Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**ЗВІТ**

**Про виконання лабораторної роботи № 1:**

**«Створення автотестів з використанням TestNg бібліотеки»**

**з дисципліни «Управління якістю програмного забезпечення»**

**Лектор:**

Ваврук І. Є.

**Виконав:**

студент групи ПЗІП-11

Бойчук Т.В.

**Прийняла:**

Ваврук І. Є.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2019

**Мета роботи:** Розробити java проект з використанням maven та log4j, створити автотести з використанням TestNg бібліотеки та Maven Surefire плагіну

**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 90% покриття коду тестами.
6. Додати Maven Surefire Plugin для запуску тестів.
7. Налаштувати файл testing.xml для запуску тестів за допомогою maven стрічки.
8. Сформувати звіт.

**ВИКОНАННЯ**

* + - 1. Написати проект на мові Java (використовуючи maven), мінімум 5-10 класів-моделей та 10 класів-контролерів (які міститимуть логіку).

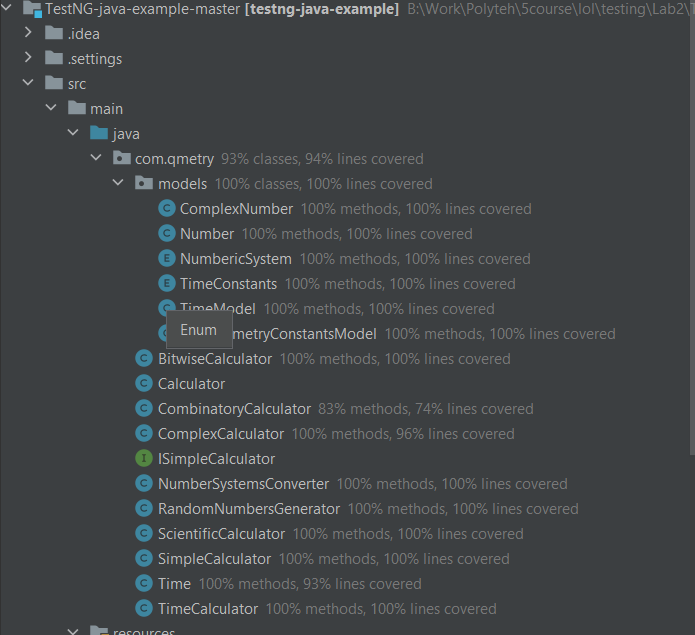
Було створено тестовий проект, який містить класи-калькулятори (Контролери) різного виду та інші допоміжні контролери:

* Простий калькулятор
* Науковий калькулятор
* Калькулятор для комплексних чисел
* Калькулятор часу (загальний)
* Калькулятор часу (допоміжний)
* Калькулятор переводу чисельних систем
* Комбінаторний калькулятор
* Генератор випадкових чисел
* Побітовий калькулятор
* Загальний калькулятор

Також було розроблено 6 класів моделей, які використовуються контролерами:

* Комплексні числа
* Число
* Чисельна система
* Часові константи
* Час
* Тригонометричні константи

Структура проекту в IDEA IntelliJ представлена на рис. 1.



*Рис. 1. Структура моделей та контролерів*

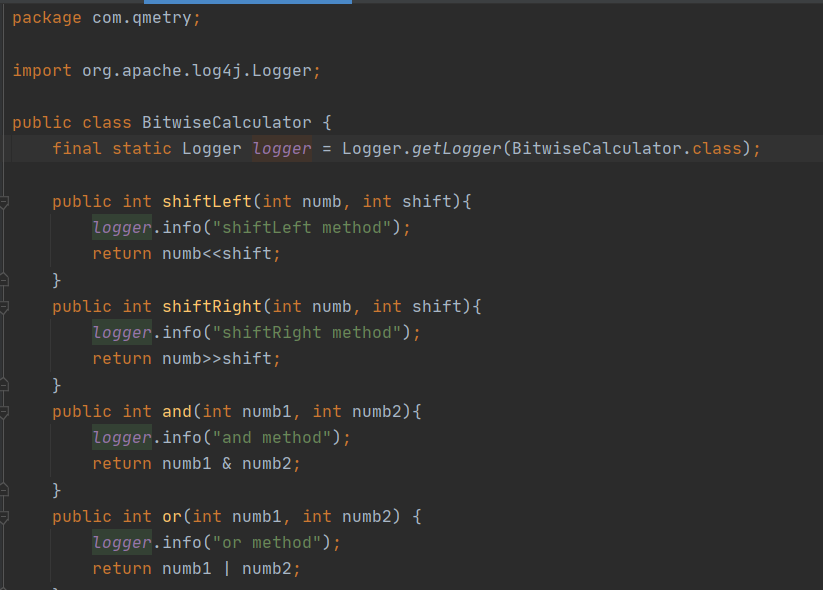
* + - 1. Кожен метод проекту в класі-контролері залогувати (використовуючи log4j бібліотеку)

В кожному методі класу контролера було викликано функцію бібліотеки log4j для здійснення логування інформації про те, що метод був виконаний. Інформація виводилася у файл log.log.

Спершу, в POM.xml файлі було додано dependency:

<dependency>  
 <groupId>log4j</groupId>  
 <artifactId>log4j</artifactId>  
 <version>1.2.17</version>  
</dependency>

Потім було створено log4j.properties файл, в якому було задано опції виконання бібліотеки. Приклад логування у коді (рис. 2)



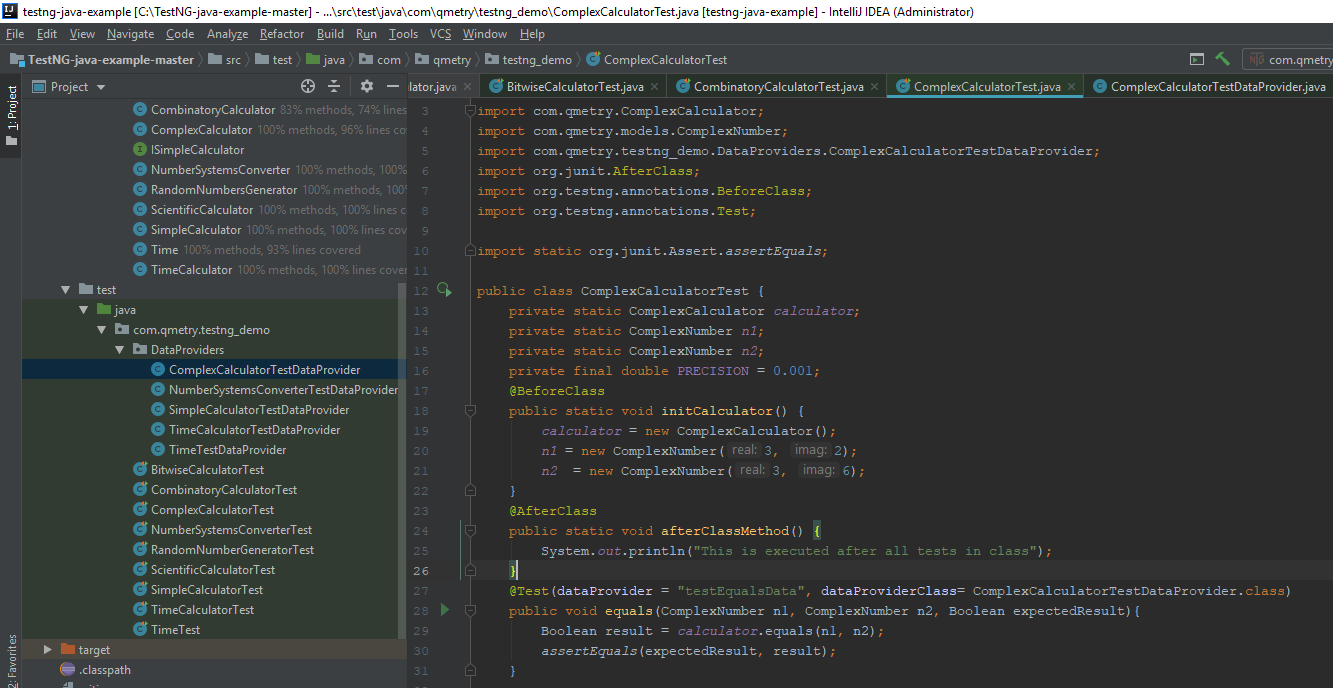
*Рис. 2. Логування з допомогою бібліотеки log4j*

* + - 1. Додати TestNg бібліотеку в dependency у файлі pom.xml.

Для того, щоб використовувати бібліотеку TestNg, подібно до log4j, у файл POM.xml було додано dependency

<dependency>  
 <groupId>org.testng</groupId>  
 <artifactId>testng</artifactId>  
 <version>6.8</version>  
 <scope>test</scope>  
</dependency>

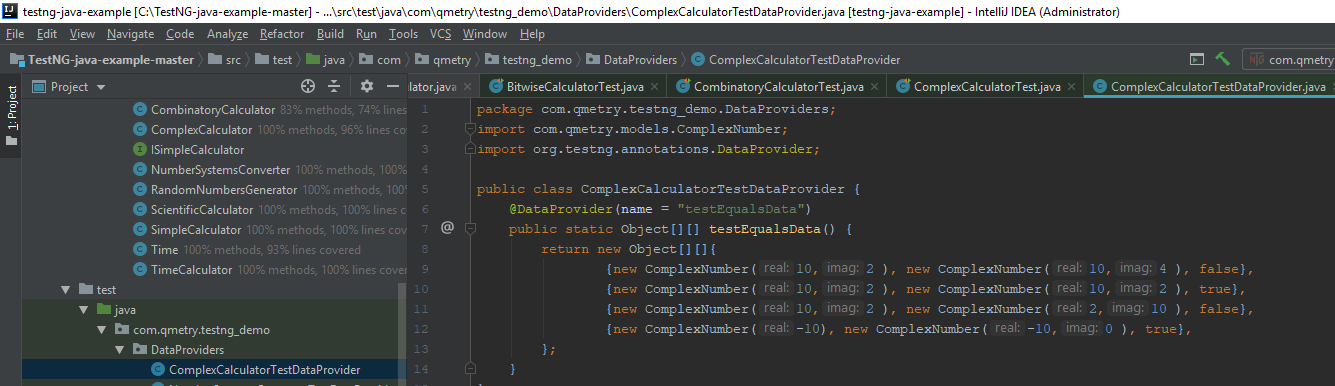
* + - 1. Написати авто(юніт)-тести з використанням data provider (обов’язкове написання методів BeforeClass, AfterClass).

В результаті, з для 16 класів, які містять 66 методів допомогою бібліотеки TestNg було розроблено 55 юніт тестів. Структура проекту з тестами та приклад класу з тестами представлений на рис. 3. 

*Рис. 3. Юніт-тести для тестування класу ComplexCalculator*

Тут також можна побачити використання анотацій TestNg: @beforeClass, @afterClass. @beforeClass в даному випадку застосовується для методу, який здійснює ініціалізацію об’єктів, необхідних для використання у багатьох тестових випадках поточного класу.

Також, для оптимізації завдання - виконання одного тесту для множини вхідних даних-було використано TestNg data providers. Виклик data provider можна побачити на рис. 3. Створення data provider для поданого тесту представлено на рис. 4.

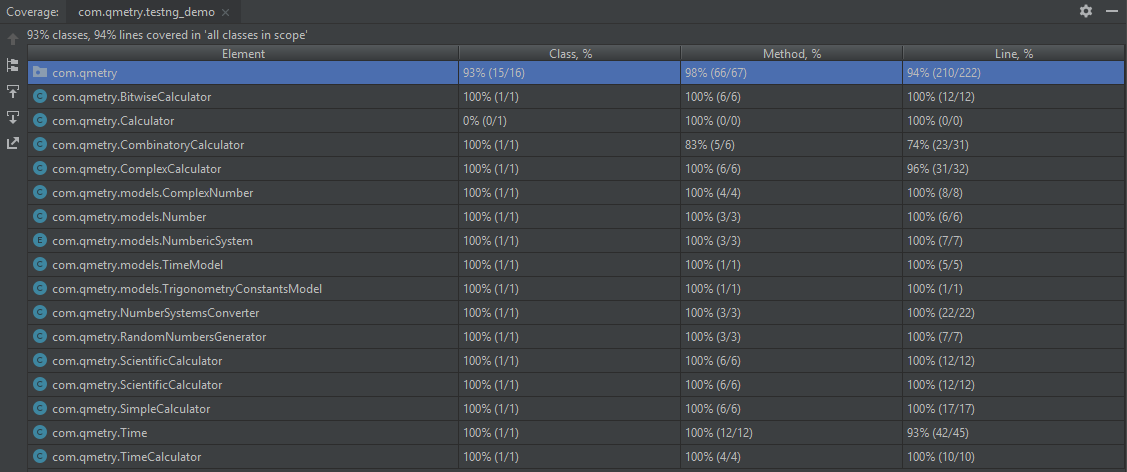


*Рис. 4. Розробка data provider для тестового випадку перевірки функції, яка порівнює комплексні числа*

* + - 1. Забезпечити 90% покриття коду тестами.

На рис. 5 зображено покриття коду розробленими юніт-тестами. Перша лінійка відображає увесь проект. Наступні лінійки –це репорт окремо по кожному класу (контролер або модель). Отже загальне покриття коду тестами:

* По рядках – 94% (210/222)
* По класах – 98% (66/67)
* По методах - 93% (15/16)



*Рис. 5. Звіт про покриття коду юніт-тестами*

1. Додати Maven Surefire Plugin для запуску тестів та налаштувати файл testing.xml для запуску тестів за допомогою maven стрічки.

Плагіни у maven проекті додаються у тому ж таки POM.xml файлі, в якому додавалися dependency для бібліотек. Додаємо Surefire плагін (із назвою файлу для налаштування автоматичного запуску тестів) та потрібну версію Surefire компілятора:

<plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.18.1</version>  
 <configuration>  
 <suiteXmlFiles>  
 <suiteXmlFile>testng.xml</suiteXmlFile>  
 </suiteXmlFiles>  
 </configuration>  
</plugin>

<plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <source>1.7</source>  
 <target>1.7</target>  
 </configuration>  
</plugin>

У файлі testng.xml задаємо налаштування для автоматичного запуску тестів за допомогою maven стрічки. У даному випадку було вказано назву testSuit, testName та перечислено тестові класи, які необхідно виконати:

<suite name="TestSuite">  
 <test name="AutoQA">  
 <classes>  
 <class name="com.qmetry.testng\_demo.BitwiseCalculatorTest" />  
 <class name="com.qmetry.testng\_demo.CombinatoryCalculatorTest" />  
 <class name="com.qmetry.testng\_demo.ComplexCalculatorTest" />  
 <class name="com.qmetry.testng\_demo.NumberSystemsConverterTest" />  
 <class name="com.qmetry.testng\_demo.RandomNumberGeneratorTest" />  
 <class name="com.qmetry.testng\_demo.ScientificCalculatorTest" />  
 <class name="com.qmetry.testng\_demo.SimpleCalculatorTest" />  
 <class name="com.qmetry.testng\_demo.TimeTest" />  
 <class name="com.qmetry.testng\_demo.TimeCalculatorTest" />  
 </classes>  
 </test>  
</suite>

Тепер, в поточному проекті відкриваємо термінал і в ньому вводимо команду mvn test для запуску на виконання вказаних тестових класів (рис. 6)

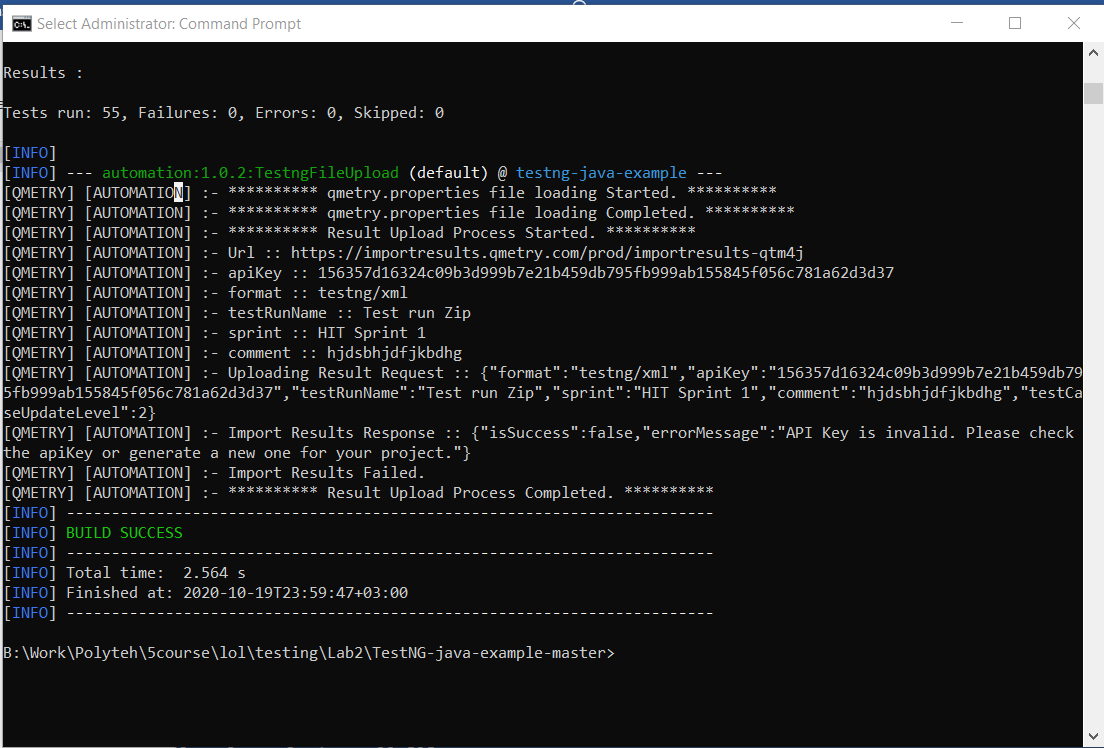


Рис. 6. Виконання тестів за допомогою команди mvn test

**ВИСНОВОК**

В результаті виконання даної роботи було розроблено тестовий проект на мові java з використанням інструменту управління проектами maven, бібліотеки для написання юніт-тестів TestNg, бібліотеки логування – Log4J та плагіну для автоматизації виконання тестів – Maven Surefire Plugin. Розроблено юніт-тести з покриттям вище 90% та налаштовано автоматичний запуск даних тестів з допомогою maven стрічки