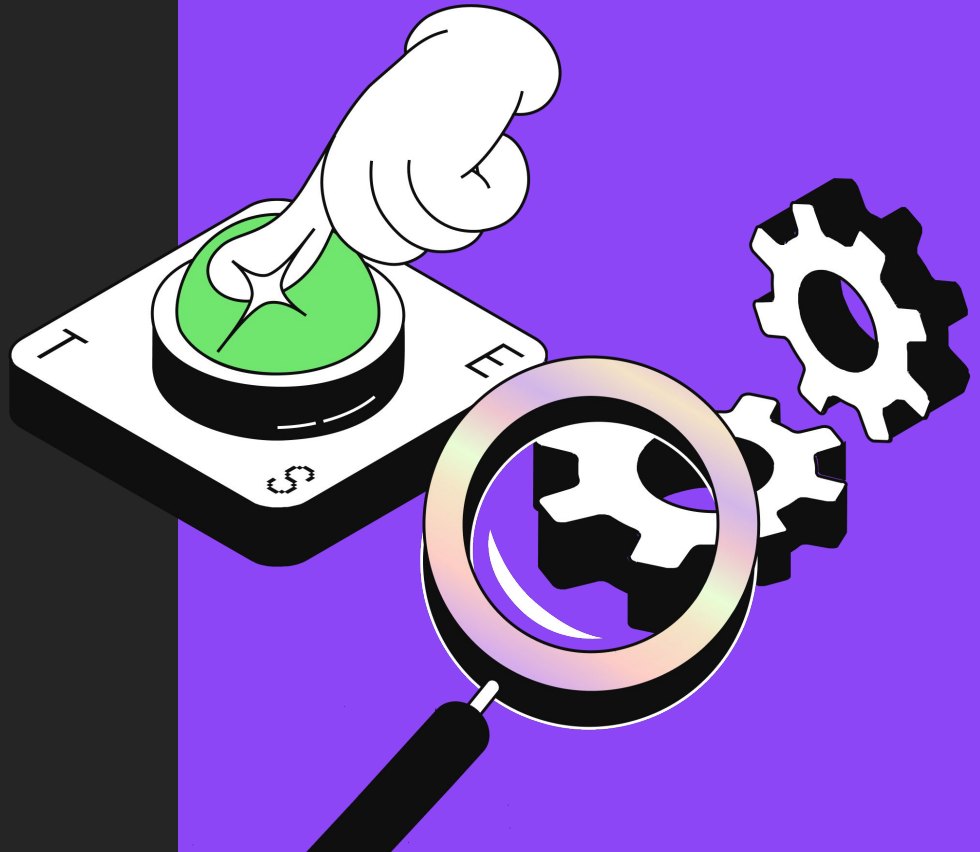
GeekBrains

Введение в юнит-тестирование

Урок 4

Зависимости в тестах



План курса

1. Цели и смысл тестирования
2. Знакомство с тестовыми фреймворками
3. Качество тестов
4. Зависимости в тестах
6. Другие виды тестирования
7. Тестирование в других языках программирования

На прошлой лекции

На прошлой лекции

- 📌 Качество тестов
- 📌 Тестирование по принципу черного и белого ящика
- 📌 Метрики тестов
- 📌 Инструменты для измерения покрытия тестами
- 📌 Тестирование через разработку
- 📌 Тестирование через поведение

Содержание урока

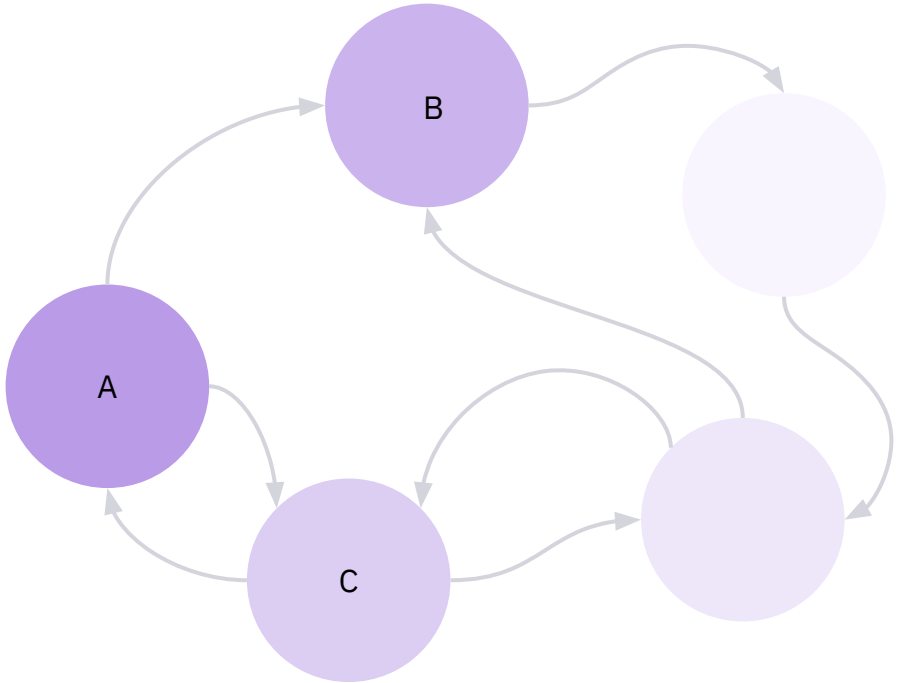
Что будет на уроке сегодня

- 📌 Понятие зависимости
- 📌 Виды зависимостей
- 📌 Тестовые заглушки (Test Double)
- 📌 Подходы к изоляции кода в школах тестирования
- 📌 Типы тестовых заглушек
- 📌 Фреймворк Mockito
- 📌 Создание и настройка моков с Mockito
- 📌 Другие mocking-фреймворки

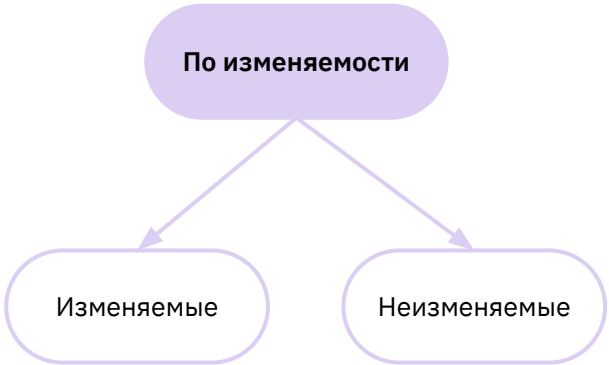
Зависимости

Преимущества качественных тестов

```
public class Service {

    private Database database;

    public Service(Database database) {
        this.database = database;
    }
}
```



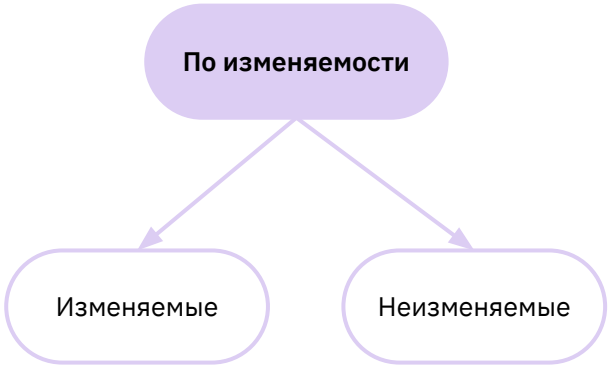
Типы зависимостей



Переменные
или база данных

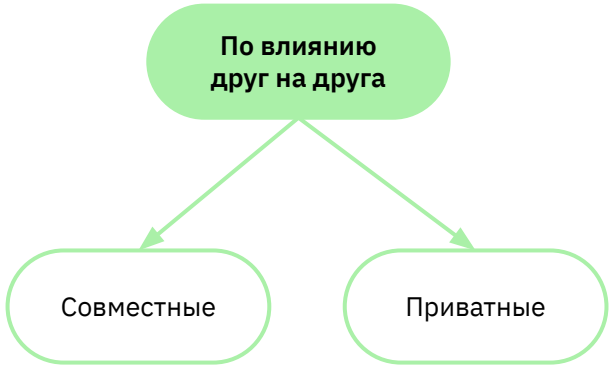
Константа

Типы зависимостей



Переменные
или база данных

Константа



Статическое
изменяемое поле
или база данных

База данных

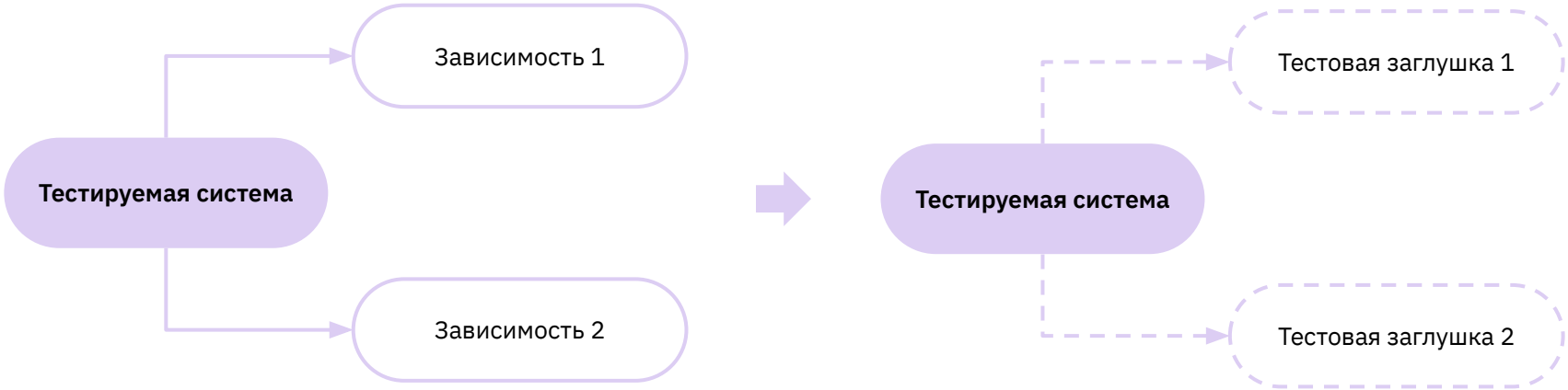
Test Double

Test Double (Тестовая заглушка)

Это любой объект, который выглядит и ведет себя как его рабочий аналог, но в действительности представляет собой упрощенную версию, более удобную для тестирования.

- ✔ Термин ввел Gerard Meszaros в книге «xUnit Test Patterns: Refactoring Test Code»
- ✔ Английский вариант термина происходит от stunt double — дублеров актеров на съемках

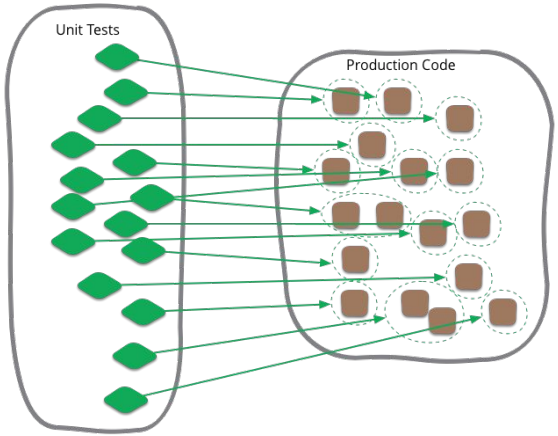
Test Double

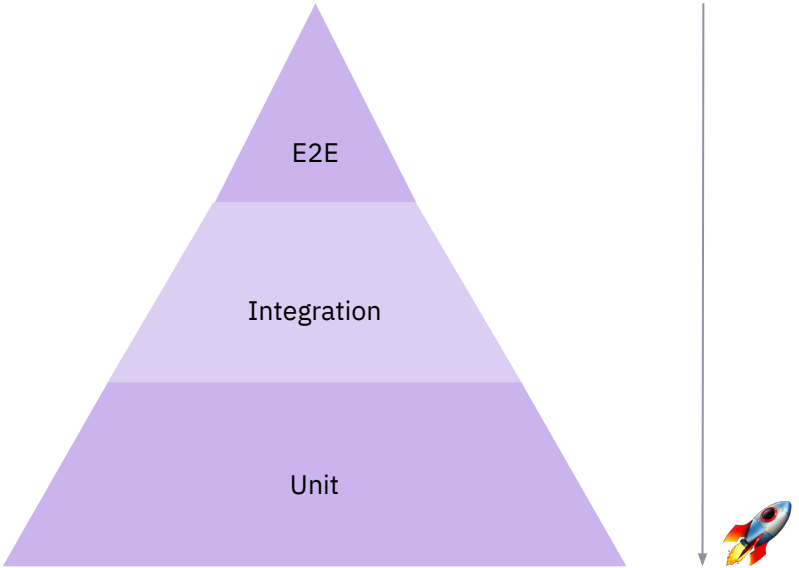

Зависимость 1
Тестируемая система
Зависимость 2
Тестируемая система
Тестовая заглушка 1
Тестовая заглушка 2

Что заменять тестовыми заглушками

Юнит-тест — это автоматизированный тест, который:

- ✔ проверяет правильность работы юнит-единицы кода
- ✔ проводит тестирование быстро
- ✔ изолирован от другого кода





Что заменять тестовыми заглушками

Юнит-тест — это автоматизированный тест, который:

- ✔ проверяет правильность работы юнит-единицы кода
- ✔ проводит тестирование быстро
- ✔ изолирован от другого кода

	Изоляция	Что считается юнитом (единицей кода)	Что именно изолируется
Лондонская школа	Юнитов	Класс	Любые **изменяемые** зависимости
Классическая школа	Юнит-тестов	Класс или набор классов	**Совместные** зависимости

Что заменять тестовыми заглушками

Лондонская школа

Большое количество тестовых заглушек

- ✘ При рефакторинге может понадобится заново настраивать все тестовые заглушки;
- ✔ Более детализированные тесты;
- ✔ Легче тестировать сложный (со многими зависимостями) код;
- ✔ Легко найти ошибку.

Классическая школа

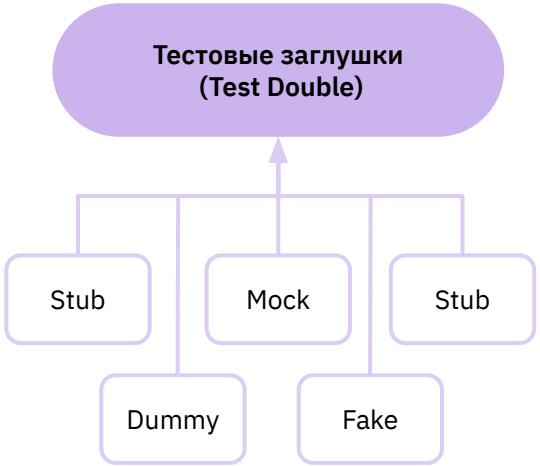
В тестах чаще используются реальные объекты

- ✘ Возможные проблемы с поиском ошибок;
- ✔ Менее хрупкие тесты.

Типы Test Doubles

Типы тестовых заглушек

- ✔ **Dummy (Фиктивный объект или Объект-заглушка)** просто передается тестируемой системе в качестве аргумента.



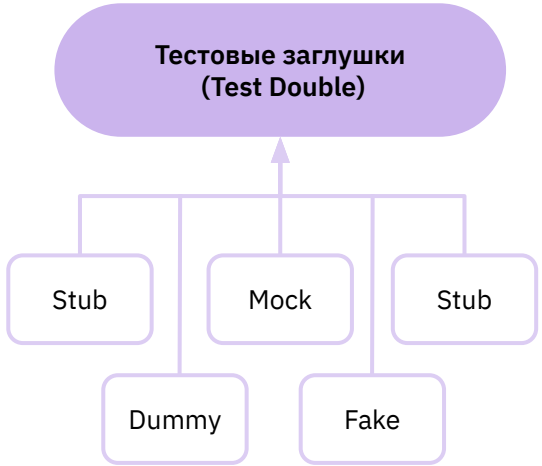
Dummy

```
class PaymentServiceShould {

    @Test
    void create_payment_request() {
        // Создается dummy-обьект без реализации
        LoggerDummy loggerDummy =new LoggerDummy();
                . . .
        // loggerDummy передается в качестве аргумента
        PaymentService paymentService =new PaymentService(loggerDummy);

        assertEquals(true, paymentService.createPaymentRequest());
    }
}
```

```
public class LoggerDummy implements Logger
{
          //      Реализация      отсутствует
}
```

Типы тестовых заглушек

- ✔ **Dummy (Фиктивный объект или Объект-заглушка)** просто передается тестируемой системе в качестве аргумента.
- ✔ **Stub (Заглушки)** представляет собой объект, заменяющий реальный компонент, от которого зависит тестируемая система, для которого задается готовый ответы на вызов



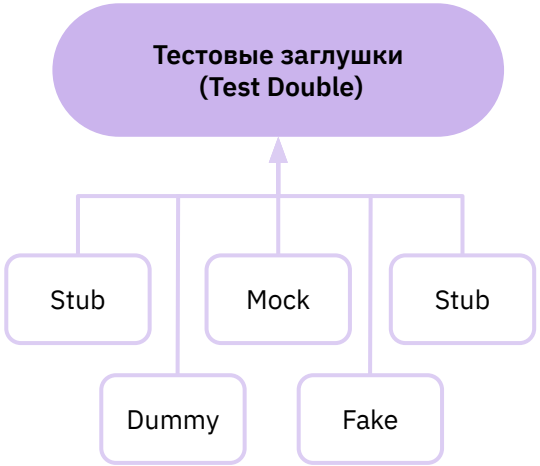
Stub

```
@Test
void create_payment_request() {
    . . .
        // 10 - жестко заданное значение
        OperatorRate operatorRate = new OperatorRateStub(10);
    PaymentService paymentService = new PaymentService(loggerDummy,
operatorRate);
                                                . . .
        assertEquals(true, paymentService.createPaymentRequest());
}
```

```
public class OperatorRateStub implements OperatorRate {
   private int rate;
   // Задается готовый ответ

   public OperatorRateStub(int rate){
       this.rate = rate;
   }
}
```

Типы тестовых заглушек

- ✔ **Dummy (Фиктивный объект или Объект-заглушка)** просто передается тестируемой системе в качестве аргумента.
- ✔ **Stub (Заглушки)** представляет собой объект, заменяющий реальный компонент, от которого зависит тестируемая система, для которого задается готовый ответы на вызов
- ✔ **Mock (Имитация)** заменяет реальный компонент, от которого зависит тестируемая система. Позволяет тесту проверять вывод



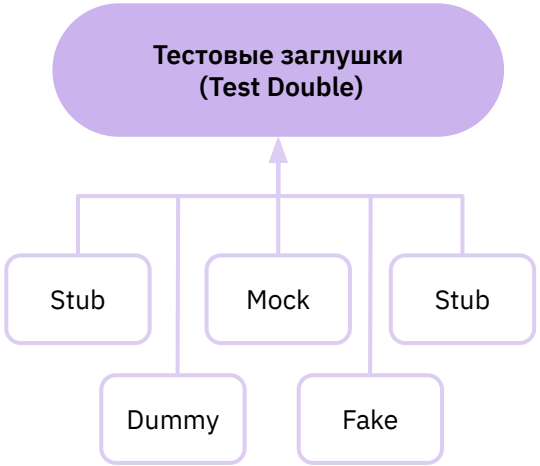
Mock

```
public class PaymentServiceMock implements PaymentEmailSender {
    . . .
    public void expect(PaymentRequest paymentRequest) {
        expectedPaymentRequest.add(paymentRequest);
    }

    public void verify() {
        assertEquals(paymentRequestSent, expectedPaymentRequest);
    }
}
```

```
@Test
void send_email_to_the_administration_if_sale_is_over_1000 () {
    EmailSenderMock emailSender = new EmailSenderMock();
    . . .
PaymentService paymentService = new PaymentService(loggerDummy, operatorRate, emailSender);
    PaymentRequest paymentRequest = new PaymentRequest(1000);
    paymentService.createPaymentRequest();

    // проверяются выходные взаимодействия:
    emailSender.expect(paymentRequest);
    emailSender.verify();
}
```

Типы тестовых заглушек

- ✔ **Dummy (Фиктивный объект или Объект-заглушка)** просто передается тестируемой системе в качестве аргумента.
- ✔ **Stub (Заглушки)** представляет собой объект, заменяющий реальный компонент, от которого зависит тестируемая система, для которого задается готовый ответы на вызов
- ✔ **Mock (Имитация)** заменяет реальный компонент, от которого зависит тестируемая система. Позволяет тесту проверять вывод
- ✔ **Spy (Шпионы)** это заглушки, которые также записывают некоторую информацию, основанную на том, как они были вызваны



Spy

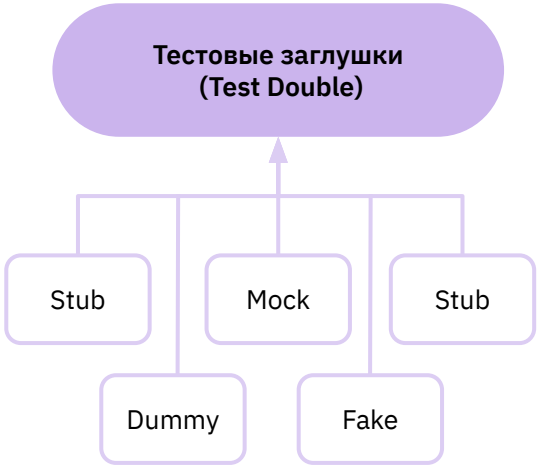
```
public class PaymentEmailSpy implements PaymentEmailSender {

    private List<PaymentRequest> paymentRequests =new ArrayList<>();

    @Override
    public void send(PaymentRequest paymentRequest) {
        paymentRequests.add(paymentRequest);
    }

    public int timesCalled() {
        return paymentRequests.size();
    }

    public boolean calledWith(PaymentRequest paymentRequest) {
        return paymentRequests.contains(paymentRequest);
    }
}
```

Spy

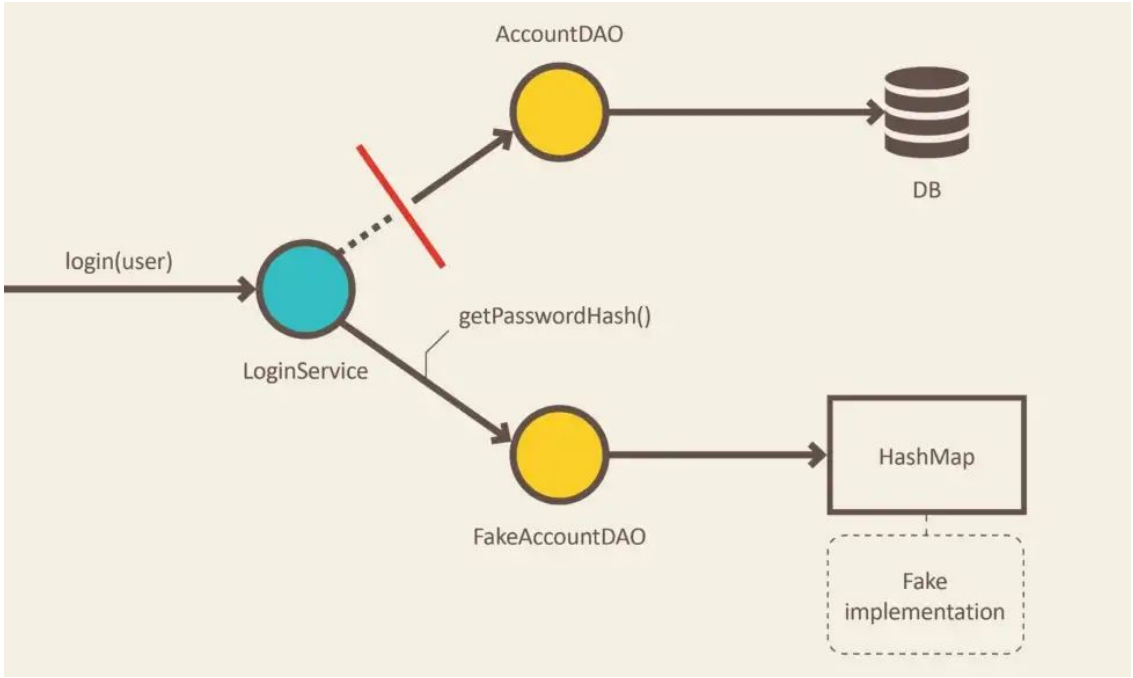
```
class                        PaymentServiceShould                        {

            . . .

                                                                    @Test
    void not_send_email_for_sales_under_1000() {
                                                                . . .
                      EmailSenderSpy  emailSpy  = new  EmailSenderSpy();
    PaymentService spiedPaymentService =new PaymentService(loggerDummy,
operatorRate,                                                   emailSpy);

                             spiedPaymentService.createPaymentRequest();

                             assertEquals(0,    emailSpy.timesCalled());
                                                                    }
}
```

Типы тестовых заглушек

- ✔ **Dummy (Фиктивный объект или Объект-заглушка)** просто передается тестируемой системе в качестве аргумента.
- ✔ **Stub (Заглушки)** представляет собой объект, заменяющий реальный компонент, от которого зависит тестируемая система, для которого задается готовый ответы на вызов
- ✔ **Mock (Имитация)** заменяет реальный компонент, от которого зависит тестируемая система. Позволяет тесту проверять вывод
- ✔ **Spy (Шпионы)** это заглушки, которые также записывают некоторую информацию, основанную на том, как они были вызваны
- ✔ **Fake (Подделка)** заменяет функциональность вызываемого компонента альтернативной реализацией



Fake

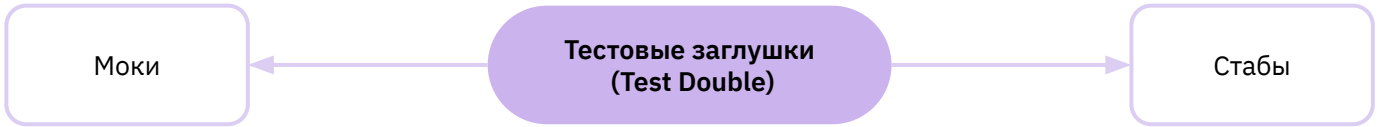

AccountDAO
DB
login(user)
getPasswordHash()
LoginService
HashMap
FakeAccountDAO
Fake
implementation

Fake

```
public class FakeAccountRepository implements
AccountRepository {

   Map<User, Account> accounts = new HashMap<>();

       public FakeAccountRepository() {
              this.accounts.put(new User("john@mail.com"),
new UserAccount());
              this.accounts.put(new User("boby@mail.com"),
new AdminAccount());
       }

       String getPasswordHash(User user) {
              return accounts.get(user).getPasswordHash();
       }
}
```

Типы тестовых заглушек

Все разновидности тестовых заглушек можно разделить на два типа: моки и стабы



Шпионы

Заглушки, фейки

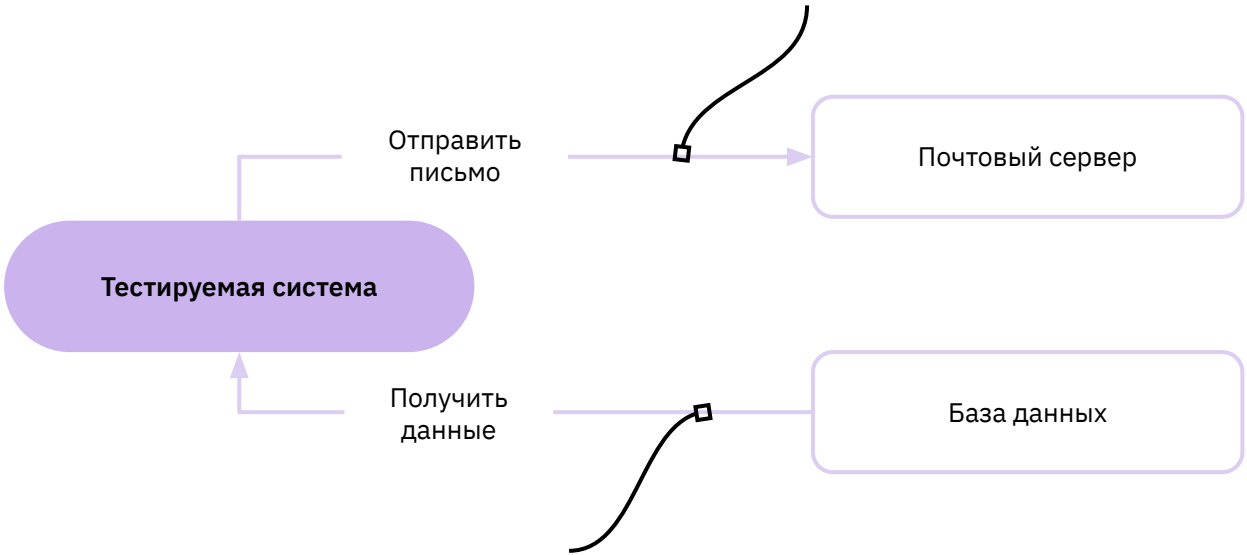
Моки помогают эмулировать и проверять **выходные** взаимодействия — то есть вызовы, совершаемые тестируемой системой к ее зависимостям для изменения их состояния.

Стабы помогают эмулировать **входные** взаимодействия — то есть вызовы, совершаемые тестируемой системой к ее зависимостям для получения входных данных.

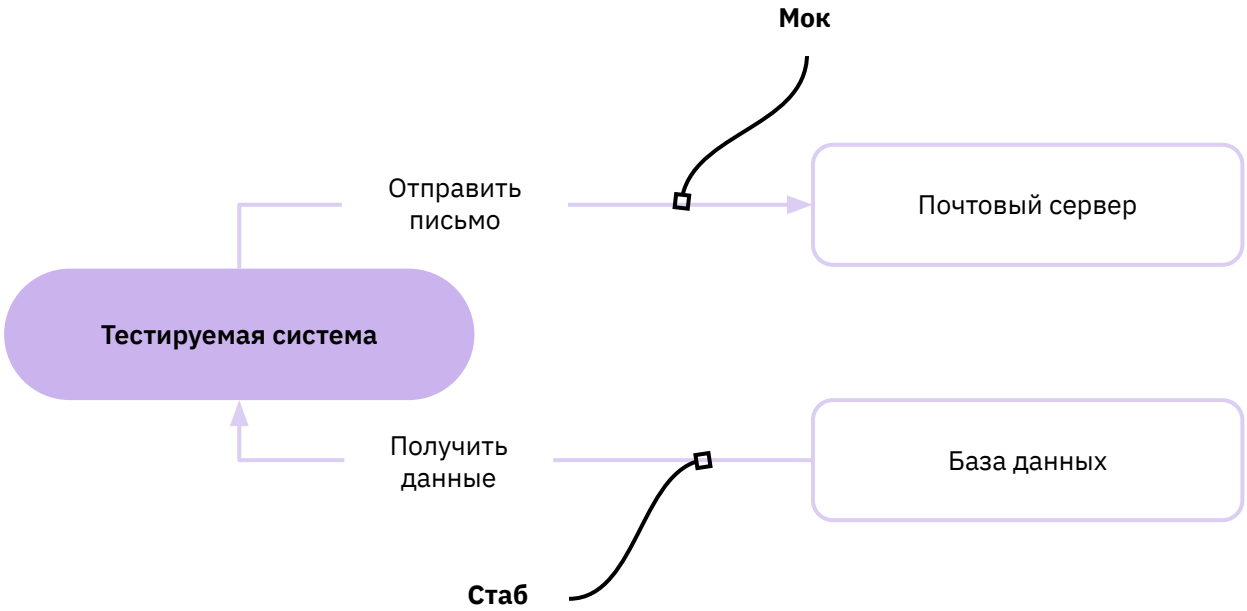
Вопрос

Как вы думаете, какие типы заглушек можно применить в системе?

Типы тестовых заглушек



Типы тестовых заглушек



Mocking framework

Mockito

Mockito — популярный фреймворк, который может быть использован с JUnit. Он позволяет создавать и настраивать mock-объекты

Входит в топ-10 библиотек Java по всем библиотекам, а не только по инструментам тестирования.

Mockito Core » 4.9.0

Mockito mock objects library core API and implementation

License	MIT
Categories	Mocking
Tags	mock mockito mocking testing
HomePage	https://github.com/mockito/mockito
Date	Nov 14, 2022
Files	pom (2 KB) jar (668 KB) View All
Repositories	Central
Ranking	**#5 in MvnRepository (See Top Artifacts)** **#1 in Mocking**
Used By	**28,319 artifacts**

Установка Mockito

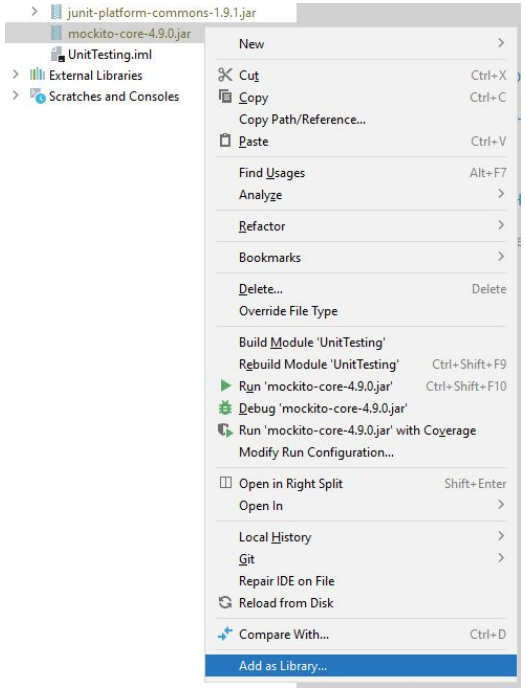
Compile Dependencies (2)

Category/License	Group / Artifact	Version	Updates
Bytecode Apache 2.0	net.bytebuddy » byte-buddy	1.12.16	1.12.19
Bytecode Apache 2.0	net.bytebuddy » byte-buddy-agent	1.12.16	1.12.19

Runtime Dependencies (1)

Category/License	Group / Artifact	Version	Updates
Reflection Apache 2.0	org.objenesis » objenesis	3.3	✓

Установка Mockito



Установка Mockito

```
import java.util.List;

import static org.mockito.Mockito.*;

public class Main {
   public static void main(String[] args) {
       // Создание mock
       List mockedList = mock(List.class);

       // Использование
       mockedList.add("one");
       System.out.println(mockedList.get(0)); // null
       mockedList.clear();
   }
}
```

Использование Mockito

Mockito. Создание mock-объектов

- ✔ Mockito позволяет создавать mock-объекты разными методами:
  с помощью статического метода mock();

```
List mockedList = mock(List.class);
```

- ✔ с использованием аннотации @Mock.

```
@Mock
private Calculator calculator;
```

Mockito. Задание условий вызова

✔ Вы можете использовать конструкцию when(...).thenReturn(...)
для указания условия и возвращаемого значения для этого условия

```
LinkedList mockedLinkedList =  mock(LinkedList.class);

when(mockedLinkedList.get( 0)).thenReturn( "nullValue");

// Выведет "nullValue"
System.out.println(mockedLinkedList.get( 0));
```

Mockito. Задание условий вызова

✔ Возможно настраивать выбрасываемое исключение:

```
LinkedList mockedLinkedList = mock(LinkedList.class);

when(mockedLinkedList.get(1)).thenThrow(new
RuntimeException());

// Вернет исключение runtime exception
System.out.println(mockedLinkedList.get(1));
```

Mockito. Слежение за вызовами методов

✔ Mockito отслеживает все вызовы методов и их параметров для mock-объекта. Вы можете использовать метод verify(), чтобы убедиться, что метод был вызван с определенными параметрам

```
LinkedList mockedLinkedList = mock(LinkedList.class);

when(mockedLinkedList.get(0)).thenReturn("nullValue");
when(mockedLinkedList.get(1)).thenThrow(new RuntimeException());
// Выведет "nullValue"
System.out.println(mockedLinkedList.get(0));

// Вернет исключение runtime exception
System.out.println(mockedLinkedList.get(1));

// Если mockedLinkedList.get(0) не будет вызван до этой проверки, то
тест провалится
verify(mockedLinkedList).get(0);
```

Mockito. Остальные возможности

- ✔ Вызов реальных методов

```
when(mock.someMethod()).thenCallRealMethod()
```

- ✔ Проверка с таймаутом

```
verify(mock, timeout(100)).someMethod();
```

- ✔ Проверка точного количества вызовов

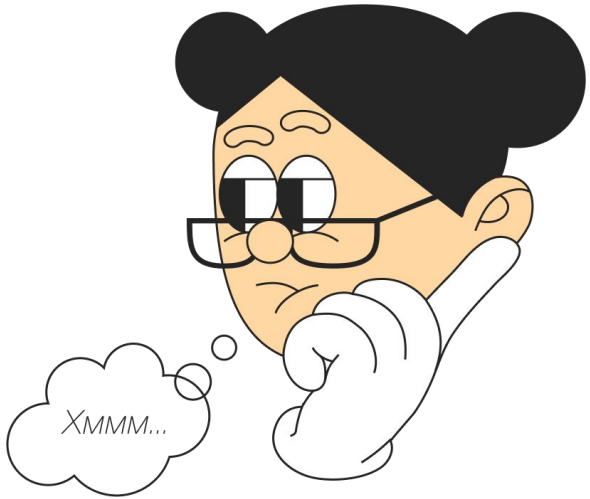
```
verify(mockedList, times(2)).add("twice");
verify(mockedList, never()).add("never happened");
```

Mockito. BDD

```
import static org.mockito.BDDMockito.*;

 Seller seller = mock(Seller.class);
 Shop shop = new Shop(seller);

 public void shouldBuyBread() throws Exception {
   //given
   given(seller.askForBread()).willReturn(new Bread());

   //when
   Goods goods = shop.buyBread();

   //then
   assertThat(goods, containBread());
 }
```

Mockito. Ограничения

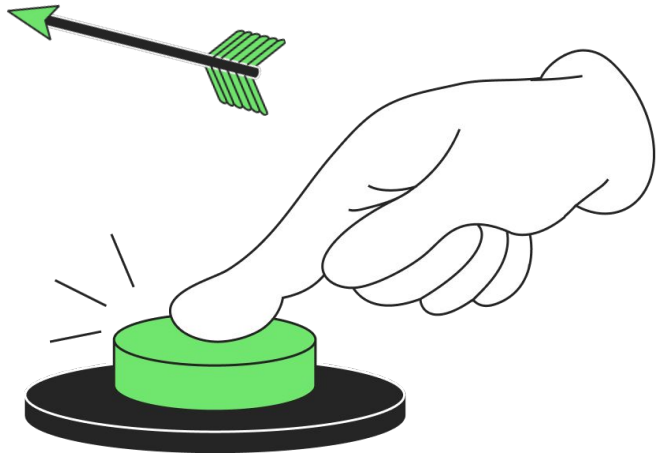
- ✔ final классы
- ✔ final методы
- ✔ static методы
- ✔ анонимные классы
- ✔ примитивные типы



Другие mocking-фреймворки

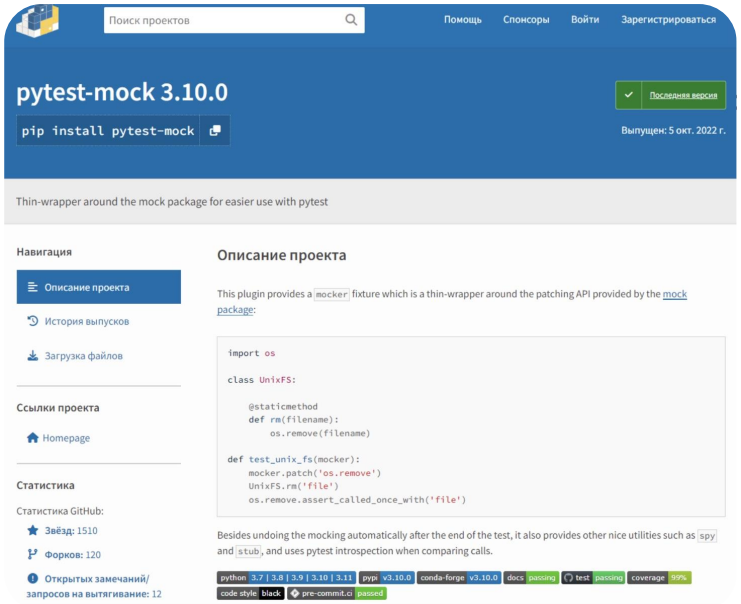
Фреймворки для моделирования тестовых зависимостей

- ✔ EasyMock
- ✔ WireMock
- ✔ MockWebServer
- ✔ PowerMock
- ✔ JMockit



Фреймворки для моделирования тестовых зависимостей в Python

- ✔ pytest-mock



Итоги занятия

Что было на уроке сегодня

- 📌 Зависимости в тестируемых системах их виды
- 📌 Тестовые заглушки (Test Double)
- 📌 Типы тестовых заглушек
- 📌 Фреймворк Mockito
- 📌 Практика создания и настройка моков с Mockito
- 📌 Mocking-фреймворки

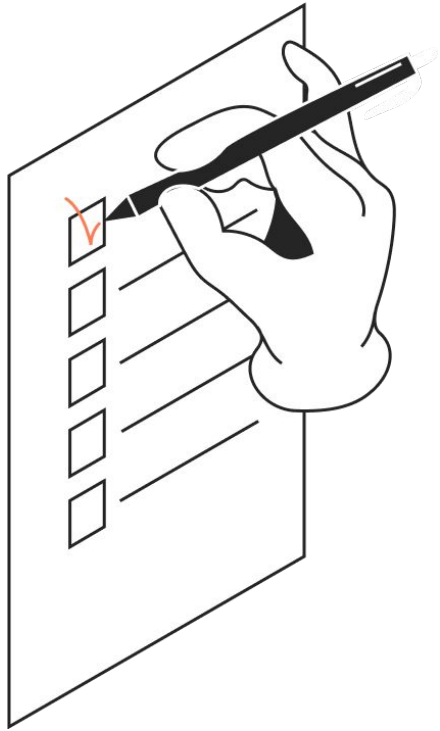
Вопросы?

Вопросы?

Вопросы?

Задание

Установите фреймворк и дополнительные зависимости, проверьте, что основной класс Mockito импортируется, попробуйте повторить примеры с урока



Спасибо за внимание