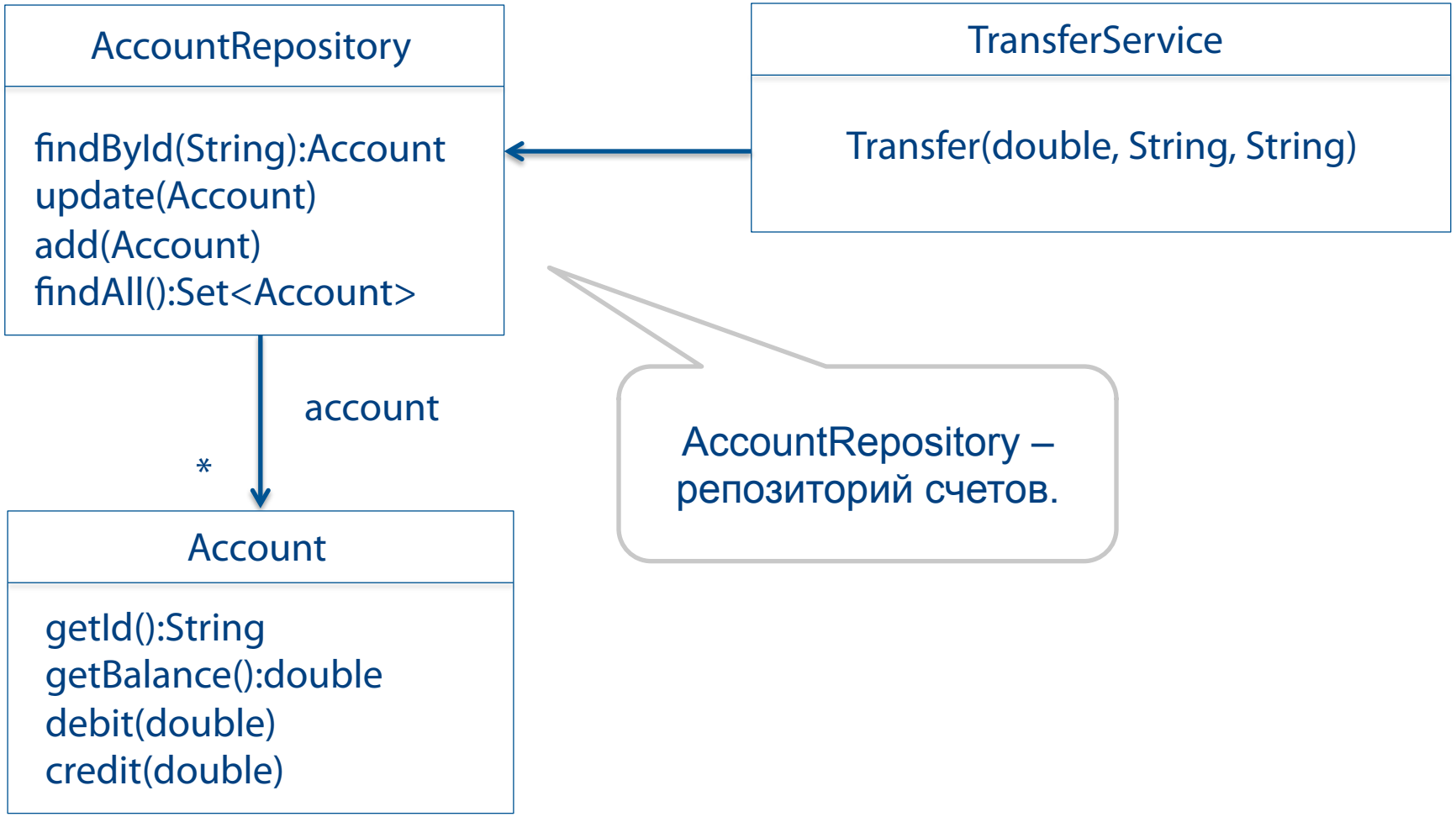
Пример



Пример



Пример



Необходимо

Протестировать работу сервиса в интеграции с AccountRepository и Account.

Как тестировать?

Надо протестировать работу сервиса на простом переводе средств с одного аккаунта на другой.

1) Дано:

- Account A123 - $1000
- Account C456 - $0
- AccountRepository - [A123, C456]

3) Проверить что:

- Account A123 - $900
- Account C456 - $100

2) Необходимо:

Перевести с Account A123 - $1000 на Account C456 - $0 сумму в $100

```
transferService.transfer(100.00, "A123", "C456");
```

JUnit

Чаще всего для написания интеграционных тестов используются те же библиотеки для тестирования, что и для модульных тестов.

Пишем тест

LUXOFT Training Center

```
@Test
public void transfer100Dollars() {
    // create instances
    AccountRepository accountRepository = new InMemoryAccountRepository();
    TransferService transferService = new TransferServiceImpl(accountRepository);

    // create accounts to test against
    accountRepository.add(new Account("A123", 1000.00));
    accountRepository.add(new Account("C456", 0.00));

    // check account balances before transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(1000.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(0.00));

    // perform transfer
    transferService.transfer(100.00, "A123", "C456");

    // check account balances after transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(900.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(100.00));
}
```

Пишем тест

Создаем классы и устанавливаем их зависимость

```
@Test
public void transfer100Dollars() {
    // create instances
    AccountRepository accountRepository = new InMemoryAccountRepository();
    TransferService transferService = new TransferServiceImpl(accountRepository);

    // create accounts to test against
    accountRepository.add(new Account("A123", 1000.00));
    accountRepository.add(new Account("C456", 0.00));

    // check account balances before transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(1000.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(0.00));

    // perform transfer
    transferService.transfer(100.00, "A123", "C456");

    // check account balances after transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(900.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(100.00));
}
```

Пишем тест

LUXOFT Training Center

```
@Test
public void transfer100Dollars() {
    // create instances
    AccountRepository accountRepository = n
    TransferService transferService = new T                    );

    // create accounts to test against
    accountRepository.add(new Account("A123", 1000.00));
    accountRepository.add(new Account("C456", 0.00));

    // check account balances before transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(1000.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(0.00));

    // perform transfer
    transferService.transfer(100.00, "A123", "C456");

    // check account balances after transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(900.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(100.00));
}
```

Создаем два счета на $1000 и $0

Пишем тест

LUXOFT Training Center

```
@Test
public void transfer100Dollars() {
    // create instances
    AccountRepository accountRepository = new InMemoryAccountRepository();
    TransferService transferService = new TransferServiceImpl(accountRepository);

    // create accounts to test
    accountRepository.add(new Account("A123", 1000.00));
    accountRepository.add(new Account("C456", 0.00));

    // check account balances before transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(1000.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(0.00));

    // perform transfer
    transferService.transfer(100.00, "A123", "C456");

    // check account balances after transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(900.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(100.00));
}
```

Проверяем состояние счетов до перевода

Пишем тест

LUXOFT Training Center

```
@Test
public void transfer100Dollars() {
    // create instances
    AccountRepository accountRepository = new InMemoryAccountRepository();
    TransferService transferService = new TransferServiceImpl(accountRepository);

    // create accounts to test against
    accountRepository.add(new Account("A123", 1000.00));
    accountRepository.add(new Account("C456", 0.00));

    // check account balances b
    assertThat(accountReposito                                    00));
    assertThat(accountReposito                                    );

    // perform transfer
    transferService.transfer(100.00, "A123", "C456");

    // check account balances after transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(900.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(100.00));
}
```

Переводим $100 с одного счета на другой

Пишем тест

LUXOFT Training Center

```
@Test
public void transfer100Dollars() {
    // create instances
    AccountRepository accountRepository = new InMemoryAccountRepository();
    TransferService transferService = new TransferServiceImpl(accountRepository);

    // create accounts to test against
    accountRepository.add(new Account("A123", 1000.00));
    accountRepository.add(new Account("C456", 0.00));

    // check account balances before transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(1000.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(0.00));

    // perform transfer
    transferService.transfer(100.00, "A123", "C456");

    // check account balances after transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(900.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(100.00));
}
```

Проверяем состояние счетов после перевода

AnnotationConfigApplicationContext

Перепишем тест с созданием Spring Context используя AppConfig и AnnotationConfigApplicationContext

AppConfig

LUXOFT Training Center

```
@Configuration
public class AppConfig {

    public @Bean TransferService transferService() {
        return new TransferServiceImpl(accountRepository());
    }

    public @Bean AccountRepository accountRepository() {
        return new InMemoryAccountRepository();
    }

}
```

Пишем тест

Мы использовали Spring для создания классов и их зависимостей (DI)

```
@Test
public void transfer100Dollars() {
    // create the spring container using the AppConfig @Configuration class
    ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);

    // retrieve the beans we'll use during tesddting
    AccountRepository accountRepository = ctx.getBean(AccountRepository.class);
    TransferService transferService = ctx.getBean(TransferService.class);

    // create accounts to test against
    accountRepository.add(new Account("A123", 1000.00));
    accountRepository.add(new Account("C456", 0.00));

    // check account balances before transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(1000.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(0.00));

    // perform transfer
    transferService.transfer(100.00, "A123", "C456");

    // check account balances after transfer
    assertThat(accountRepository.findById("A123").getBalance(), equalTo(900.00));
    assertThat(accountRepository.findById("C456").getBalance(), equalTo(100.00));
}
```

Black Box Tesitng

Тестирование чёрного ящика или поведенческое тестирование — стратегия (метод) тестирования функционального поведения объекта (программы, системы) с точки зрения внешнего мира, при котором не используется знание о внутреннем устройстве тестируемого объекта.

Spring Integration Testing

LUXOFT Training Center

Goals of integration testing:

Spring Testing

Spring's integration testing support has the following goals:

- Spring IoC container caching between test execution.
- Dependency Injection of test fixture instances.
- Transaction management appropriate to integration testing.
- Spring-specific support classes that are useful in writing integration tests

Специализированные классы

LUXOFT Training Center

JUnit4 support:

- AbstractJUnit4SpringContextTests
- AbstractTransactionalJUnit4SpringContextTests
- SpringJUnit4ClassRunner

Spring Framework API 2.5

Spring Testing

Что бы воспользоваться этими классами, нам надо добавить артефакт spring-test в pom.xml

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${org.springframework.version}</version>
</dependency>
```

Spring Framework API 2.5

Пишем тест

LUXOFT Training Center

```
@ContextConfiguration(locations = {"classpath:/applicationContext.xml"})
public class TransferServiceContextTest extends AbstractJUnit4SpringContextTests {

    @Autowired
    private ApplicationContext context;

    @Autowired
    private AccountRepository accountRepository;

    @Autowired
    private TransferService transferService;

    @Test
    public void transfer100Dollars() {
        // create accounts to test against
        accountRepository.add(new Account("A123", 1000.00));
        accountRepository.add(new Account("C456", 0.00));

        // check account balances before transfer
        assertThat(accountRepository.findById("A123").getBalance(), equalTo(1000.00));
        assertThat(accountRepository.findById("C456").getBalance(), equalTo(0.00));

        // perform transfer
        transferService.transfer(100.00, "A123", "C456");

        // check account balances after transfer
        assertThat(accountRepository.findById("A123").getBalance(), equalTo(900.00));
        assertThat(accountRepository.findById("C456").getBalance(), equalTo(100.00));
    }

}
```

Типичный шаблон интеграционного теста

Перепишем тест с использованием **SpringJUnit4ClassRunner** и декомпозицией метода **transfer100Dollars** на отдельные Use Cases.

А также вынесем логику по созданию и добавлению счетов в **AccountRepository** в отдельный метод **setUp**

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath:/applicationContext.xml"})
public class TransferServiceClassRunnerTest {

    @Autowired
    private AccountRepository accountRepository;

    @Autowired
    private TransferService transferService;

    @Before
    public void setUp() {
        accountRepository.add(new Account("A123", 1000.00));
        accountRepository.add(new Account("C456", 0.00));
    }

    @After
    public void tearDown() {
        accountRepository.clear();
    }

    @Test
    public void shouldHaveCorrectInitialState() {
        assertThat(accountRepository.findById("A123").getBalance(), equalTo(1000.00));
        assertThat(accountRepository.findById("C456").getBalance(), equalTo(0.00));
    }

    @Test
    public void shouldTransferMoneyBetweenAccounts() {
        // when
        transferService.transfer(100.00, "A123", "C456");
        // then
        assertThat(accountRepository.findById("A123").getBalance(), equalTo(900.00));
        assertThat(accountRepository.findById("C456").getBalance(), equalTo(100.00));
    }
}
```

LUXOFT Training Center

Инициализация и отчистка данных для каждого тестового метода

Test-driving DB layer

Взаимодействие с источниками данных

Взаимодействие с источниками данных

Интеграционные тесты, которые изменяют данные в базе данных, должны откатывать состоянии базы данных к тому, которое было до запуска теста, даже если тест не прошёл.

Для этого часто применяются следующие техники:

- Метод TearDown, присутствующий в большинстве библиотек для тестирования.
- Try – catch - finally структуры обработки исключений, там где они доступны.
- Транзакции баз данных.
- Создание снимка (англ. snapshot) базы данных перед запуском тестов и откат к нему после окончания тестирования.
- Сброс базы данных в чистое состояние перед тестом, а не после них. Это может быть удобно, если интересно посмотреть состояние базы данных, оставшееся после не прошедшего теста.

Цель

Протестировать поведение класса в spring-приложении взаимодействующим с базой данных, дополнительно необходимо вручную управлять транзакциями.

Термины

- **Application context config** — конфигурационный файл в XML формате для описания структуры spring приложения.
- **DAO** — объект доступа к данным или Data Acess Object. Основное предназначение этого шаблона проектирования: связать вместе БД и наше приложение.
- **Транзакция** — группа последовательных операций, которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще и тогда она не должна произвести никакого эффекта.

Технологии в приложении

- **Application context config** — конфигурационный файл в XML формате для описания структуры spring приложения.
- **DAO** — объект доступа к данным или Data Acess Object. Основное предназначение этого шаблона проектирования: связать вместе БД и наше приложение.
- **Транзакция** — группа последовательных операций, которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще и тогда она не должна произвести никакого эффекта.

Test Asynchronous Processes

http://stackoverflow.com/questions/631598/how-to-use-junit-to-test-asynchronous-processes

Вопросы ?

Разработка через тестирование

IDyachenko@luxoft.com

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

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