Utair

# 1 занятие

2 занятие Unit-тестирование

Изучаемые технологии:

* Junit
* Mockito3
* Hamcrest

Junit

Список аннотаций:

@Test – определяет что метод method() является тестовым.

@Before – указывает на то, что метод будет выполнятся перед каждым тестируемым методом @Test.

@After – указывает на то что метод будет выполнятся после каждого тестируемого метода @Test

@BeforeClass – указывает на то, что метод будет выполнятся в начале всех тестов,

а точней в момент запуска тестов(перед всеми тестами @Test).

@AfterClass – указывает на то, что метод будет выполнятся после всех тестов.

@Ignore – говорит, что метод будет проигнорирован в момент проведения тестирования.

(expected = Exception.class) – указывает на то, что в данном тестовом методе

вы преднамеренно ожидаете Exception.

(timeout = 100) – указывает, что тестируемый метод не должен занимать больше чем 100 миллисекунд.

Основные методы класса Assert для проверки:

fail(String) – указывает на то что бы тестовый метод завалился при этом выводя текстовое сообщение.

assertTrue([message], boolean condition) – проверяет, что логическое условие истинно.

assertsEquals([String message], expected, actual) – проверяет, что два значения совпадают.

Примечание: для массивов проверяются ссылки, а не содержание массивов.

assertNull([message], object) – проверяет, что объект является пустым null.

assertNotNull([message], object) – проверяет, что объект не является пустым null.

assertSame([String], expected, actual) – проверяет, что обе переменные относятся к одному объекту.

assertNotSame([String], expected, actual) – проверяет, что обе переменные относятся к разным объектам.

*Mockito* :

* создание заглушек для классов и интерфейсов;
* проверка вызыва метода и значений передаваемых методу параметров;
* использование концепции «частичной заглушки», при которой заглушка создается на класс с определением поведения, требуемое для некоторых методов класса;
* подключение к реальному классу «шпиона» *spy* для контроля вызова методов.

### when(mock).thenReturn(value) - позволяет определить возвращаемое значение при вызове метода *mock* с заданными параметрами

EX:

when(calc.add(10.0, 20.0)).thenReturn(30.0);

ИЛИ

doReturn(value).when(mock).method(params)

@Mock – определяет mock

@InjectMocks – создает mock объект

*Verify -*  позволяет проверить, была ли выполнена проверка с определенными параметрами. Если проверка не выполнялась или выполнялась с другими параметрами, то *verify* вызовет исключение.

EX:

verify(mcalc).add(10.0, 20.0); где mcalc – объект тестирования созданный с помощью @InjectMocks

Для проверки количества вызовов определенных методов *Mockito* предоставляет следующие методы :

* **atLeast (int min)** - не меньше min вызовов;
* **atLeastOnce ()** - хотя бы один вызов;
* **atMost (int max)** - не более max вызовов;
* **times (int cnt)** - cnt вызовов;
* **never ()** - вызовов не было;

EX:

* @Test
* public void testCallMethod()
* {
* // определяем поведение (результаты)
* when(mcalc.subtract(15.0, 25.0)).thenReturn(10.0);
* when(mcalc.subtract(35.0, 25.0)).thenReturn(-10.0);
* // вызов метода subtract и проверка результата
* assertEquals (calc.subtract(15.0, 25.0), 10, 0);
* assertEquals (calc.subtract(15.0, 25.0), 10, 0);
* assertEquals (calc.subtract(35.0, 25.0),-10, 0);
* // проверка вызова методов
* verify(mcalc, atLeastOnce()).subtract(35.0, 25.0);
* verify(mcalc, atLeast (2)).subtract(15.0, 25.0);
* // проверка - был ли метод вызван 2 раза?
* verify(mcalc, times(2)).subtract(15.0, 25.0);
* // проверка - метод не был вызван ни разу
* verify(mcalc, never()).divide(10.0,20.0);
* /\* Если снять комментарий со следующей проверки, то
* \* ожидается exception, поскольку метод "subtract"
* \* с параметрами (35.0, 25.0) был вызван 1 раз
* \*/
* // verify(mcalc, atLeast (2)).subtract(35.0, 25.0);
* /\* Если снять комментарий со следующей проверки, то
* \* ожидается exception, поскольку метод "subtract"
* \* с параметрами (15.0, 25.0) был вызван 2 раза, а
* \* ожидался всего один вызов
* \*/
* // verify(mcalc, atMost (1)).subtract(15.0, 25.0);
* }

### Обработка исключений - when(mock).thenThrow()

EX:

// создаем исключение

RuntimeException exception = new RuntimeException ("Division by zero");

// определение поведения для вызова исключения

doThrow(exception).when(mock).divide(5.0, 0));

### При сложном поведении – org.mockito.stubbing.Answer<T>

Иногда описание поведения *mock* объекта требует определенной проверки с усложнением логики. В этом случае можно использовать интерфейс **Answer<T>**, который позволяет реализовать заглушки методов со сложным поведением.

EX:

// метод обработки ответа

private Answer<Double> answer = new Answer<Double>() {

@Override

public Double answer(InvocationOnMock invocation) throws Throwable

{

// получение объекта mock

Object mock = invocation.getMock();

System.out.println ("mock object : " + mock.toString());

// аргументы метода, переданные mock

Object[] args = invocation.getArguments();

double d1 = (double) args[0];

double d2 = (double) args[1];

double d3 = d1 + d2;

System.out.println ("" + d1 + " + " + d2);

return d3;

}

};

@Test

public void testThenAnswer()

{

// определение поведения mock для метода с параметрами

when(mcalc.add(11.0, 12.0)).thenAnswer(answer);

assertEquals(calc.add(11.0,12.0), 23.0, 0);

}

*Mockito* позволяет подключать к реальным объектам «шпиона» **spy**, контролировать возвращаемые методами значения и отслеживать количество вызовов методов. (?)

### Timeout – Проверка вызова метода с задержкой

EX:

@Test

public void testTimout()

{

// определение результирующего значения mock для метода

when(mcalc.add(11.0, 12.0)).thenReturn(23.0);

// проверка значения

assertEquals(calc.add(11.0,12.0), 23.0, 0);

// проверка вызова метода в течение 10 мс

verify(mcalc, timeout(100)).add(11.0, 12.0);

}

Hamcrest

Def:

Matcher – это такое выражение, тестирующее на совпадение с определенным условием.

Импорты

import static org.hamcrest.MatcherAssert.assertThat;

import static org.hamcrest.Matchers.\*;

Логические matcher:

* *allOf()* – И
* *anyOf()* – ИЛИ
* *not()* – НЕ

Другие(для работы с объектами)

* *less/greaterThan*
* *less/greaterThenOrEqualTo*
* *equalTo*
* декоратор *is()*, он ничего не добавляет, кроме читабельности.
* notNullValue
* sameInstance(object)
* сравнивать числа с плавающей запятой в Java можно только с некоторой погрешностью, обычное равенство тут не работает. Для сравнения таких чисел существует матчер *closeTo()*.
* equalToIgnoringCase

*EX*:

* @Test
* public void givenNumber\_whenConditions\_thenCorrect() {
* Integer intVal = 7;
* assertThat(intVal, allOf(greaterThan(5), lessThanOrEqualTo(7), not(equalTo(6))));
* }

EX:

* @Test
* public void givenDouble\_whenCompare\_thenCorrect() {
* Double doubleVal = 7.70001;
* assertThat(doubleVal, closeTo(7.7, 0.005));
* }

Другие (для работы с коллекциями и массивами)

* empty
* hasSize
* everyItem
* hasItem
* contains

EX:

assertThat(list, everyItem(greaterThan(0)));

Специфичные для работы с Map

* hasKey
* hasValue
* hasEntry

Специфичные для работы с массивом

* emptyArray
* arrayWithSize
* arrayContainingInAnyOrder
* arrayContaining
* hasItemInArray

Специфичные для работы с пользовательскими классами:

* hasToString
* typeCompatibleWith
* isA

Java-bin(?)

Для сравнения bin-ов (предположительно, bin – объект класса)

@Test

public void givenBean\_whenCheckProperty\_thenHas() {

Animal animal = new Animal("gaf");

Dog dog = new Dog("gaf");

assertThat(animal, Matchers.<Animal> hasProperty("sound"));

assertThat(dog, Matchers.<Animal> hasProperty("sound"));

}

@Test

public void givenBean\_whenCheckPropertyValue\_thenEqual() {

Animal animal = new Animal("gaf");

assertThat(animal, Matchers.<Animal> hasProperty("sound", equalTo("gaf")));

}

@Test

public void given2Beans\_whenHavingSameValues\_thenCorrect() {

Animal animal1 = new Animal("gaf");

Animal animal2 = new Animal("gaf");

assertThat(animal1, samePropertyValuesAs(animal2));

}

Также есть возможность создать свой customMatcher - расширить класс *TypeSafeDiagnosingMatcher<T>*, где *T* –  тип объекта, который надо проверить

<https://stackoverflow.com/questions/156503/how-do-you-assert-that-a-certain-exception-is-thrown-in-junit-4-tests>

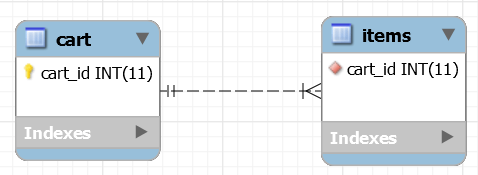
<https://stackoverflow.com/questions/27724660/how-to-use-hamcrest-in-java-to-test-for-a-exception>

<https://stackoverflow.com/questions/19256175/how-to-use-hamcrest-to-test-for-exception>

3 занятие Работа с Hibernate

***one-to-many* mapping means that one row in a table is mapped to multiple rows in another table**

**EX:**



public class Cart {

//...

@OneToMany(mappedBy="cart")

private Set<Items> items;

//...

}

The *mappedBy* property is what we use to tell Hibernate which variable we are using to represent the parent class in our child class.

@Entity

@Table(name=)

public class ItemsOIO {

// ...

@ManyToOne

@JoinColumn(name = "cart\_id", insertable = false, updatable = false)

private CartOIO cart;

//..

}

public class CartOIO {

//..

@OneToMany

@JoinColumn(name = "cart\_id") // we need to duplicate the physical information

private Set<ItemsOIO> items;

//..

}

@Id

@Column(name=)

@GeneratedValue

Crud:

@Service("personService")

@Transactional

public class PersonService {

protected static Logger logger = Logger.getLogger("service");

@Resource(name="sessionFactory")

private SessionFactory sessionFactory;

/\*\*

\* Retrieves all persons

\*

\* @return a list of persons

\*/

public List<Person> getAll() {

logger.debug("Retrieving all persons");

// Retrieve session from Hibernate

Session session = sessionFactory.getCurrentSession();

// Create a Hibernate query (HQL)

Query query = session.createQuery("FROM Person");

// Retrieve all

return query.list();

}

/\*\*

\* Retrieves a single person

\*/

public Person get( Integer id ) {

// Retrieve session from Hibernate

Session session = sessionFactory.getCurrentSession();

// Retrieve existing person

// Create a Hibernate query (HQL)

Query query = session.createQuery("FROM Person as p LEFT JOIN FETCH p.creditCards WHERE p.id="+id);

return (Person) query.uniqueResult();

}

/\*\*

\* Adds a new person

\*/

public void add(Person person) {

logger.debug("Adding new person");

// Retrieve session from Hibernate

Session session = sessionFactory.getCurrentSession();

// Persists to db

session.save(person);

}

/\*\*

\* Deletes an existing person

\* @param id the id of the existing person

\*/

public void delete(Integer id) {

logger.debug("Deleting existing person");

// Retrieve session from Hibernate

Session session = sessionFactory.getCurrentSession();

// Create a Hibernate query (HQL)

Query query = session.createQuery("FROM Person as p LEFT JOIN FETCH p.creditCards WHERE p.id="+id);

// Retrieve record

Person person = (Person) query.uniqueResult();

Set<CreditCard> creditCards =person.getCreditCards();

// Delete person

session.delete(person);

// Delete associated credit cards

for (CreditCard creditCard: creditCards) {

session.delete(creditCard);

}

}

/\*\*

\* Edits an existing person

\*/

public void edit(Person person) {

logger.debug("Editing existing person");

// Retrieve session from Hibernate

Session session = sessionFactory.getCurrentSession();

// Retrieve existing person via id

Person existingPerson = (Person) session.get(Person.class, person.getId());

// Assign updated values to this person

existingPerson.setFirstName(person.getFirstName());

existingPerson.setLastName(person.getLastName());

existingPerson.setMoney(person.getMoney());

// Save updates

session.save(existingPerson);

}

}

@Service("creditCardService")

@Transactional

public class CreditCardService {

protected static Logger logger = Logger.getLogger("service");

@Resource(name="sessionFactory")

private SessionFactory sessionFactory;

/\*\*

\* Retrieves all credit cards

\*/

public List<CreditCard> getAll(Integer personId) {

logger.debug("Retrieving all credit cards");

// Retrieve session from Hibernate

Session session = sessionFactory.getCurrentSession();

// Create a Hibernate query (HQL)

Query query = session.createQuery("FROM Person as p LEFT JOIN FETCH p.creditCards WHERE p.id="+personId);

Person person = (Person) query.uniqueResult();

// Retrieve all

return new ArrayList<CreditCard>(person.getCreditCards());

}

/\*\*

\* Retrieves all credit cards

\*/

public List<CreditCard> getAll() {

logger.debug("Retrieving all credit cards");

// Retrieve session from Hibernate

Session session = sessionFactory.getCurrentSession();

// Create a Hibernate query (HQL)

Query query = session.createQuery("FROM CreditCard");

// Retrieve all

return query.list();

}

/\*\*

\* Retrieves a single credit card

\*/

public CreditCard get( Integer id ) {

// Retrieve session from Hibernate

Session session = sessionFactory.getCurrentSession();

// Retrieve existing credit card

CreditCard creditCard = (CreditCard) session.get(CreditCard.class, id);

// Persists to db

return creditCard;

}

/\*\*

\* Adds a new credit card

\*/

public void add(Integer personId, CreditCard creditCard) {

logger.debug("Adding new credit card");

// Retrieve session from Hibernate

Session session = sessionFactory.getCurrentSession();

// Persists to db

session.save(creditCard);

// Add to person as well

// Retrieve existing person via id

Person existingPerson = (Person) session.get(Person.class, personId);

// Assign updated values to this person

existingPerson.getCreditCards().add(creditCard);

// Save updates

session.save(existingPerson);

}

/\*\*

\* Deletes an existing credit card

\*/

public void delete(Integer id) {

logger.debug("Deleting existing credit card");

// Retrieve session from Hibernate

Session session = sessionFactory.getCurrentSession();

// Delete reference to foreign key credit card first

// We need a SQL query instead of HQL query here to access the third table

Query query = session.createSQLQuery("DELETE FROM PERSON\_CREDIT\_CARD " +

"WHERE creditCards\_ID="+id);

query.executeUpdate();

// Retrieve existing credit card

CreditCard creditCard = (CreditCard) session.get(CreditCard.class, id);

// Delete

session.delete(creditCard);

}

/\*\*

\* Edits an existing credit card

\*/

public void edit(CreditCard creditCard) {

logger.debug("Editing existing creditCard");

// Retrieve session from Hibernate

Session session = sessionFactory.getCurrentSession();

// Retrieve existing credit card via id

CreditCard existingCreditCard = (CreditCard) session.get(CreditCard.class, creditCard.getId());

// Assign updated values to this credit card

existingCreditCard.setNumber(creditCard.getNumber());

existingCreditCard.setType(creditCard.getType());

// Save updates

session.save(existingCreditCard);

}

}

/\*\*

\* Retrieves the "Records" page

\*/

@RequestMapping(value = "/list", method = RequestMethod.GET)

public String getRecords(Model model) {

logger.debug("Received request to show records page");

// Retrieve all persons

List<Person> persons = personService.getAll();

// Prepare model object

List<PersonDTO> personsDTO = new ArrayList<PersonDTO>();

for (Person person: persons) {

// Create new data transfer object

PersonDTO dto = new PersonDTO();

dto.setId(person.getId());

dto.setFirstName(person.getFirstName());

dto.setLastName(person.getLastName());

dto.setMoney(person.getMoney());

dto.setCreditCards(creditCardService.getAll(person.getId()));

// Add to model list

personsDTO.add(dto);

}

// Add to model

model.addAttribute("persons", personsDTO);

// This will resolve to /WEB-INF/jsp/records.jsp

return "records";

}

<https://dou.ua/lenta/articles/how-to-use-hibernate/>

@Getter

@EqualsAndHashCode(of = {"firstName", "lastName"})

@ToString(of = {"firstName", "lastName"})

@NoArgsConstructor

@Entity

@Table(name = "clients")

public class Client {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(length = 32, nullable = false)

private String firstName;

@Column(length = 32, nullable = false)

private String lastName;

public Client(String firstName, String lastName) {

this.firstName = firstName;

this.lastName = lastName;

}

}

@Getter

@EqualsAndHashCode(of = "name")

@ToString(of = {"name", "price"})

@NoArgsConstructor

@Entity

@Table(name = "items")

public class Item {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, length = 32)

private String name;

@Columns(

columns = {

@Column(name = "price\_currency", length = 3, nullable = false),

@Column(name = "price\_amount", precision = 7, scale = 2, nullable = false)

}

)

@Type(type = "org.jadira.usertype.moneyandcurrency.moneta.PersistentMoneyAmountAndCurrency")

private Money price;

public Item(String name, Money price) {

this.name = name;

this.price = price;

}

}

@Getter

@EqualsAndHashCode(of = {"city", "street", "building"})

@ToString(of = {"city", "street", "building"})

@NoArgsConstructor

@AllArgsConstructor

@Embeddable

public class Address {

private String city;

private String street;

private String building;

}

@AllArgsConstructor

public enum Status {

NEW("N"),

PROCESSING("P"),

COMPLETED("C"),

DEFERRED("D");

@Getter

private final String code;

}

@Converter(autoApply = true)

public class StatusConverter implements AttributeConverter<Status, String> {

@Override

public String convertToDatabaseColumn(Status status) {

return status.getCode();

}

@Override

public Status convertToEntityAttribute(String code) {

for (Status status : Status.values()) {

if (status.getCode().equals(code)) {

return status;

}

}

throw new IllegalArgumentException("Unknown code " + code);

}

}

@Getter

@EqualsAndHashCode(of = {"created", "client"})

@ToString(of = {"created", "address", "express", "status"})

@NoArgsConstructor

@Entity

@Table(name = "orders")

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private LocalDateTime created = LocalDateTime.now();

@AttributeOverrides({

@AttributeOverride(name = "city", column = @Column(name = "address\_city", nullable = false, length = 32)),

@AttributeOverride(name = "street", column = @Column(name = "address\_street", nullable = false, length = 32)),

@AttributeOverride(name = "building", column = @Column(name = "address\_building", nullable = false, length = 32))

})

private Address address;

@Setter

private boolean express;

@Column(length = 1, nullable = false)

@Setter

private Status status = Status.NEW;

@ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.PERSIST, optional = false)

private Client client;

@ElementCollection

@Column(name = "quantity", nullable = false)

@MapKeyJoinColumn(name = "item\_id")

private Map<Item, Integer> items = new HashMap<>();

public Order(Address address, Client client) {

this.address = address;

this.client = client;

}

public Map<Item, Integer> getItems() {

return Collections.unmodifiableMap(items);

}

public void addItem(Item item) {

items.merge(item, 1, (v1, v2) -> v1 + v2);

}

public void removeItem(Item item) {

items.computeIfPresent(item, (k, v) -> v > 1 ? v - 1 : null);

}

}

Criteria API

TaskService.java – Как выглядит запрос с FindParams…

Task.java – Как создать таблицу в Idea Ultimate…

TODO:

Фильтрация – поля определяются из запроса(criteria api – сделано, см код)

Удаление связи – clear(см код)

4 занятие Быки и коровы

* Stream
* ParallelStream

TODO: почитать про них

5 занятие React

<https://ant.design/components/grid/>

<https://www.npmjs.com/package/react-hooks-useform>

TODO изучить пользовательские хуки

Reducer ⬄def описание структуры store

Persist сохраняет store в localStorage

UUID

Чистые функции

- не меняют тип

- не имеют side эффектов

Примером чистых функций в react является reducer

Оператор расширения используется для осуществления иммутабельности проекта

TODO изучить immutable.js

Payload – новая todo(в проекте смотри) – полезные данные

Контейнеры образуют области без верстки + навешивается логика

EX:

const todos = useSelector((state)=> state.todo.todos);

срез по state

Дз

- знакомиться с react, react-redux по документации

- JS нативные методы

- spread, Object.assign

- useSelector, useDispatch

- Context

- Деструктуризация через {}

- исправить код

7 занятие React

* Async await и аналогия с setTimeout
* EventLoop
* Fetch

Fetch

Ex(аналогия с fetch):

async function f() {

let promise = new Promise((resolve, reject) => {

setTimeout(() => resolve("готово!"), 1000)

});

*let result = await promise; // будет ждать, пока промис не выполнится (\*)*

alert(result); // "готово!"

}

f();

UI

fadeIn,fadeOut

hover нет в мобильных устройствах

TODO:

Перекрасить todo-программу

fadeOut – fadeIn при нажатии на switch и замена цвета

покрасить switch

добавить подзадачи и раскрыть с эффектом

раскрытие меню с дополнительными кнопками => удалить все, скопировать текст задач в определенном формате

перетаскивание элементов drag and drop

адрес/swagger-ui.html

config/package – вставить код corsfilter.java

ctrl(js), ctrl+shift+r(java)

добавление, удаление, сменить статус + данные из бд подгружать

CORS

Java:

@Component вызывает создание bin => класс создается и вызываются его методы

CorsRegistry, WebMVcConfigurer

8 занятие Docker

Технология виртуализации

Vmware

Xen

Vb

Запускается image ОС в контейнерах

Сохранение состояния в контейнерх через config

Docker hub – docker register – содержит готовые образцы (альпайны) (or nexus)

Dockerfile управляется docker compose

TODO

Почитать про kubernetes

В каждом контейнере – ОС

Ipconfig

docker build -t todo-front:latest .

docker run --name=todo-front -p 3000:3000 todo-front:latest

/\*

docker pull - с репозитория

\*/

build 40524192

docker-compose -v

image можно задавать ссылкой

docker-compose up – если изменить файлы => building

docker ps -a

mvn clean package – build java приложения в jar

MAVEN\_HOME системная переменная, Path %MAVEN\_HOME%\bin

docker-compose stop

ci/cd

TODO:

Почитать про gitlab-ci.yml, runner, pipline(ветка), job