

# Лекция 8

## Разработка АС реального времени ( часть 2)

ФИО преподавателя: Зорина Наталья Валентиновна

e-mail: [zorina\\_n@mail.ru](mailto:zorina_n@mail.ru)

# Тема лекции:

## «Тестирование протоколирование и хранение пользователей в БД»

# Тестирование

- JUnit5
- JUnit4
- TestNG

# Тестирование с JUnit5

```
plugins {  
    id("java")  
}  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    testImplementation("org.junit.jupiter:junit-jupiter-api:5.7.1")  
    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine")  
}  
  
test {  
    useJUnitPlatform()  
}
```

# Тестирование с JUnit5

- `testImplementation` – классы, которые будут использоваться для компиляции тестового кода (`src/test/java`)
- `testRuntimeOnly` – классы, которые будут использоваться при выполнении тестового кода, но не при компиляции
- `useJUnitPlatform` указывает Gradle, что при выполнении задачи `test` нужно использовать JUnit 5

# Тестирование с JUnit5

```
package ru.mirea.testing;

import java.util.Objects;

public final class Example {

    private final String greeting;

    public Example(String greeting) {
        Objects.requireNonNull(greeting);
        this.greeting = greeting;
    }

    public String greet(String name) {
        Objects.requireNonNull(name);
        return String.format("%s %s!", greeting, name);
    }
}
```

# Тестирование с JUnit5

```
package ru.mirea.testing;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class SimpleExampleTest {

    @Test
    public void testGreeting() {
        Example example = new Example("Hello");
        String result = example.greet("world");
        assertEquals(result, "Hello world!");
    }
}
```

# Тестирование с JUnit5

- Тестовый код находится в папке `src/test/java`
- Тестовые методы помечаются аннотацией `@Test`
- Тестовые методы:
  - Вызывают проверяемый код (в нашем примере `Example.greet`)
  - Проверяют, что результат соответствует ожидаемому
  - Для проверки соответствия обычно используются методы класса `Assertions` (в нашем примере `Assertions.assertEquals`)
  - Так как эти методы в тестах используются часто, мы используем `static import` для сокращения записи кода

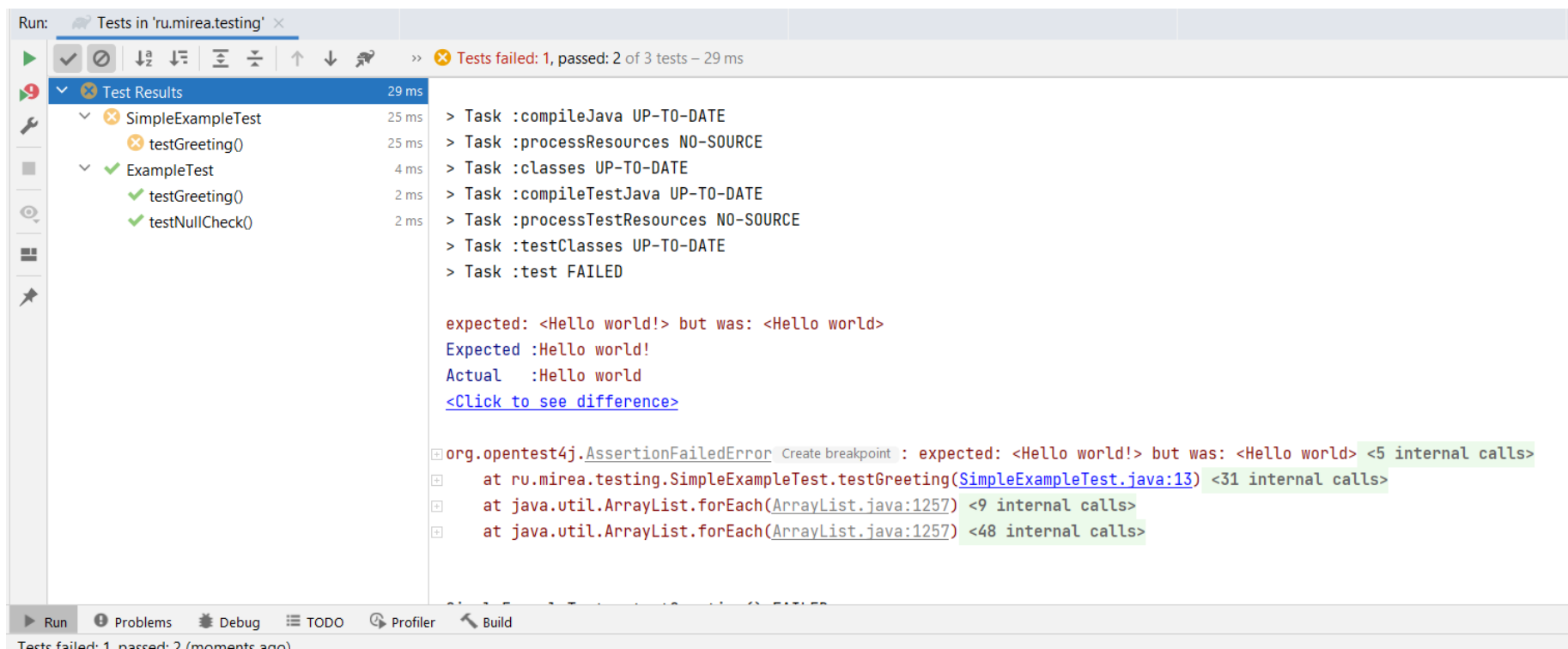


# Тестирование с JUnit5

Для запуска тестов используем команду  
`gradlew test`

Или можно запустить их из среды разработки  
(можно выполнить индивидуальные методы, все  
методы класса, или все методы всех тестовых  
классов)

# Тестирование с JUnit5



Run: Tests in 'ru.mirea.testing' x

Tests failed: 1, passed: 2 of 3 tests – 29 ms

Test Results

- SimpleExampleTest (25 ms)
  - testGreeting() (25 ms)
    - Task :compileJava UP-TO-DATE
    - Task :processResources NO-SOURCE
    - Task :classes UP-TO-DATE
    - Task :compileTestJava UP-TO-DATE
    - Task :processTestResources NO-SOURCE
    - Task :testClasses UP-TO-DATE
    - Task :test FAILED
- ExampleTest (4 ms)
  - testGreeting() (2 ms)
  - testNullCheck() (2 ms)

expected: <Hello world!> but was: <Hello world>  
Expected :Hello world!  
Actual :Hello world  
[<Click to see difference>](#)

org.opentest4j.AssertionFailedError: expected: <Hello world!> but was: <Hello world> <5 internal calls>  
at ru.mirea.testing.SimpleExampleTest.testGreeting(SimpleExampleTest.java:13) <31 internal calls>  
at java.util.ArrayList.forEach(ArrayList.java:1257) <9 internal calls>  
at java.util.ArrayList.forEach(ArrayList.java:1257) <48 internal calls>

Run Problems Debug TODO Profiler Build

Tests failed: 1, passed: 2 (moments ago)

# Тестирование с JUnit5

Аннотация `@BeforeEach` используется для метода, который будет вызываться перед вызовом каждого тестового метода. Она обычно используется для инициализации, общей для всех тестовых методов

# Тестирование с JUnit5

```
package ru.mirea.testing;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class ExampleTest {

    private Example example;

    @BeforeEach
    public void setUp() {
        example = new Example("Hello");
    }

    @Test
    public void testGreeting() {
        String result = example.greet("world");
        assertEquals(result, "Hello world!");
    }
}
```

# Тестирование с JUnit5

Иногда нужно проверить не результат метода, а его исключения.

Если нам нужно проверить отсутствие исключений, можно ничего не предпринимать – любое исключение, выбрасываемое из тестового (@Test) метода, считается ошибкой.

Если же нам нужно проверить наличие исключения (т.е. его отсутствие является ошибкой), можно использовать метод `Assertions.assertThrows`.

# Тестирование с JUnit5

@Test

```
public void testNullCheck() {  
    assertThrows(NullPointerException.class, () -> {  
        example.greet(null);  
    }, "Should throw NPE in case of null parameter");  
}
```

# Тестирование с JUnit5

Другие методы Assertions:

- `assertNotEquals(a, b)`
- `assertNull(a)`
- `assertNotNull(a)`
- `assertTrue(condition)`

Все методы Assertions имеют необязательный параметр (String message) или (Supplier<String> messageSupplier) – сообщение, которое выводится при нарушении проверки.

# Тестирование с JUnit5

Модульное тестирование, иногда блочное тестирование или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

В Java под модулем обычно понимается класс.



# Тестирование в Spring

<https://spring.io/guides/gs/testing-web/>

Тестирование полного приложения: <https://github.com/spring-guides/gs-testing-web/blob/main/complete/src/test/java/com/example/testingweb/HttpRequestTest.java>

- Используется аннотация  
`@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)`
- Используется поле  
`@LocalServerPort private int port;`
- Используется поле  
`@Autowired private TestRestTemplate restTemplate;`
- Тестируем, обращаясь через `restTemplate` к `localhost:<port>`

# Тестирование в Spring

<https://spring.io/guides/gs/testing-web/>

Тестирование без запуска сервера: <https://github.com/spring-guides/gs-testing-web/blob/main/complete/src/test/java/com/example/testingweb/TestingWebApplicationTest.java>

- Используются аннотации `@SpringBootTest` и `@AutoConfigureMockMvc`
- Используется поле `@Autowired private MockMvc mockMvc;`
- Тестируем, обращаясь к `mockMvc`:

```
this.mockMvc.perform(get("/")).andDo(print()).andExpect(status().isOk())  
    .andExpect(content().string(containsString("Hello, World")));
```

# Тестирование в Spring

Такие тесты, строго говоря, не являются юнит-тестами: они тестируют одновременно все слои приложения. Для тестирования только одного слоя используются заглушки: например, чтобы протестировать слой бизнес-логики, можно создать заглушку (mock) для слоя хранения данных.

Для создания заглушек используется библиотека Mockito, которая позволяет "подменить" реальный вызов методов объекта на тот, что возвращает некоторые тестовые значения.

# Тестирование в Spring

<https://spring.io/guides/gs/testing-web/>

Тестирование слоя представления с заглушкой для слоя бизнес-логики: <https://github.com/spring-guides/gs-testing-web/blob/main/complete/src/test/java/com/example/testingweb/WebMockTest.java>

- Используются аннотация `@WebMvcTest`
- Используется поле `@MockBean private GreetingService service;`
- Для задания поведения заглушки `GreetingService` при вызове метода `greet` используем Mockito:  
`when(service.greet()).thenReturn("Hello, Mock");`

# Протоколирование

Приложение в процессе работы обычно записывает протокол работы, используемый для:

- отладки приложения
- отслеживания возможных попыток взлома
- диагностики ошибок и других проблем

Протоколируемое сообщение характеризуется:

- Уровнем (TRACE, DEBUG, INFO, WARN, ERROR)
- Текстом сообщения
- Классом-источником

# Протоколирование

- SLF4J (Simple Logging Facade For Java)
- Log4J
- java.util.logging
- Apache Common Logging (Java Common Logging, JCL)

# Протоколирование

## Использование SLF4J:

```
dependencies {  
    implementation("org.slf4j:slf4j-api:1.7.30")  
}
```

## Объявление LOGGER:

```
private static final Logger LOGGER = LoggerFactory.getLogger(Example.class);
```

## Использование LOGGER:

```
LOGGER.info("Called greet with parameter: {}", name);
```

# Протоколирование

SLF4J API является только API без конкретной реализации. Для того, чтобы протокол куда-то записывался, нужно использовать реализацию SLF4J API. В простейшем случае можно использовать `slf4j-simple`, который просто выводит сообщения в КОНСОЛИ:

```
dependencies {  
    implementation("org.slf4j:slf4j-api:1.7.30")  
    runtimeOnly("org.slf4j:slf4j-simple:1.7.30")  
}
```



# Протоколирование

К сожалению, разные библиотеки используют разные библиотеки протоколирования. Например, библиотеки Apache Commons используют JCL, а не SLF4J. Таким образом, если мы захотим конфигурировать, куда выводить протокол, нам придется знать, какие библиотеки протоколирования используются и конфигурировать каждую из них по-отдельности.

Но для SLF4J есть другое решение – адаптеры для сторонних библиотек

# Протоколирование

Адаптеры, перенаправляющие протоколирование из других библиотек в SLF4J:

- Log4J: log4j-over-slf4j
- JCL: jcl-over-slf4j
- java.util.logging: jul-over-slf4j

# Протоколирование

Наиболее часто используемая реализация SLF4J –  
logback:

```
dependencies {  
    implementation("org.slf4j:slf4j-api:1.7.30")  
    runtimeOnly("ch.qos.logback:logback-classic:1.2.3")  
}
```

Logback позволяет управлять форматом файлов протокола, настраивать политику именования и создания файлов протокола. Настройка задается через файл .xml

# Хранение пользователей в БД

login="admin"

password="qwerty123"

Как проверить, что логин и пароль правильные? Будем хранить информацию о логине/пароле в БД.

Вариант 1: хранение пароля как есть (plain text)

id	login	password
120	admin	qwerty123



Такой вариант плох потому, что если содержимое БД утечет, по злоумышленники смогут легко войти под любым пользователем

# Хранение пользователей в БД

Вариант 2: использование криптографических хэш-функций

- MD5 – использование нежелательно, так как ее можно взломать на современных компьютерах
- SHA256/SHA512

SHA256("qwerty123")="B5439BBDD7A6A7401009A0FDC8DA8162FE6AE2918D109FBD178E4E60E5D6AADA"

Проверка правильности пароля:

```
if (SHA256(password) == password_hash) { ... }
```

id	login	password_hash
120	admin	B5439BBDD7A6A7401009A0FDC8DA8162FE6AE2918D109FBD178E4E60E5D6AADA

Такой вариант хорошо работает для сложных паролей, но простые пароли можно достаточно быстро подобрать перебором, если иметь доступ к БД, так как одинаковым паролям соответствуют одинаковые хэши

# Хранение пользователей в БД

```
public static byte[] sha(String password) throws NoSuchAlgorithmException {  
    MessageDigest alg = MessageDigest.getInstance("SHA256");  
    return alg.digest(password.getBytes(StandardCharsets.UTF_8));  
}
```

# Хранение пользователей в БД

Вариант 3: использование криптографических хэш-функций + соль

- bcrypt – добавляет в хэш случайное значение (“соль”), благодаря чему один и тот же пароль может иметь разный хэш

```
bcrypt_hash("qwerty123")="$2a$12$CKKDg6pP93xLm16KmF5lpOsrvtvejDepBXNwvM/K8Ap2fQzf9zHy"  
bcrypt_hash("qwerty123")="$2a$12$0Z07V8LI8OOHYVzhDymgte7NXStR6.let4W5uHGuxrYLW30DAhmFG"
```

Проверка правильности пароля:

```
if (bcrypt_verify(password, password_hash)) { ... }
```

id	login	password_hash
120	admin	\$2a\$12\$CKKDg6pP93xLm16KmF5lpOsrvtvejDepBXNwvM/K8Ap2fQzf9zHy

Нужно использовать отдельную библиотеку, реализующую bcrypt, например  
at.favre.lib:bcrypt

# Хранение пользователей в БД

Вычисление bcrypt\_hash:

```
BCrypt.Hasher hasher = BCrypt.with(  
    LongPasswordStrategies.truncate(BCrypt.Version.VERSION_2A)  
);  
String password = "qwerty123";  
String hash = hasher.hashToString(12, password.toCharArray());
```



# Хранение пользователей в БД

```
String login = ...;
```

```
String password = ...;
```

```
LoginUser user = /* загрузить из БД данные пользователя по login */;
```

```
BCrypt.Verifier verifier = BCrypt.verifier();
```

```
BCrypt.Result result = verifier.verify(  
    password.toCharArray(), user.passwordHash  
);
```

```
if (result.verified) {
```

```
    // Успешный вход
```

```
}
```

# Хранение пользователей в БД

@Bean

@Override

```
public UserDetailsService userDetailsService() {  
    PasswordEncoder encoder = passwordEncoder();  
    UserDetails user =  
        User.builder()  
            .username("user")  
            .password("password")  
            .passwordEncoder(encoder::encode)  
            .roles("USER")  
            .build();  
  
    return new InMemoryUserDetailsManager(user);  
}
```

@Bean

```
public PasswordEncoder passwordEncoder() {  
    return new BCryptPasswordEncoder();  
}
```