

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

**ІКНІТ
Кафедра ПЗ**



Звіт

До лабораторної роботи №1

З дисципліни: *“Управління якістю програмного забезпечення”*

На тему: *«Створення автотестів з використанням TestNg бібліотеки»*

Лектор:
Ваврук І.Є.

Виконав:
ст. гр. ПЗП-11
Жуков Є.В.

Прийняв:
Ваврук І.Є.

« ____ » _____ 2022 р.

Σ = _____

Тема: «Створення автотестів з використанням TestNg бібліотеки»

ЗАВДАННЯ

1. Написати проект на мові Java (використовуючи maven), мінімум 5-10 класів-моделей та 10 класів-контролерів (які міститимуть логіку).
2. Кожен метод проекту в класі-контролері залогувати (використовуючи log4j бібліотеку).
3. Додати TestNg бібліотеку в dependency у файлі pom.xml.
4. Написати авто(юніт)-тести з використанням data provider (обов'язкове написання методів BeforeClass, AfterClass).
5. Забезпечити паралельність виконання тестів.
6. Написати тест, який буде очікувати Exception.
7. Написати тести, виконання яких залежить від інших тестів.
8. Забезпечити 90% покриття коду тестами.
9. Додати Maven Surefire Plugin для запуску тестів.
10. Налаштувати файл testing.xml для запуску тестів за допомогою maven стрічки.

ХІД ВИКОНАННЯ

1. Класи-моделі та класи-контролери (сервіси) із бізнес-логікою.

1.1. Класи-моделі

Для виконання поточної лабораторної роботи було створено наступні класи-моделі:

- Book;
- BookStore;
- Person;
- Employee;
- CompanyEmploy;
- Abs;
- Wheels;
- Engine;
- Car;
- CarMarket.

Наведемо лістинг дотичного до наведених вище класів-моделей вихідного коду.

Клас-модель Book

```
public class Book {  
  
    private List<String> authors;  
    private String title;  
    private String editionName;  
    private int editionYear;  
    private Category category;  
  
    public enum Category {  
        SCIENCE, FICTION  
    }  
}
```

Клас-модель BookStore

```
public class BookStore {  
  
    private String title;  
    private Map<Float, Book> priceList;  
}
```

Клас-модель Person

```
public class Person {  
  
    private LocalDateTime bornDate;  
    private String firstName;  
    private String lastName;  
}
```

Клас-модель Employee

```
public class Employee extends Person {  
  
    private final String taxId;  
    private final String profession;  
  
    @Builder  
    public Employee(LocalDateTime bornDate, String firstName,  
                    String lastName, String taxId, String profession) {  
        super(bornDate, firstName, lastName);  
        this.taxId = taxId;  
        this.profession = profession;  
    }  
}
```

Клас-модель CompanyEmploy

```
public class CompanyEmployee extends Employee {  
  
    private final Department department;  
  
    @Builder (builderMethodName = "companyBuilder")  
    public CompanyEmployee(LocalDateTime bornDate, String firstName,  
                            String lastName, String taxId,  
                            String profession, Department department) {  
        super(bornDate, firstName, lastName, taxId, profession);  
        this.department = department;  
    }  
  
    public enum Department {  
        FRONT_OFFICE, BACK_OFFICE  
    }  
}
```

Клас-модель Abs

```
@Getter
public class Abs implements SpareParts {

    private final String serialNumber;
    private final Type type;

    @Builder
    public Abs(String serialNumber, Type type) {
        this.serialNumber = serialNumber;
        this.type = type;
    }

    public enum Type {
        FOUR_CHANNEL_SIGNAL,
        THREE_CHANNEL_SIGNAL,
        ONE_CHANNEL_SIGNAL
    }
}
```

Клас-модель Car

```
@Getter
public class Car implements SpareParts {

    private final String producer;
    private final String model;
    private final Type type;
    private final Engine engine;
    private final Wheels wheels;

    @Builder
    public Car(String producer, String model, Type type,
               Engine engine, Wheels wheels) {
        this.producer = producer;
        this.model = model;
        this.type = type;
        this.engine = engine;
        this.wheels = wheels;
    }

    public enum Type {
        SUV, TRACK, SEDAN, VAN, COUPE, WAGON, CONVERTIBLE, SPORT,
        CROSSOVER, LUXURY
    }
}
```

Клас-модель Engine

```
@Getter
public class Engine implements SpareParts {

    private final String serialNumber;
    private final float capacity;

    @Builder
    public Engine(String serialNumber, float capacity) {
        this.serialNumber = serialNumber;
        this.capacity = capacity;
    }
}
```

Клас-модель CarMarket

```
@Getter
public class CarMarket {

    private final List<SpareParts> spareParts;

    @Builder
    public CarMarket(List<SpareParts> spareParts) {
        this.spareParts = spareParts;
    }
}
```

Клас-модель Wheels

```
@Getter
public class Wheels implements SpareParts {

    private final String serialNumber;
    private final Type type;

    @Builder
    public Wheels(String serialNumber, Type type) {
        this.serialNumber = serialNumber;
        this.type = type;
    }

    public enum Type {
        STEEL, ALLOY, MULTI_PIECE, CHROME, DIAMOND, FORGED,
        REPLICA_OEM
    }
}
```

1.2. Класи-контролери (сервіси)

Для нашарування бізнес-логіки до зазначених у попередньому пункті класів-моделей було реалізовано наступні класи-контролери (сервіси):

- BookService;
- BookStoreService;
- PersonService;
- EmployeeService;
- CompanyEmployService;
- AbsService;
- WheelsService;
- EngineService;
- CarService;
- CarMarketService.

Наведемо лістинг дотичного до наведених вище класів-моделей вихідного коду.

Клас-контролер BookService

```
public class BookService {  
  
    private static final Logger log =  
        LogManager.getLogger(BookService.class.getName());  
  
    public List<Book> filterByCategory(List<Book> books, Book.Category filter) {  
        if (Objects.isNull(books)) {  
            throw new BooksNotGivenException("Null passed instead of books");  
        } else if (books.isEmpty()) {  
            log.warn("No books given! An empty list will be returned.");  
            return Collections.emptyList();  
        }  
  
        return books.stream()  
            .filter(book -> book.getCategory().equals(filter))  
            .collect(Collectors.toList());  
    }  
}
```

Клас-контролер BookStoreService

```
public class BookStoreService {

    private static final Logger log =
        LogManager.getLogger(BookStoreService.class.getName());

    private BookStore bookStore;

    public Map<Float, Book> findByPriceRange(float min, float max) {
        if (min > max) {
            throw new IncorrectPriceListRangeException(
                "Min price-list value should be less, than Max");
        } else if (bookStore.getPriceList().isEmpty()) {
            log.warn("Price-list is empty");
            return Collections.emptyMap();
        }

        return bookStore.getPriceList().entrySet().stream()
            .filter(priceListEntry ->
                priceListEntry.getKey() >= min
                && priceListEntry.getKey() <= max)
            .collect(
                Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue));
    }
}
```

Клас-контролер PersonService

```
public class PersonService {
    private static final Logger log =
        LogManager.getLogger(PersonService.class.getName());

    private static final int ADULT_AGE = 18;

    public long getAge(Person person) {
        if (Objects.isNull(person)) {
            log.warn("Null Person instance passed");

            return 0;
        }
        LocalDateTime bornDate = person.getBornDate();
        LocalDateTime currentDate = LocalDateTime.now();
        long age =
            ChronoUnit.YEARS.between(bornDate, currentDate);

        if (age < 0) {
            throw new IncorrectPersonAgeException(
                "Incorrect Person Born Date passed! It should not be in future!");
        }

        return age;
    }

    public boolean isAdult(Person person) {
        if (Objects.isNull(person)) {
            log.warn("Null Person instance passed");

            return false;
        }
        long personAge = getAge(person);

        return personAge >= ADULT_AGE;
    }
}
```


Клас-контролер EmployeeService

```
public class EmployeeService {

    private static final Logger log =
        LogManager.getLogger(EmployeeService.class.getName());
    private List<Employee> employeeList;

    public Employee findByTaxId(String taxId) {
        if (StringUtils.isBlank(taxId)) {
            throw new NotGivenTaxIdException(
                "taxId is blank! (null or empty string)");
        } else if (employeeList.isEmpty()) {
            log.warn("employee-list is empty");
            return null;
        }

        return employeeList.stream()
            .filter(employee -> employee.getTaxId().equals(taxId))
            .findFirst()
            .orElse(null);
    }
}
```

Клас-контролер CompanyEmployeeService

```
public class CompanyEmployeeService {

    private static final Logger log =
        LogManager.getLogger(CompanyEmployeeService.class.getName());

    public List<CompanyEmployee> findByDepartment(
        List<CompanyEmployee> companyEmployees,
        CompanyEmployee.Department department) {
        if (Objects.isNull(department)) {
            throw new CompanyDepartmentNotGivenException(
                "Min price-list value should be less, than Max");
        } else if (companyEmployees.isEmpty()) {
            log.warn("Price-list is empty");
            return Collections.emptyList();
        }

        return companyEmployees.stream()
            .filter(companyEmployee ->
                companyEmployee.getDepartment().equals(department))
            .collect(Collectors.toList());
    }
}
```

Клас-контролер AbsService

```
@Getter
public class AbsService {

    private static final Logger log =
        LogManager.getLogger(AbsService.class.getName());

    public List<Abs> findByType(List<Abs> absList,
                               Abs.Type type) {
        if (Objects.isNull(type)) {
            throw new AbsTypeNotGivenException("Abs type is null!");
        } else if (absList.isEmpty()) {
            log.warn("Abs-list is empty");
            return Collections.emptyList();
        }

        return absList.stream()
            .filter(abs -> abs.getType().equals(type))
            .collect(Collectors.toList());
    }
}
```

Клас-контролер CarService

```
@Getter
public class CarService {

    private static final Logger log =
        LogManager.getLogger(CarService.class.getName());

    public Car findByType(List<Car> carList,
                          Car.Type caType) {
        if (Objects.isNull(caType)) {
            throw new CarTypeNotGivenException(
                "Car type is null!");
        } else if (carList.isEmpty()) {
            log.warn("Car-list is empty");
            return null;
        }

        return carList.stream()
            .filter(car -> car.getType().equals(caType))
            .findFirst()
            .orElseGet(() -> {
                log.warn("No any Engine found by car type {}",
                    caType);
                return null;
            });
    }
}
```

Клас-контролер EngineService

```
@Getter
public class EngineService {

    private static final Logger log =
        LogManager.getLogger(EngineService.class.getName());

    public Engine findBySerialNumber(List<Engine> absList,
                                     String serialNumber) {
        if (StringUtils.isBlank(serialNumber)) {
            throw new EngineNotGivenException(
                "Engine type is null!");
        } else if (absList.isEmpty()) {
            log.warn("Engine-list is empty");
            return null;
        }

        return absList.stream()
            .filter(engine ->
                engine.getSerialNumber().equals(serialNumber))
            .findFirst()
            .orElseGet(() -> {
                log.warn(
                    "No any Engine found by serial number {}",
                    serialNumber);
                return null;
            });
    }
}
```

Клас-контролер WheelsService

```
@Getter
public class WheelsService {

    private static final Logger log =
        LogManager.getLogger(WheelsService.class.getName());

    public List<Wheels> findByType(List<Wheels> wheelsList,
                                    Wheels.Type type) {
        if (Objects.isNull(type)) {
            throw new WheelsTypeNotGivenException(
                "Wheels type is null!");
        } else if (wheelsList.isEmpty()) {
            log.warn("Wheels-list is empty");
            return Collections.emptyList();
        }

        return wheelsList.stream()
            .filter(wheels -> wheels.getType().equals(type))
            .collect(Collectors.toList());
    }
}
```

Клас-контролер CarMarketService

```
@Getter
public class CarMarketService {

    private static final Logger log =
        LogManager.getLogger(CarMarketService.class.getName());

    public <T extends SpareParts> List<T>
        findByType(List<SpareParts> spareParts,
                   Class<T> sparePartsType) {
        if (Objects.isNull(sparePartsType)) {
            throw new SparePartsTypeNotGivenException(
                "Spare parts type is null!");
        } else if (spareParts.isEmpty()) {
            log.warn("Spare parts-list is empty");
            return Collections.emptyList();
        }

        return spareParts.stream()
            .filter(sparePart ->
                sparePart.getClass().equals(sparePartsType))
            .map(sparePartsType::cast)
            .collect(Collectors.toList());
    }
}
```

2. Логування класів-контролерів за допомоги бібліотеку log4j

Для логування класів-контролерів, а також класів-тестів, що покривають бізнес-логіку перших, було використано базову конфігурацію бібліотеки log4j, де лише LOG_PATTERN змінено для символічної кастомізації.

Log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
  <Properties>
    <Property name="LOG_PATTERN">%d{yyyy-MM-dd'T'HH:mm:ss} %p %m%n
  </Property>
</Properties>

  <Appenders>
    <Console name="console" target="SYSTEM OUT" follow="true">
      <PatternLayout pattern="%${LOG_PATTERN}"/>
    </Console>
  </Appenders>

  <Loggers>
    <Root level="info">
      <AppenderRef ref="console"/>
    </Root>
  </Loggers>
</Configuration>
```

Конкретні ж випадки застосування логування вичерпуються рівнями INFO (тест-класи) та WARN (класи-контролери). Дотичний вихідний останнього можна побачити у лістингах попереднього пункту.

У класах-тестах логування було використано на рівні методів, що мають анотації BeforeClass, AfterClass, а також у класі-аспекті LoggingAspect. Останній використовує логування для демонстрації паралельного виконання тестів.

LoggingAspect

```
public class LoggingAspect {
    private static final Logger log =
        LogManager.getLogger(LoggingAspect.class.getName());

    @Pointcut("@annotation(org.testng.annotations.Test)")
    public void callAt() {
    }

    @Around(value = "callAt()")
    public Object around(ProceedingJoinPoint pjp) throws Throwable {
        MethodSignature signature = (MethodSignature) pjp.getSignature();
        Method method = signature.getMethod();

        log.info(String.format("The thread ID for %s is %d", method.getName(),
            Thread.currentThread().getId()));

        return pjp.proceed();
    }
}
```

3. Додати TestNg та інші бібліотеки в dependencies у файлі pom.xml/build.gradle

Для забезпечення покриття бізнес-логіки класів-контролерів юніт-тестами та на виконання поточної умови лабораторної роботи до проекту було долучено (з-поміж інших) бібліотеку TestNG.

Project dependencies

```
dependencies {
    compileOnly group: 'org.projectlombok', name: 'lombok', version: '1.18.22'
    annotationProcessor 'org.projectlombok:lombok:1.18.22'
    testAnnotationProcessor 'org.projectlombok:lombok:1.18.22'

    implementation group: 'org.aspectj', name: 'aspectjweaver',
        version: '1.9.7'
    implementation group: 'org.aspectj', name: 'aspectjrt', version: '1.9.7'

    implementation group: 'org.apache.commons', name: 'commons-lang3',
        version: '3.0'

    implementation group: 'org.apache.logging.log4j', name: 'log4j-api',
        version: '2.17.1'
    implementation group: 'org.apache.logging.log4j', name: 'log4j-core',
        version: '2.17.1'
    testImplementation group: 'org.apache.logging.log4j',
        name: 'log4j-slf4j-impl',
        version: '2.17.1'

    testImplementation group: 'org.testng', name: 'testng', version: '7.5'
}
```

4. Написати авто(юніт)-тести з використанням data provider (обов'язкове написання методів BeforeClass, AfterClass)

В усіх класах-тестах було використано методи із анотаціями BeforeClass та AfterClass для логування початку та завершення виконання тест-кейсів.

Наведемо приклад з класу BookServiceTest:

```
@BeforeClass
public void before() {
    log.info("[{}] Starting test cases run...",
        getClass().getName());
}

@AfterClass
public void after() {
    log.info("[{}] Test cases run finished!",
        getClass().getName());
}
```

```
@DataProvider(name = "booksDataProvider")
public Object[][] booksDataProvider() {
    return new Object[][]{
        {
            new Book(Collections.emptyList(), null, null, 2022,
                Book.Category.FICTION),
            new Book(Collections.emptyList(), null, null, 2022,
                Book.Category.SCIENCE),
            new Book(Collections.emptyList(), null, null, 2022,
                Book.Category.SCIENCE)
        }
    };
}
```

5. Забезпечити паралельність виконання тестів

На забезпечення паралельного виконання тестів, було використано спеціальні налаштування TestNG у файлі suite.xml.

suite.xml

```
<suite name="Parallel Testing Suite">
  <test name="Parallel Tests" parallel="methods" thread-count="10">
    <classes>
      <class name="com.lpnu.swqm.services.PersonServiceTest"/>
      <class name="com.lpnu.swqm.services.BookServiceTest"/>
      <class name="com.lpnu.swqm.services.BookStoreServiceTest"/>
      <class name="com.lpnu.swqm.services.EmployeeServiceTest"/>
      <class name="com.lpnu.swqm.services.CompanyEmployeeServiceTest"/>
      <class name="com.lpnu.swqm.services.WheelsServiceTest"/>
      <class name="com.lpnu.swqm.services.AbsServiceTest"/>
      <class name="com.lpnu.swqm.services.EngineServiceTest"/>
      <class name="com.lpnu.swqm.services.CarServiceTest"/>
      <class name="com.lpnu.swqm.services.CarMarketServiceTest"/>
    </classes>
  </test>
</suite>
```

6. Написати тест, який буде очікувати Exception

У класах тестах є низка тест-кейсів, що передбачають очікування кастомних RuntimeException-класів. Наведемо деякі з них:

```
@Test(expectedExceptions = BooksNotGivenException.class)
public void testFilterByCategoryNullBooks() {
    bookService.filterByCategory(null, Book.Category.SCIENCE);
}
```

```
@Test(expectedExceptions =
CompanyDepartmentNotGivenException.class)
public void testFindByDepartmentFailDepartmentIsNull() {
    CompanyEmployeeService companyEmployeeService =
        new CompanyEmployeeService();
    companyEmployeeService.findByDepartment(null, null);
}
```


7. Написати тести, виконання яких залежить від інших тестів

На виконання поточної умови поточної лабораторної роботи було також додано тест-кейс, що залежить від результату іншого:

```
@Test(dataProvider = "personDataProvider")
public void testGetAge(Person person) {

    long actual = personService.getAge(person);
    long expected = 40;

    Assert.assertEquals(actual, expected);
}

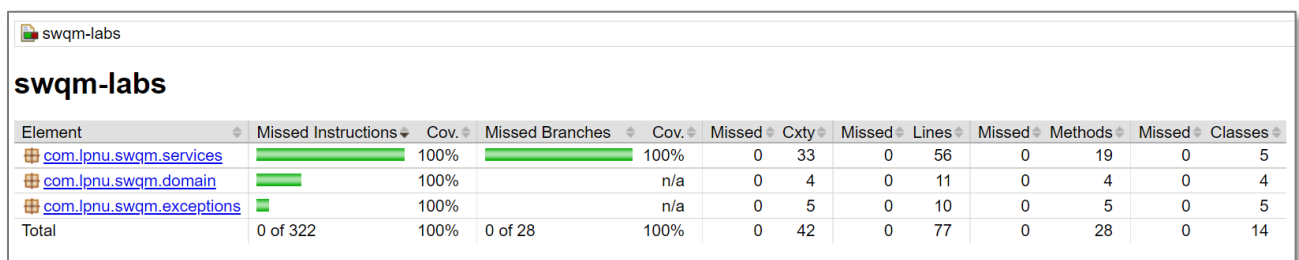
@Test(
    dataProvider = "personDataProvider",
    dependsOnMethods = "testGetAge")
public void testIsAdult(Person person) {

    boolean isAdultPerson = personService.isAdult(person);

    Assert.assertTrue(isAdultPerson);
}
```

8. Забезпечити 90% покриття коду тестами

На визначення ступеня покриття юніт-тестами класів-контролерів було використано плагін Ясосо. Відповідно до звітів останнього рівень покриття був визначений значенням 100%:



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.ljnu.swqm.services	<div></div>	100%	<div></div>	100%	0	33	0	56	0	19	0	5
com.ljnu.swqm.domain	<div></div>	100%		n/a	0	4	0	11	0	4	0	4
com.ljnu.swqm.exceptions	<div></div>	100%		n/a	0	5	0	10	0	5	0	5
Total	0 of 322	100%	0 of 28	100%	0	42	0	77	0	28	0	14

9. Додати Maven Surefire Plugin/Gradle “test” task для запуску тестів

Оскільки Gradle Build Tool вже має по замовченню завдання, що виконує усі юніт-тести, залучення та налаштування будь-яких інших додаткових плагінів не було необхідним.

10. Налаштувати файл `testing.xml` для запуску тестів за допомогою `maven` стрічки

Виходячи з тези, визначеної у попередньому пункті, Gradle Build Tool дозволяє запустити всі юніт-тести без додаткових налаштувань TestNG простою командою

```
gradle clean test
```

ВИСНОВКИ

За результатами виконання поточної лабораторної роботи можна засвідчити виконання всіх її умов у повному обсязі.

У даному звіті були наведені лише окремі частини коду цілого проекту лабораторної роботи. Ознайомитися із повною версією вихідного коду можна за наступним посиланням:

<https://github.com/yevhen-zhukov-mpzip-2021/swqm-labs>