

Лабораторная работа № 4

по дисциплине

«Математичні методи та технології тестування та верифікації програмного забезпечення»

Тема: Изучения модульного тестирования (unit testing) на примере фреймверка JUnit.

Цель работы: Цель – изучение фреймверка для модульного тестирования JUnit, сборщиков проектов, написание автоматизированных тестов для программного обеспечения.

Содержание отчета

1. Название и цель работы.
2. Выполнение индивидуального задания (задание в конце работы).
3. Выводы.
4. Ответы на контрольные вопросы.

1. Уровни тестирования [2]:

- компонентное тестирование (модульное), сфокусированное на тестировании индивидуальных компонентов (unit testing);

Занимается поиском дефектов и верификацией функционирования программных модулей, программ, объектов, классов [1], которые можно протестировать изолированно. Это может быть сделано изолированно от остальной части системы. Компонентное тестирование может включать как тестирование функциональности и специфичных нефункциональных характеристик, таких как поведение ресурсов (например, поиск утечки памяти) или тестирование надежности, так и структурное тестирование (например, покрытие кода). На практике компонентное тестирование обычно производится разработчиками, которые пишут код. Дефекты обычно исправляются сразу же без занесения их в базу дефектов.

Один из подходов к компонентному тестированию – составить автоматизированные тестовые сценарии до кодирования. Это называется разработкой, управляемой тестированием. Тесты будут выполняться до тех пор, пока не будет получен положительный результат [1].

- интеграционное тестирование выполняется, чтобы находить дефекты интерфейса и ошибки между взаимодействующими компонентами или системами (integration testing);

Интеграционное тестирование проверяет [1] интерфейсы между компонентами, взаимодействие различных частей системы, таких как операционные системы, файловая система, аппаратное обеспечение, интерфейсы между системами.

Уровни интеграционного тестирования:

- компонентное интеграционное тестирование проверяет взаимодействие между программными компонентами и производится после компонентного тестирования.

- системное интеграционное тестирование проверяет взаимодействие между системами или между аппаратным обеспечением и может быть выполнено после системного тестирования. Могут быть важны кроссплатформенные различия.

Для того, чтобы упростить процесс изоляции отказа и как можно раньше обнаружить дефекты, интеграция должна проводиться по возрастающей, а не происходить по сценарию «большого взрыва»

На каждой стадии интеграции тестировщики концентрируют все внимание именно на интеграции как таковой. Например, если интегрируется модуль А с модулем В, они проверяют взаимодействие модулей, а не функциональность каждого из них, так как она должна быть проверена во время компонентного тестирования.

В идеале, тестировщики должны понимать архитектуру и ее влияние на интеграционное тестирование.

- системное тестирование (system testing) проверяет взаимосвязанные системы для верификации указанных требований, выполняется инженерами по контролю качества;

Системное тестирование сконцентрировано на поведении тестового объекта как целостной системы или продукта. Область тестирования должна быть четко определена в главном плане тестирования либо плане тестирования для конкретного уровня тестирования.

Во время системного тестирования тестовое окружение должно быть как можно ближе к предполагаемому эксплуатационному окружению системы для минимизации риска пропуска отказов.

Системное тестирование должно может включать тесты, основанные на рисках или спецификациях требований, бизнес процессах, сценариях использования системы, моделях поведения системы, взаимодействия с операционной системой и системными ресурсами.

Системное тестирование занимается исследованием функциональных и нефункциональных требований к системе

- приемочное тестирование (acceptance testing) верифицирует соблюдение с требованиями, бизнес процессами и пользовательскими нуждами.

Системное интеграционное тестирование (system integration testing) – тестирование интеграции систем и пакетов программ, тестирование интерфейсов связи с внешними системами.

Системное тестирование – процесс тестирования системы в целом с целью проверки, что она соответствует установленным требованиям.

Стороннее приемочное тестирование – приемочное тестирование пользователями или заказчиком на своей стороне.

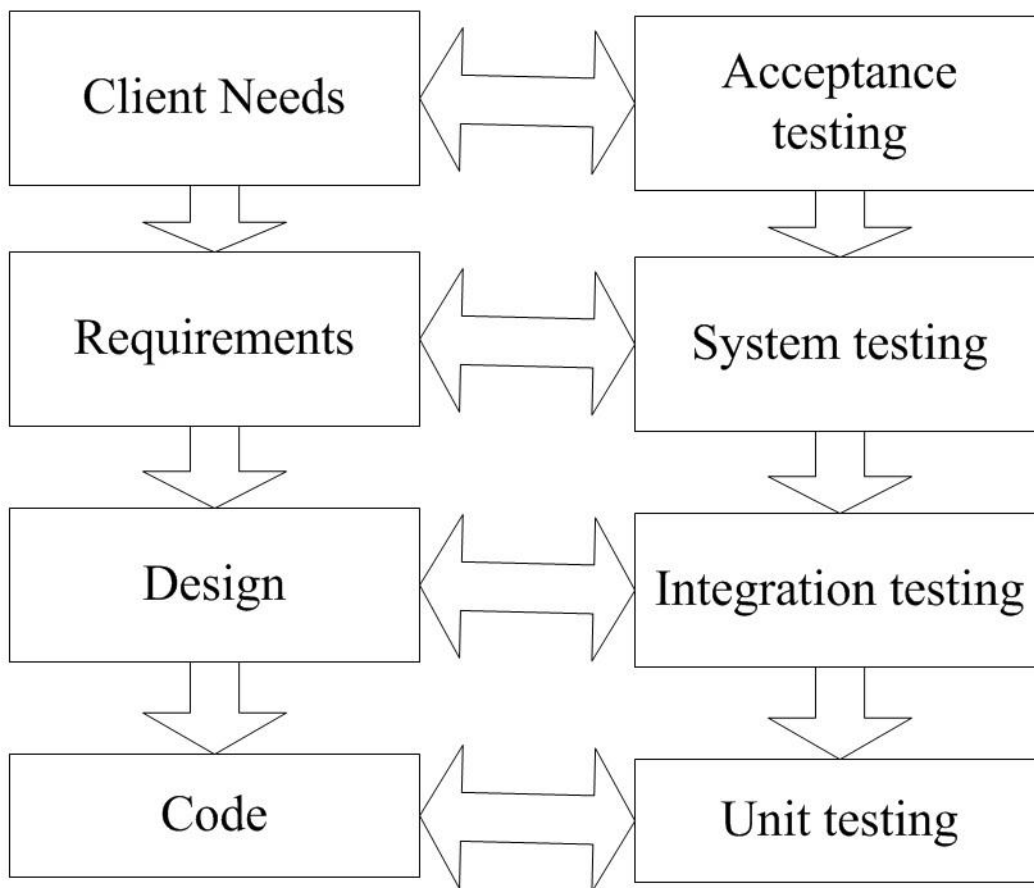


Рисунок 1 – Уровни тестирования

2. Установка студии для Java IntelliJ IDEA.

Ссылка для скачивания IntelliJ IDEA:

<https://www.jetbrains.com/idea/download/#section=windows>

Ссылка для скачивания jdk для IntelliJ IDEA.

<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

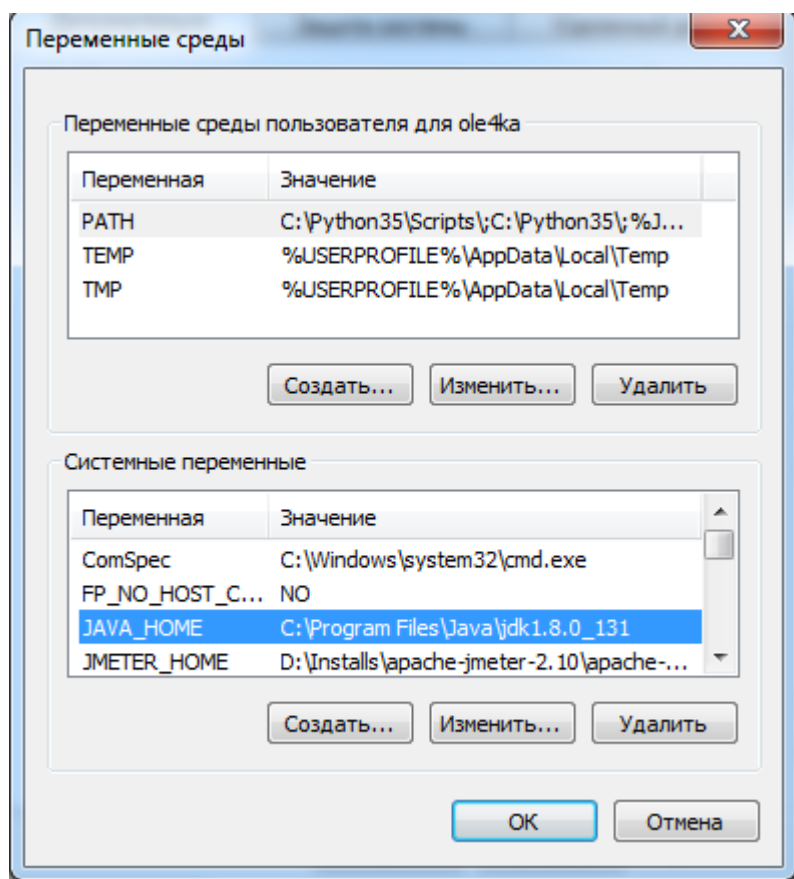


Рисунок 2 – Для установки jdk

<https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html>

3. Сборщики проектов и подключение зависимостей JUnit.

Maven - фреймверк для автоматизации и сборки проектов на основе описания их структуры в файлах на языке POM (Project Object Model), который является подмножеством XML.

Gradle – система автоматической сборки, построенная на принципах Apache Ant и Apache Ant, но предоставляющая DSL на языке Groovy вместо традиционной XML образной формы представления конфигурации проекта.

Подключать различные библиотеки можно к обычному проекту Java, рисунок 2.

JUnit – фреймверк для написания повторяющихся модульных тестов (требуется более подробного описания, но в работе пока ограничимся только подключением и написанием пары однотипных тестов).

JUnit – можно скачать по ссылке:

<https://jar-download.com/artifacts/junit/junit/4.12/source-code>

Добавить библиотеку JUnit

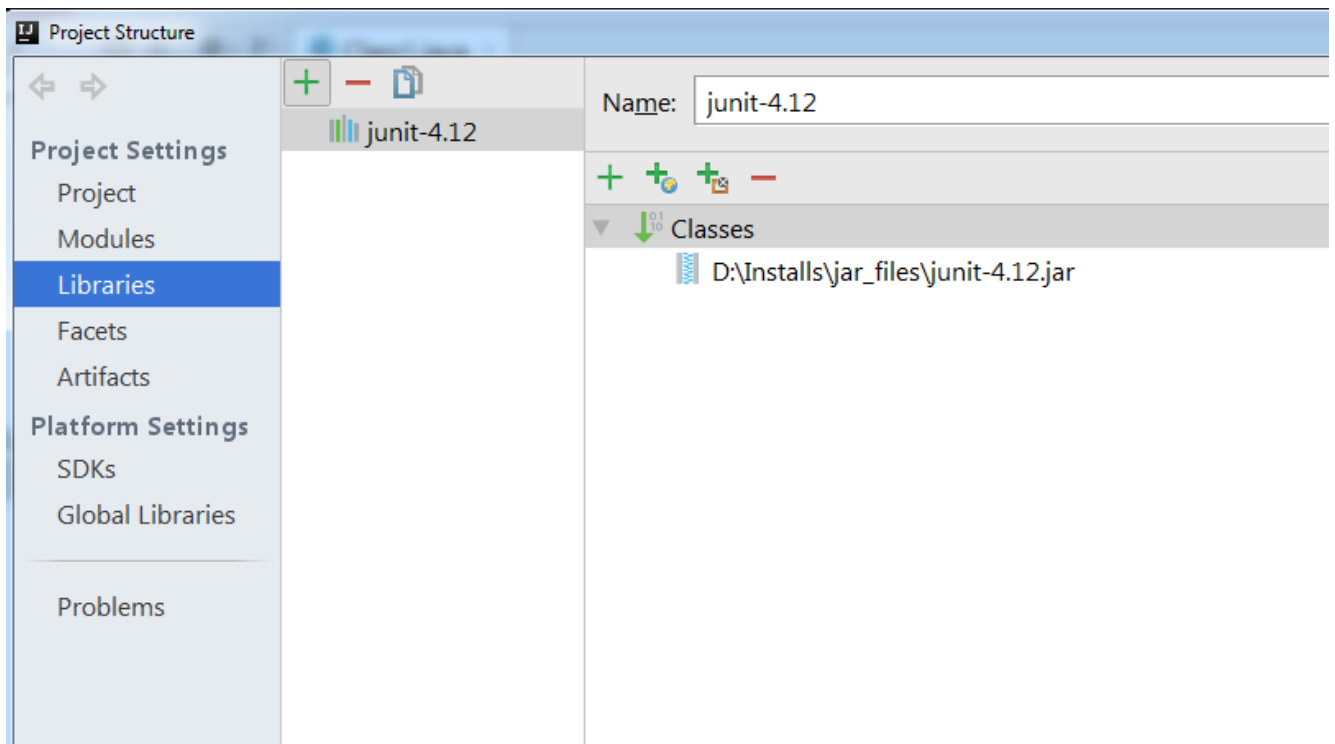


Рисунок 3

Зависимости – это те библиотеки, которые непосредственно используются в вашем проекте для компиляции кода или его тестирования. На сайте <https://mvnrepository.com/artifact/junit/junit/4.12> можно найти зависимости, к примеру для JUnit (и не только).

Создаем проект Maven.

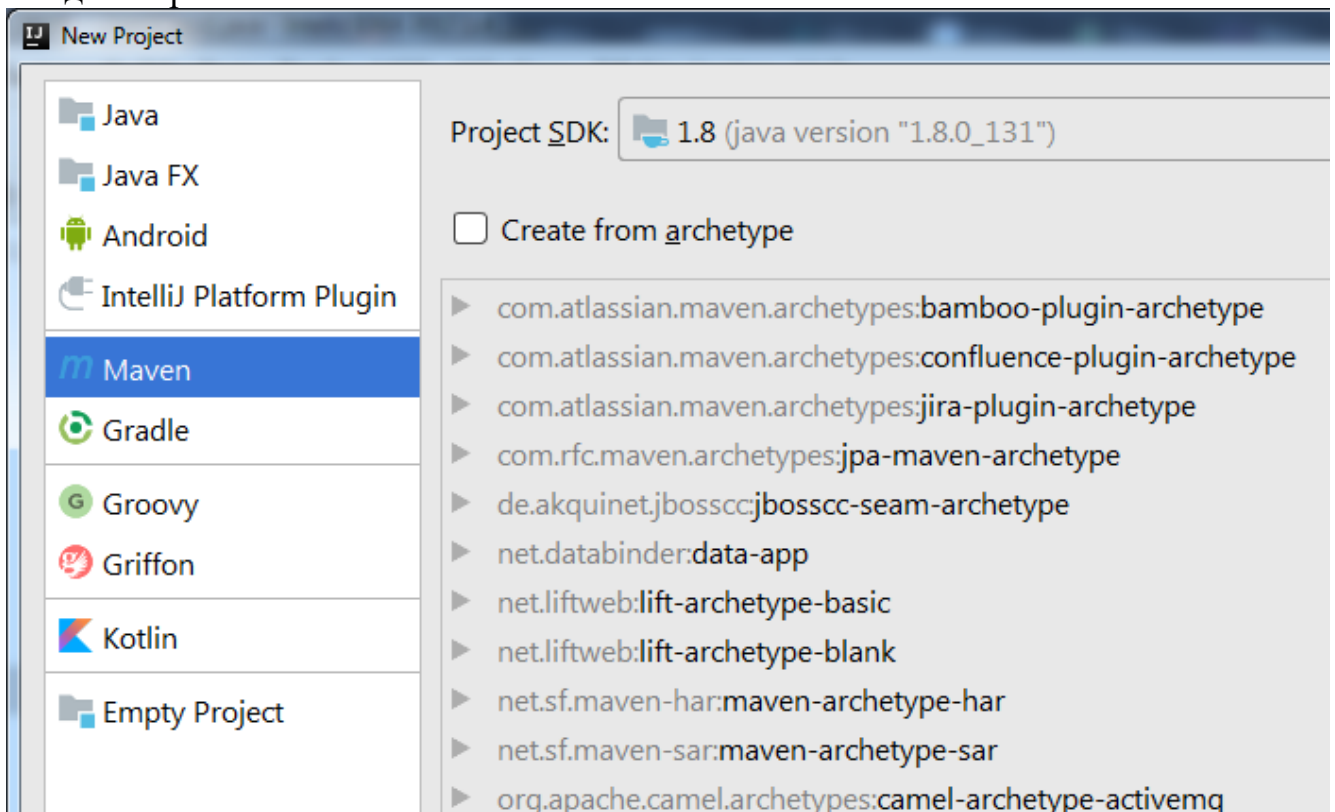


Рисунок 4

Записываем название проекта.

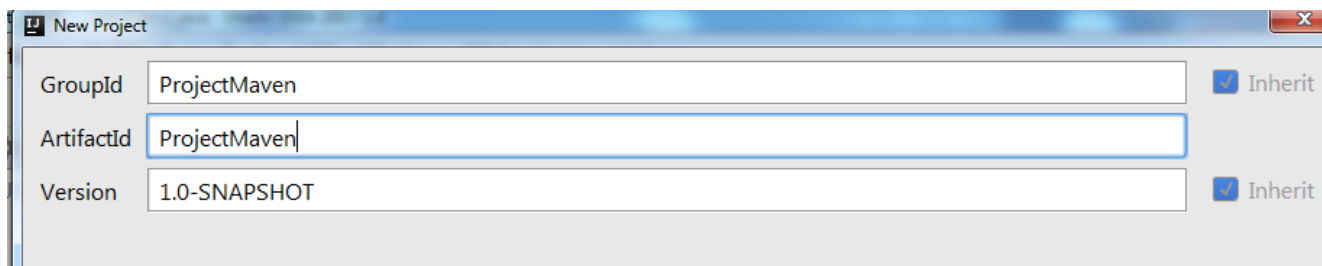


Рисунок 5

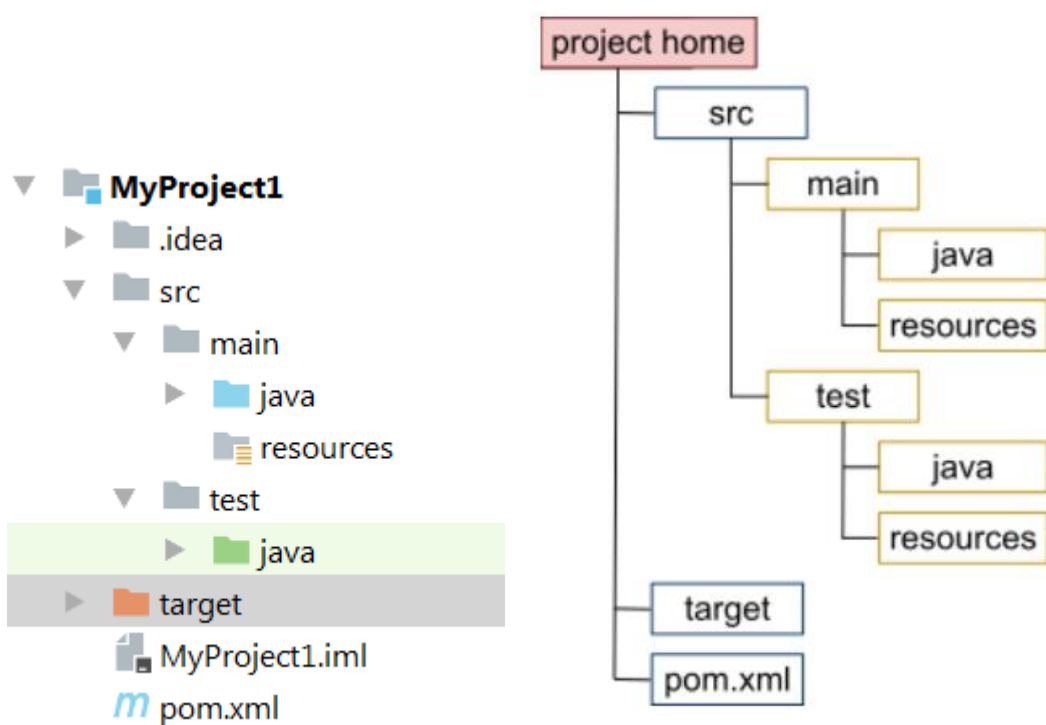


Рисунок 6 - Структура для Java проекта

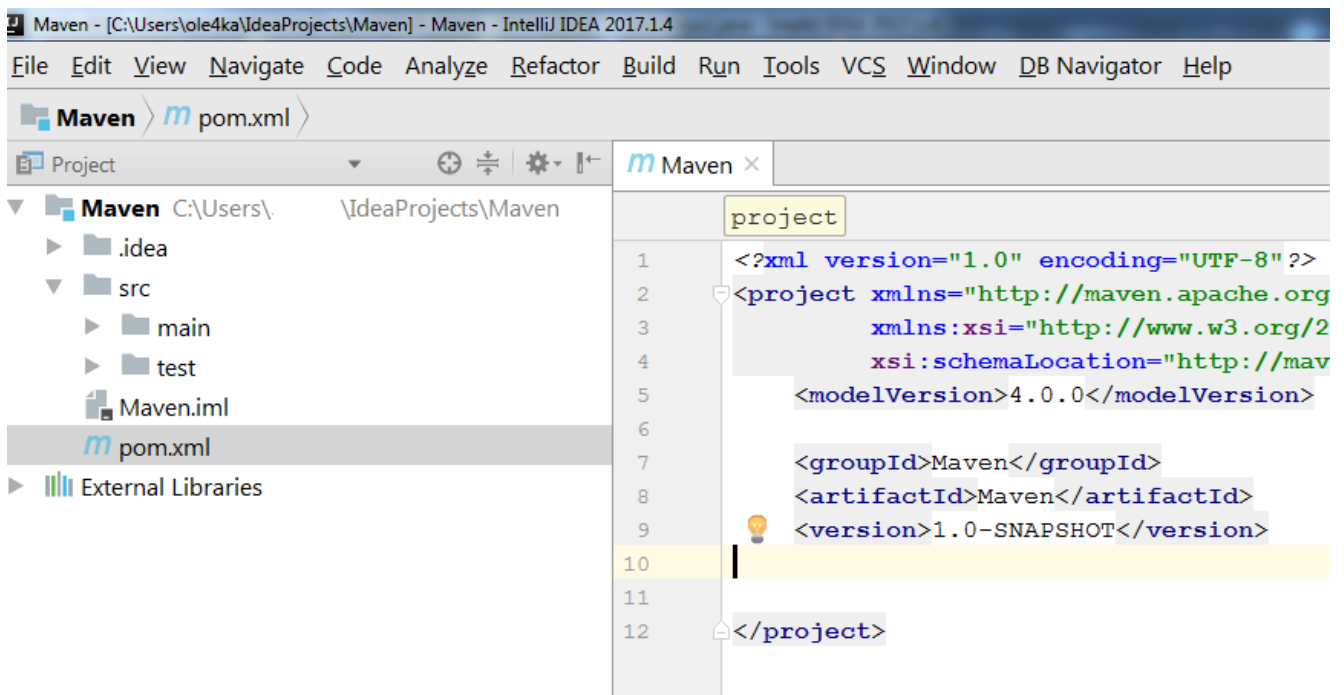


Рисунок 7
В pom.xml копируем зависимости (рисунок 7):

```

<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
</dependency>

```

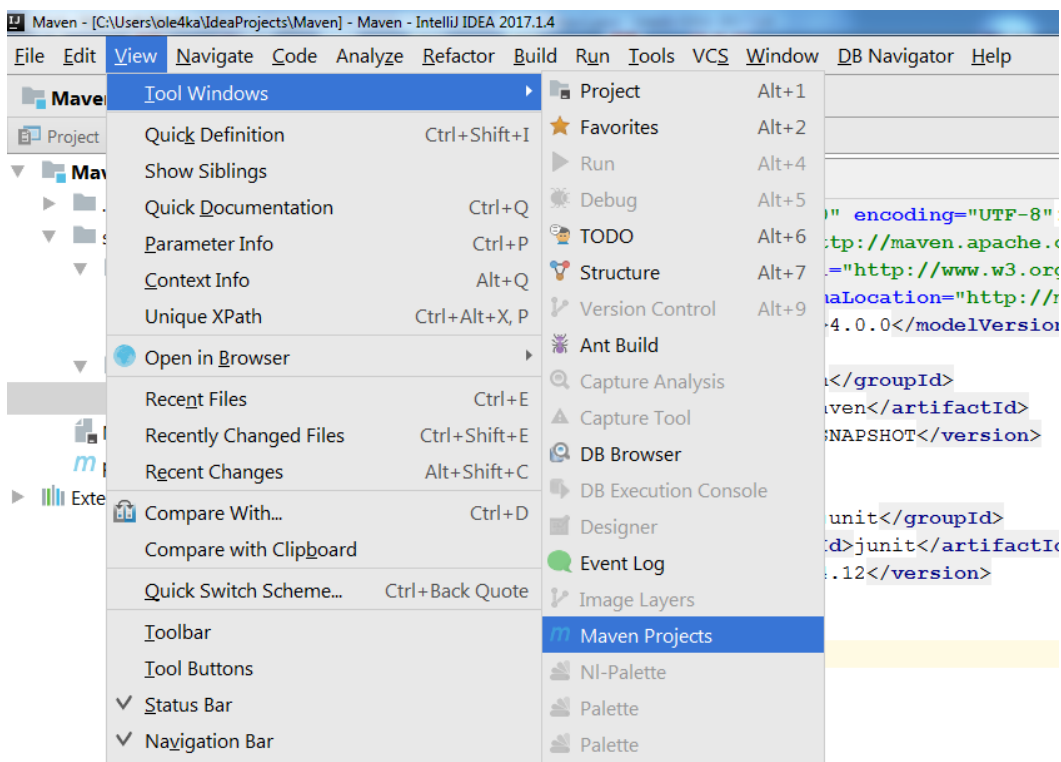



Рисунок 8

Справа откроется панель, необходимо нажать на клавишу , тогда зависимость подтянется (рисунок 9-10).

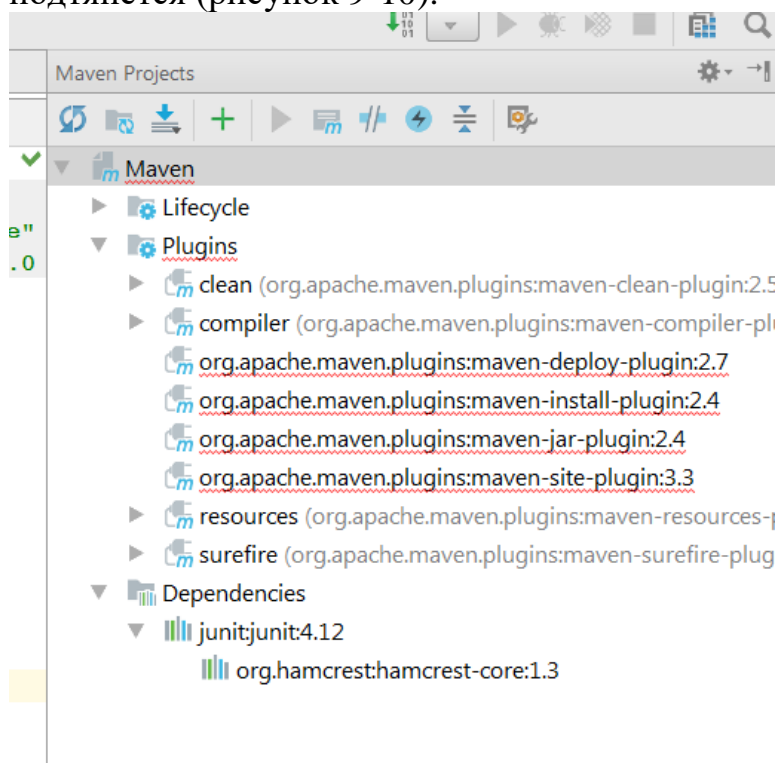


Рисунок 9

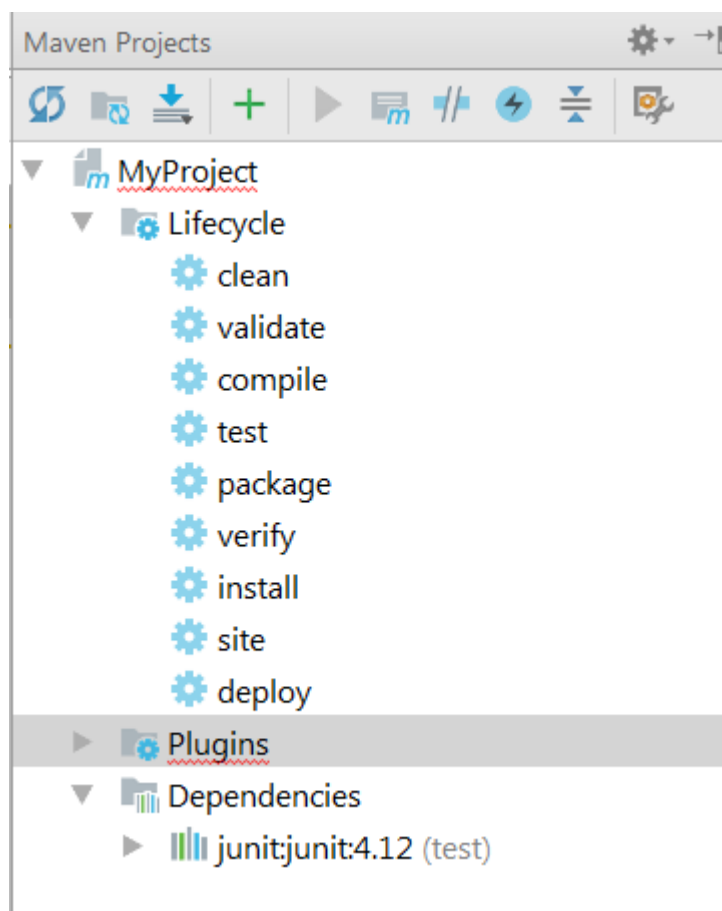


Рисунок 10

Цикл по умолчанию, цикл сборки состоит из шагов, из которых явно используется только 7 фаз сборки (рисунок 10):

validate — в этой фазе проверяется корректность проекта и обеспечивается доступность необходимых зависимостей;

compile — Компилируется исходный код проекта;

test — Код тестов компилируется и запускаются unit тесты;

package — Скомпилированный код собирается в пакет (jar/war/ear/etc);

verify — Запускаются интеграционные тесты;

install — Собранный ранее пакет устанавливается в локальный репозиторий и становится доступен для сборки других локальных проектов;

deploy — Пакет публикуется в удалённых репозиториях, устанавливается на серверы приложений и так далее.

Как пример для калькулятора можно использовать код, приведенный ниже.

```
public class Calculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Calculator calculator = new Calculator();
        System.out.println("Input number A");
        int A = scanner.nextInt();
        System.out.println("Input number B");
        int B = scanner.nextInt();
        System.out.println("Choose operation +, -, *, /");
        String operation = new String();
        operation = scanner.next();
        switch (operation) {
            case "+":
                System.out.println(calculator.Sum(A, B));
                break;
            case "-":
                System.out.println(calculator.Substraction(A, B));
                break;
            case "*":
                calculator.Multiplication(A, B);
                break;
            case "/":
                calculator.Division(A, B);
                break;
        }
    }

    public int Sum(int A, int B) {
        System.out.println("Result add " + (A + B));
        return (A + B);
    }

    public double Division(int A, int B) {
        System.out.println("Result division " + (double) A / B);
        return A / B;
    }
}
```

4. Пример теста с использованием JUnit, аннотации JUnit, описание некоторых перегруженных методов JUnit.

Для пояснения, как работают методы с аннотациями `@BeforeClass`, `@Before`, `@After` и `@AfterClass` приведен код и результат выполнения кода приведен ниже:

```
@BeforeClass
public static void bef() throws Exception {
    System.out.println("метод BeforeClass");
}

@Before
public void setUp() throws Exception {
    f = new method();
    System.out.println("метод Before");
}

@Test
public void testMaven1() throws Exception {
    System.out.println("testMaven1");
    Assert.assertTrue("my test", true);
}

@Test
public void testMaven2() throws Exception {
    System.out.println("testMaven2");
    Assert.assertEquals(133, 133);
}

@Test
public void testMaven3() throws Exception {
    System.out.println("testMaven3");
    Assert.assertEquals(11, 11);
}

@Test
public void testMaven4() throws Exception {
    System.out.println("testMaven4");
    Assert.assertEquals(11, 11);
}

@Test
public void testMaven5() throws Exception {
    System.out.println("testMaven5");
    Assert.assertEquals(12, 12);
}

@Test
public void testMaven6() throws Exception {
    System.out.println("testMaven6");
    Assert.assertEquals(11, 11);
}

@Test
public void summa() throws Exception {
    //System.out.println("summa");
    //method f=new method();
    Assert.assertEquals(4, f.sum(2, 2));
}

@org.junit.After
public void tearDown() throws Exception {
    System.out.println("after");
}

@AfterClass
public static void afterClass() throws Exception {
```

```

    System.out.println("метод AfterClass");
}

```

```

"C:\Program Files\Java\jdk1.8.0_131\bin\java" ...
метод BeforeClass
метод Before
testMaven
after
метод Before
after
метод Before
testMaven2
after
метод Before
testMaven3
after
метод Before
testMaven4
after
метод Before
testMaven5
after
метод Before
testMaven6
after
метод AfterClass
Process finished with exit code 0

```

JUnit обеспечивает перегруженные методы для всех примитивных типов, объектов и массивов (некоторые из методов записаны в таблице 1). Параметры: ожидаемое значение и актуальное значение. Опционально первым параметром может быть сообщение, которое появляется, если тест упал.

Например, приведенная ниже строка проверяет, что два примитива равны.

```
Assert.assertEquals(expectedResult, result, 0);
```

```

import org.junit.Assert;
import org.junit.Test;

public class CalculatorTest{
    Calculator calculator=new Calculator();

    @Test
    public void testDiv(){
        double result=calculator.Division(2,2);
        double expectedResult=1;
        Assert.assertEquals(expectedResult,result,0);
    }
}

```

Рисунок 11 – Пример автоматизированного теста

Таблица 1 – Некоторые из методов JUnit.

Метод	Описание метода
assertArrayEquals(boolean [] expecteds, boolean []actuals)	Сравнивает, что два массива типа boolean равны, с положительной дельтой.
assertFalse(boolean condition)	Проверяет, что условие ложно
assertNotEquals(double unexpected, double actual, double delta)	Проверяет, что два числа double не равны, с положительной дельтой.
assertEquals(String message, double expected, double actual, double delta)	Проверяет, что два числа double равны, с положительной дельтой. Первый параметр – сообщение.

assertNotNull(Object object)	Проверяет, что объект не null.
assertNull(Object object)	Проверяет, что объект null.
assertTrue(String message, boolean condition)	Проверяет, что условие не ложно.

Задание

1. Установить IntelliJ IDEA (если установлено, то пропустить).
2. Создать проект Java, подключить JUnit к проекту.
3. Создать проект при помощи сборщика проектов Maven, подключить JUnit.
4. Запустить пример калькулятора, который описан в работе (см. описание).
5. Написать несколько методов в среде IntelliJ IDEA (ограничений по использованию чего-либо нет, используйте все, что знаете). Вариант в таблице 2. Поработать с требованиями к программе самостоятельно, в связи с этим подкорректировать задание к таблице 2 (**в отчет написать уточненное задание**). Например, в описании нет ни слова о том, какие должны быть числа в массивах.
6. Написать 4-5 тестов с использованием JUnit для методов, написанных в 4 пункте.

Таблица 2 – Задания для выполнения пункта 5.

Вариант	
1	Сортировка массива по убыванию, размер массива любой, ввод массива с консоли.
2	Сортировка массива по возрастанию, размер массива любой. Ввод массива с консоли.
3	Поиск минимального числа в массиве, размер массива любой. Ввод массива с консоли.
4	Поиск максимального числа в массиве, размер массива любой. Ввод массива с консоли.
5	Метод, позволяющий перемножить две матрицы, размер матриц любой.
6	Метод, удаляющий повторяющиеся элементы из массива. [5,5,3,3,8,8,9,9,10] → [5,3,8,9,10].
7	Метод, меняющий элементы массива местами. Например, [5,6,7,8] → [8,7,6,5].
8	Метод, удаляющий повторяющиеся элементы из массива. [5,5,3,3,8,8,9,9,10] → [10].
9	Метод, который возвращает максимальное число верхней треугольной матрицы, размер матрицы любой.
10	Метод, который возвращает минимальное число нижней треугольной матрицы, размер матрицы любой.
11	Сортировка массива по убыванию, размер массива любой, ввод массива с консоли.
12	Сортировка массива по возрастанию, размер массива любой. Ввод

	массива с консоли.
13	Поиск минимального числа в массиве, размер массива любой. Ввод массива с консоли.
14	Поиск максимального числа в массиве, размер массива любой. Ввод массива с консоли.
15	Метод, позволяющий перемножить две матрицы, размер матриц любой.
16	Метод, удаляющий повторяющиеся элементы из массива. [5,5,3,3,8,8,9,9,10] → [5,3,8,9,10].
17	Метод, меняющий элементы массива местами. Например, [5,6,7,8] → [8,7,6,5].
18	Метод, удаляющий повторяющиеся элементы из массива. [5,5,3,3,8,8,9,9,10] → [10].
19	Метод, который возвращает максимальное число верхней треугольной матрицы, размер матрицы любой.
20	Метод, который возвращает минимальное число нижней треугольной матрицы, размер матрицы любой.
21	Сортировка массива по возрастанию, размер массива любой. Ввод массива с консоли.
22	Поиск минимального числа в массиве, размер массива любой. Ввод массива с консоли.
23	Поиск максимального числа в массиве, размер массива любой. Ввод массива с консоли.
24	Метод, позволяющий перемножить две матрицы, размер матриц любой.
25	Метод, удаляющий повторяющиеся элементы из массива. [5,5,3,3,8,8,9,9,10] → [5,3,8,9,10].
26	Поиск максимального числа в массиве, размер массива любой. Ввод массива с консоли.