



## Модуль 06 - Piscine Java

### JUnit/Mockito

Резюме: Сегодня вы узнаете основы модульного и интеграционного тестирования

# Содержание

I	Предисловие	2
II	Инструкции	3
III	Правила дня	5
IV	Упражнение Первые тесты 00 :	6
V	Упражнение Встроенная база данных 01 :	8
VI	Упражнение Тест для репозитория JDBC 02 :	10
VII	Упражнение Испытание для службы 03 :	12



# Глава I

## Предисловие

Модульные и интеграционные тесты позволяют программисту обеспечить корректную работу создаваемых им программ. Эти методы тестирования выполняются автоматически.

Таким образом, ваша цель - не просто написать правильный код, но и создать код для проверки правильности вашей реализации.

Модульные тесты в Java - это классы, которые содержат несколько методов тестирования для публичных методов тестируемых классов. Каждый класс модульного теста должен проверять функциональность только одного класса. Такие тесты позволяют точно определить ошибки. Для выполнения тестов без конкретных зависимостей используются объекты-заглушки с временной реализацией.

В отличие от модульных тестов, интеграционные тесты позволяют проверять связки различных компонентов. Ниже приведены несколько лучших практик для модульного и интеграционного тестирования:

- Используйте адекватные названия для методов тестирования.
- Рассмотрите различные ситуации.
- Убедитесь, что тесты охватывают не менее 80% кода.
- Каждый метод тестирования должен содержать небольшую часть кода и выполняться быстро.
- Методы тестирования должны быть изолированы друг от друга и не иметь побочных эффектов.



## Глава II

### Инструкции

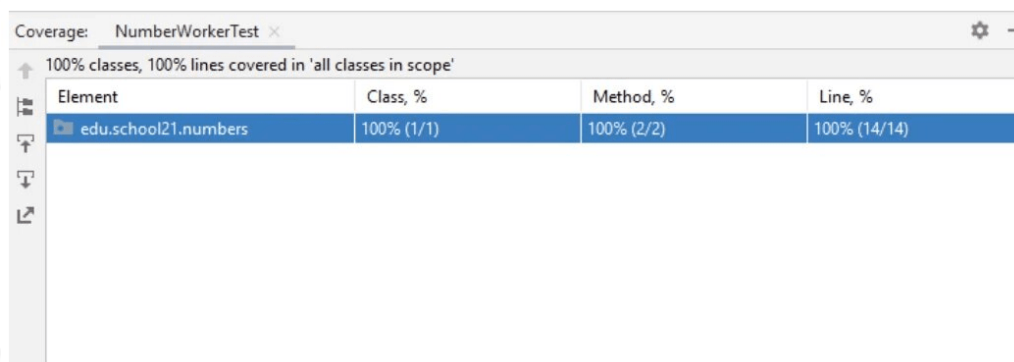
- Используйте эту страницу как единственную ссылку. Не слушайте никаких слухов и домыслов о том, как приготовить раствор.
- Теперь для вас существует только одна версия Java - 1.8. Убедитесь, что компилятор и интерпретатор этой версии установлены на вашей машине.
- Вы можете использовать IDE для написания и отладки исходного кода.
- Код чаще читают, чем пишут. Внимательно прочитайте [документ](#), в котором приведены правила форматирования кода. При выполнении каждой задачи убедитесь, что вы следуете общепринятым [стандартам Oracle](#)
- Комментарии не допускаются в исходном коде вашего решения. Они затрудняют чтение кода.
- Обратите внимание на разрешения ваших файлов и каталогов.
- Для оценки ваше решение должно находиться в вашем GIT-репозитории.
- Ваши решения будут оценивать ваши товарищи по аквариуму.
- Вы не должны оставлять в своем каталоге никаких других файлов, кроме тех, которые явно указаны в инструкциях к упражнению. Рекомендуется изменить свой .gitignore во избежание несчастных случаев.
- Когда вам нужно получить точный вывод в ваших программах, запрещено выводить предварительно рассчитанный вывод вместо правильного выполнения упражнения.
- У вас есть вопрос? Спросите своего соседа справа. В противном случае попробуйте поговорить с соседом слева.
- Ваше справочное пособие: товарищи / Интернет / Google. И еще кое-что. На любой ваш вопрос есть ответ на Stackoverflow. Узнайте, как правильно задавать вопросы.
- Внимательно прочитайте примеры. В них могут потребоваться вещи, которые не указаны в предмете.
- Используйте "System.out" для вывода

- И да пребудет с вами Сила!
- Никогда не оставляйте на завтра то, что вы можете сделать сегодня ;)

## Глава III

### Правила дня

- Используйте фреймворк JUnit 5 во всех задачах
- Используйте следующие зависимости и плагины для обеспечения правильной работы:
  - maven-surefire-plugin
  - junit-jupiter-engine
  - junit-jupiter-params
  - junit-jupiter-api
- Все тесты должны быть запускаемыми при выполнении команды `mvn clean compile test`
- Исходный код тестируемого класса должен быть полностью покрыт во всех реализованных тестах. Ниже приведен пример демонстрации полного покрытия с помощью IntelliJ IDEA для упражнения 00:



The screenshot shows the Coverage tool window in IntelliJ IDEA. The title bar reads 'Coverage: NumberWorkerTest'. Below the title bar, it states '100% classes, 100% lines covered in 'all classes in scope''. A table displays the coverage details for the element 'edu.school21.numbers'.


Element	Class, %	Method, %	Line, %
edu.school21.numbers	100% (1/1)	100% (2/2)	100% (14/14)





## Глава IV

### Упражнение 00: Первые тесты

	Упражнение
	00 Первые
Каталог для сдачи : ex00/	
Файлы для сдачи : Tests-	испытания
folder Разрешенные	
функции : Все	

Теперь вам нужно реализовать класс `NumberWorker`, который содержит следующую функциональность:

```
public boolean isPrime(int number) {  
    ...  
}
```

Этот метод определяет, является ли число простым, и возвращает `true/false` для всех натуральных (целых положительных) чисел. Для отрицательных чисел, а также `0` и `1`, программа выбрасывает непроверенное исключение. `IllegalNumberException`.

```
public int digitsSum(int number) {  
    ...  
}
```

Этот метод возвращает сумму цифр исходного числа.

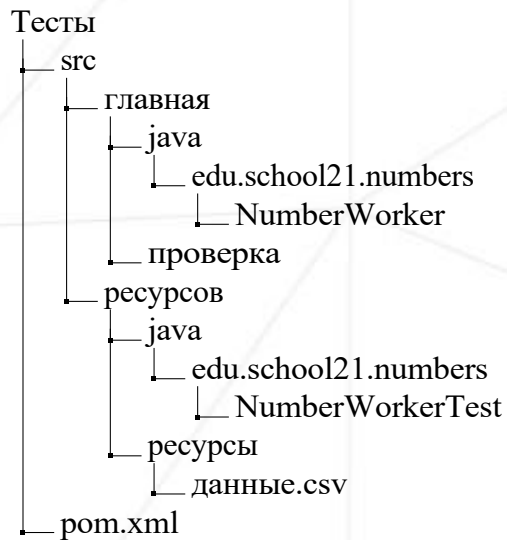
Нам также необходимо создать класс `NumberWorkerTest`, который реализует логику тестирования модуля. Методы класса `NumberWorkerTest` должны проверять правильность работы методов `NumberWorker` для различных входных данных:

1. метод `isPrimeForPrimes` для проверки `isPrime` с использованием простых чисел (не менее трех)
2. метод `isPrimeForNotPrimes` для проверки `isPrime` с использованием составных чисел (не менее трех)
3. метод `isPrimeForIncorrectNumbers` для проверки `isPrime` с использованием неправильных чисел (не менее трех)
4. метод проверки `digitsSum` с использованием набора из не

менее 10 чисел Требования:


- Класс `NumberWorkerTest` должен содержать не менее 4 методов для тестирования функциональности `NumberWorker`
- Использование `@ParameterizedTest` и `@ValueSource` обязательно для методов 1-3.
- Использование `@ParameterizedTest` и `@CsvFileSource` обязательно для метода 4.
- Необходимо подготовить файл `data.csv` для метода 4, в котором нужно указать не менее 10 чисел и их правильную сумму цифр. Пример содержимого файла:
  - 1234, 10

Структура проекта:



## Глава V

### Упражнение 01: Встроенная база данных

	Упражнение 01
Встроенная база	
Каталог для сдачи :	
ex01/	Файлы для сдачи :      данных
Тесты Разрешенные	
функции : Все	

Не используйте тяжелую СУБД (например, PostgreSQL) для проведения интеграционного тестирования компонентов, взаимодействующих с базой данных. Лучше всего создать легкую базу данных in-memory с заранее подготовленными данными.

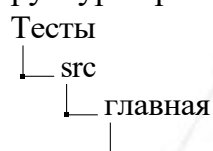
Реализуйте механизм создания DataSource для СУБД HSQL. Для этого подключите к проекту зависимости spring-jdbc и hsqldb. Подготовьте файлы schema.sql и data.sql, в которых вы опишите структуру таблиц продукта и тестовые данные (не менее пяти).

Структура таблицы продуктов:

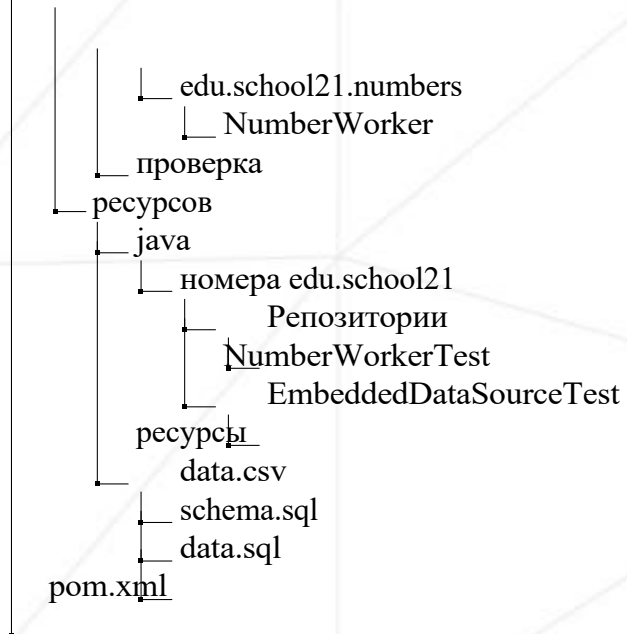
- идентификатор
- имя
- цена

Также создайте класс EmbeddedDataSourceTest. В этом классе реализуйте метод init(), помеченный аннотацией @BeforeEach. В этом классе реализуйте функциональность для создания DataSource с помощью EmbeddedDataBaseBuilder (класс из библиотеки spring-jdbc). Реализуйте простой метод тестирования для проверки возвращаемого значения метода getConnection(), созданного DataSource (это значение не должно быть null).

Структура проекта:




java



## Глава VI

### Упражнение 02: Тест для репозитория JDBC

	Упражнение 02
Тест для репозитория JDBC	
Каталог для сдачи :	
ex02/ Файлы для сдачи	
: Тесты Разрешенные	
функции : Все	

Реализуйте пару интерфейс/класс `ProductsRepository/ProductsRepositoryJdbcImpl` со следующими методами:

```
List<Product> findAll()
Optional<Product> findById(Long id)
void update(Product product)
void save(Product product)
void delete (Long id)
```

Вы должны реализовать класс `ProductsRepositoryJdbcImplTest`, содержащий методы, проверяющие функциональность репозитория с использованием базы данных in-memory, упомянутой в предыдущем упражнении. В этом классе необходимо заранее подготовить объекты модели, которые будут использоваться для сравнения во всех тестах.

Пример объявления тестовых данных приведен ниже:

```
class ProductsRepositoryJdbcImplTest {
    final List<Product> EXPECTED_FIND_ALL_PRODUCTS = ...; final
    Product EXPECTED_FIND_BY_ID_PRODUCT = ...;
    final Product EXPECTED_UPDATED_PRODUCT = ...;
}
```

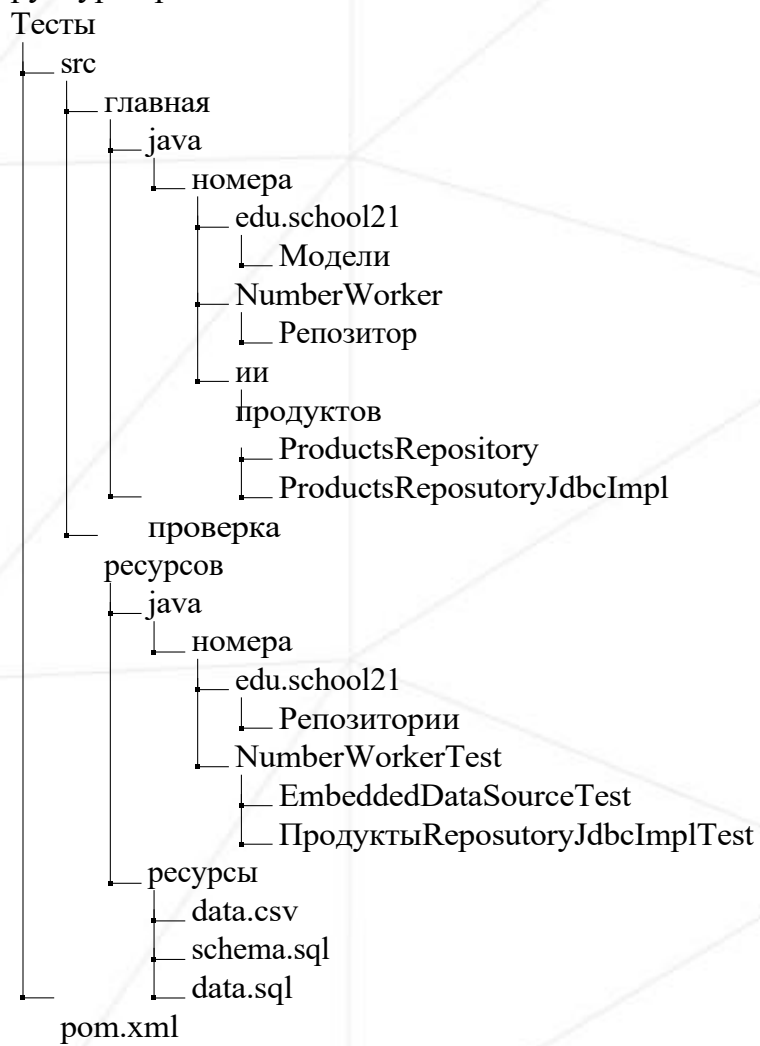
Примечания:

- Каждый тест должен быть изолирован от поведения других тестов. Таким образом, перед выполнением каждого теста база данных должна находиться в исходном состоянии.
- Тестовые методы могут вызывать другие методы, которые не относятся к текущему тесту. Например, при тестировании метода `update()` может быть вызван метод `findById()` для проверки достоверности обновления сущности в базе данных.






## Структура проекта:



## Глава VII

### Упражнение 03: Тест на обслуживание

	Упражнение 03
Испытание	
Каталог для сдачи :	
ex03/ Файлы для сдачи	для службы
: Тесты Разрешенные функции : Все	

Важное правило для модульных тестов: отдельный компонент системы должен тестироваться без вызова функциональности его зависимостей. Такой подход позволяет разработчикам создавать и тестировать компоненты независимо друг от друга, а также откладывать реализацию отдельных частей приложения.

Теперь необходимо реализовать уровень бизнес-логики, представленный классом `UserServiceImpl`. Этот класс содержит логику аутентификации пользователей. Он также имеет зависимость от интерфейса `UsersRepository` (в данной задаче вам не нужно реализовывать этот интерфейс).

Интерфейс `UsersRepository` (который вы описали) должен содержать следующие методы:

```
User findByLogin(String login);
void update(User user);
```

Предполагается, что метод `findByLogin` возвращает объект `Userobject`, найденный через логин, или выбрасывает `EntityNotFoundException`, если пользователь с указанным логином не найден. Метод `Update` выбрасывает аналогичное исключение при обновлении пользователя, не существующего в базе данных.

Пользовательская сущность должна содержать следующие поля:

- Идентификатор
- Вход в систему
- Пароль
- Статус успешной аутентификации (true - аутентифицирован, false - не

аутентифицирован) 12

В свою очередь, класс `UserServiceImpl` вызывает эти методы внутри функции аутентификации: `boolean authenticate(String login, String password)`

Этот метод:

1. Проверяет, был ли пользователь аутентифицирован в системе, используя данный логин. Если аутентификация была выполнена, должно быть выброшено исключение `AlreadyAuthenticatedException`.
2. Пользователь с таким логином извлекается из `UsersRepository`.
3. Если полученный пароль пользователя совпадает с указанным паролем, метод устанавливает статус успешной аутентификации для пользователя, обновляет его информацию в базе данных и возвращает `true`. Если пароли не совпадают, метод возвращает `false`.

Ваша цель состоит в том, чтобы:

1. Создание интерфейса `UsersRepository`
2. Создайте класс `UserServiceImpl` и метод аутентификации
3. Создайте модульный тест для класса `UserServiceImpl`

Поскольку ваша цель - проверить корректность работы метода `authenticate` независимо от компонента `UsersRepository`, вам следует использовать mock-объект и заглушки методов `findByLogin` и `update` (см. библиотеку Mockito).

Метод аутентификации должен быть проверен в трех случаях:

1. Правильный логин/пароль (проверьте вызов метода обновления с помощью инструкции `verify` библиотеки Mockito)
2. Неправильный вход в систему
3. Неверный пароль

Структура проекта:

