Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет инженерно-экономический

Кафедра экономической информатики

# **ОТЧЁТ**

по лабораторной работе №5

**Разработка эффективного кода**

**Функциональное тестирование приложений Java**

|  |  |  |
| --- | --- | --- |
| Выполнила: |  | студентка гр. 074004  Лопарева А. В. |
| Проверила: |  | Лукашевич А. Э. |

Минск 2022

**Задание:** частично реализовать методы классов из UML диаграмм информационной системы.

Разработать Unit-тесты для паттернов поведения, структурных паттернов и порождающих паттернов.

**Результат выполнения лабораторной работы:**

Рассмотрим этапы создания Unit-теста.

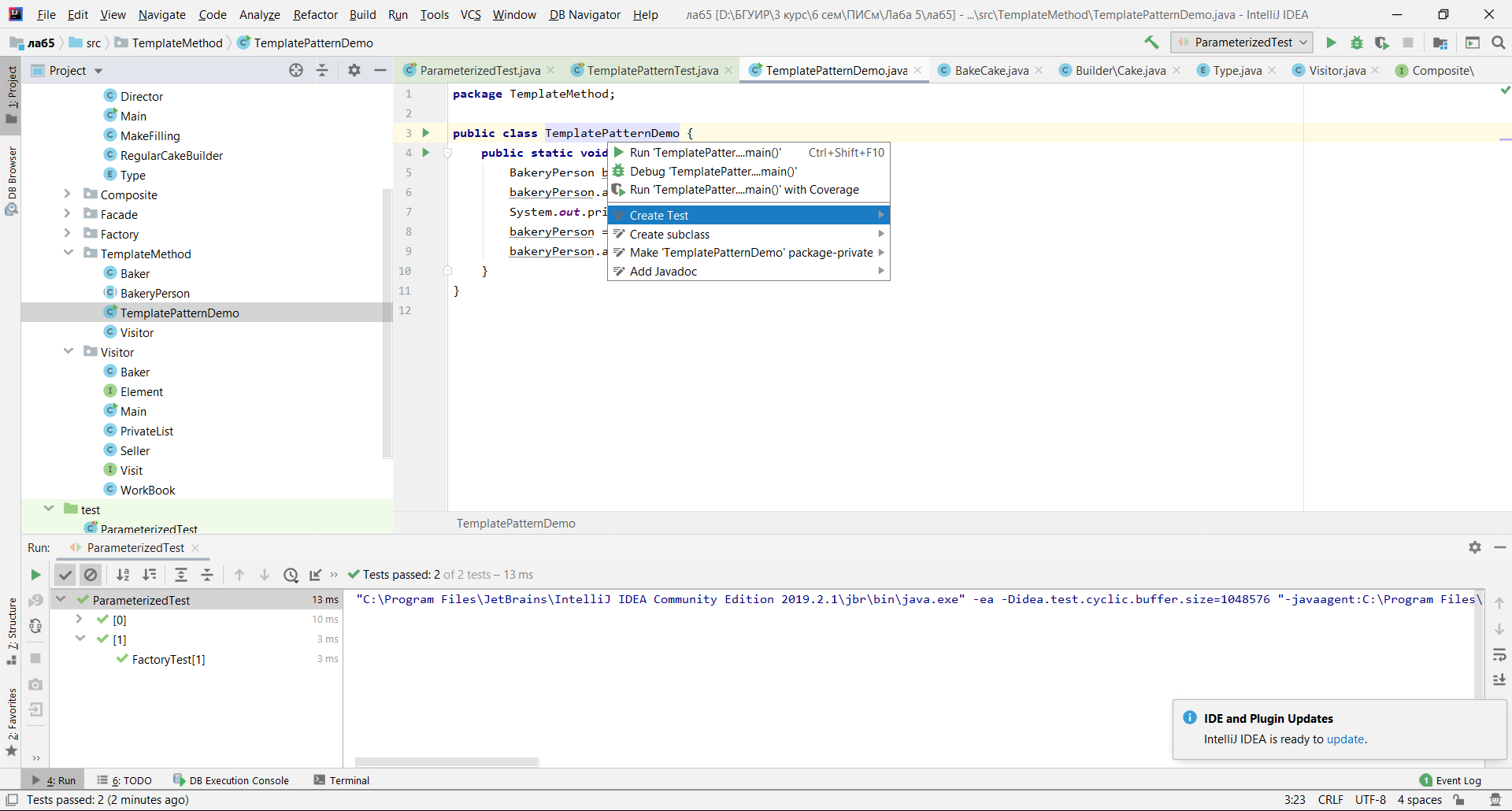


Рисунок 1 – Создание Unit-теста

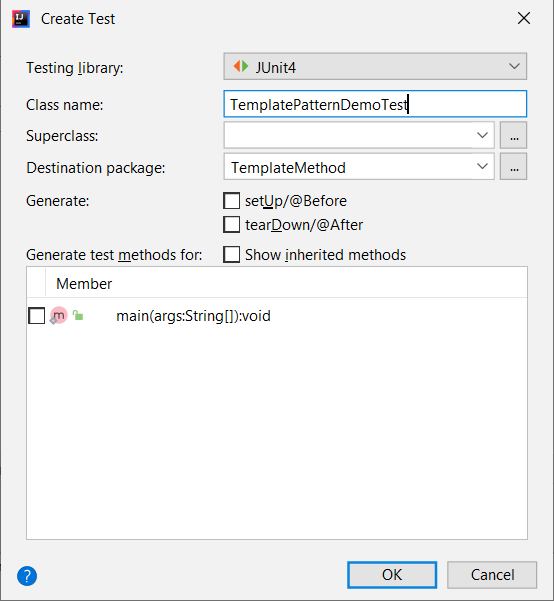


Рисунок 2 – Выбор необходимых параметров

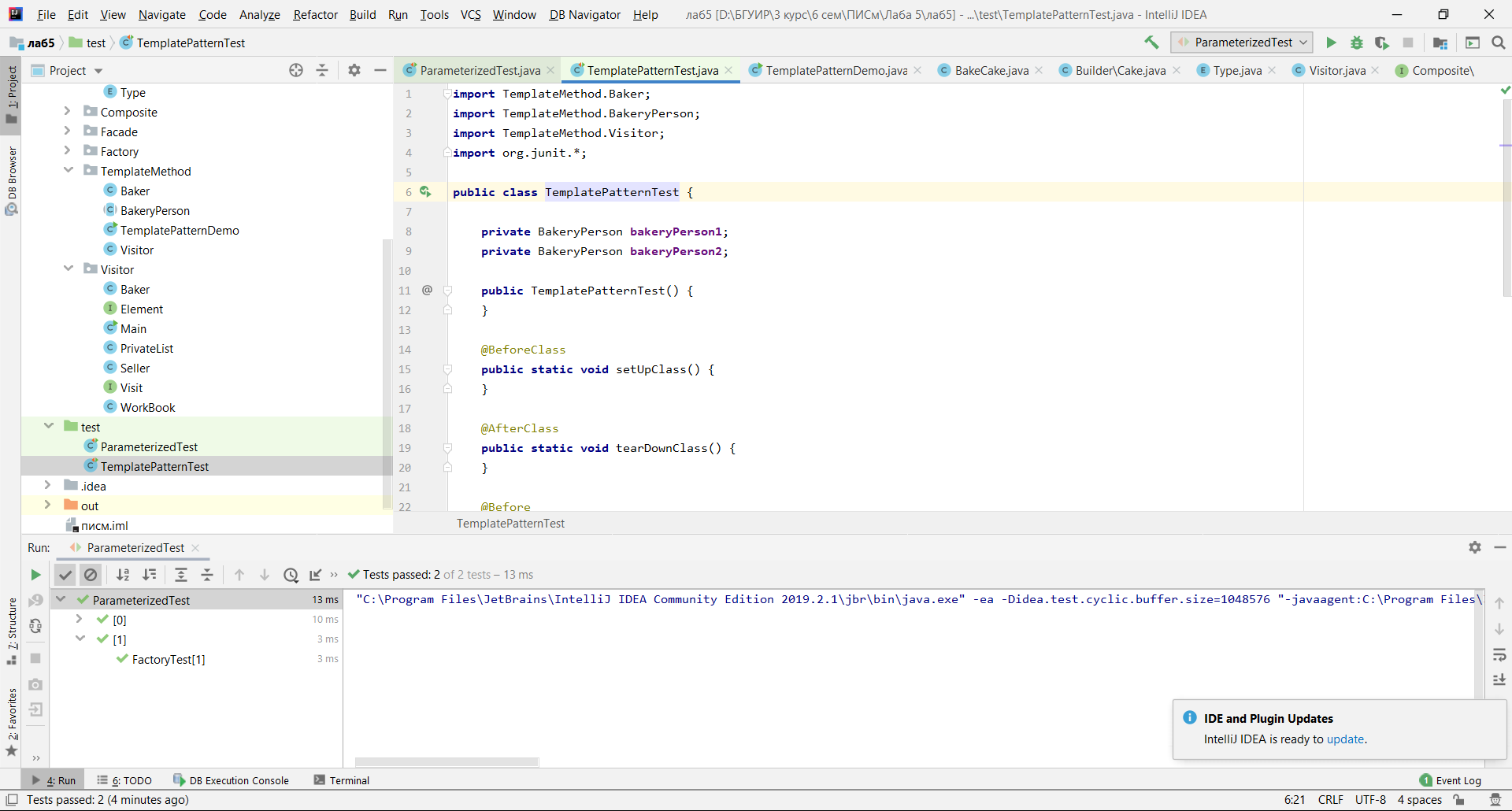


Рисунок 3 – Код программы

Код:

**import** TemplateMethod.Baker;  
**import** TemplateMethod.BakeryPerson;  
**import** TemplateMethod.Visitor;  
**import** org.junit.\*;  
  
**public class** TemplatePatternTest {  
  
 **private** BakeryPerson **bakeryPerson1**;  
 **private** BakeryPerson **bakeryPerson2**;  
  
 **public** TemplatePatternTest() {  
 }  
  
 @BeforeClass  
 **public static void** setUpClass() {  
 }  
  
 @AfterClass  
 **public static void** tearDownClass() {  
 }  
  
 @Before  
 **public void** setUp() {  
 **bakeryPerson1** = **new** Baker();  
 **bakeryPerson2** = **new** Visitor();  
 }  
  
 @After  
 **public void** tearDown() {  
 }  
  
 @Test  
 **public void** Test() {  
 Assert.*assertNotNull*(**bakeryPerson1**);  
 Assert.*assertNotNull*(**bakeryPerson2**);  
 System.***out***.println(**"Test passed"**);  
 }  
}

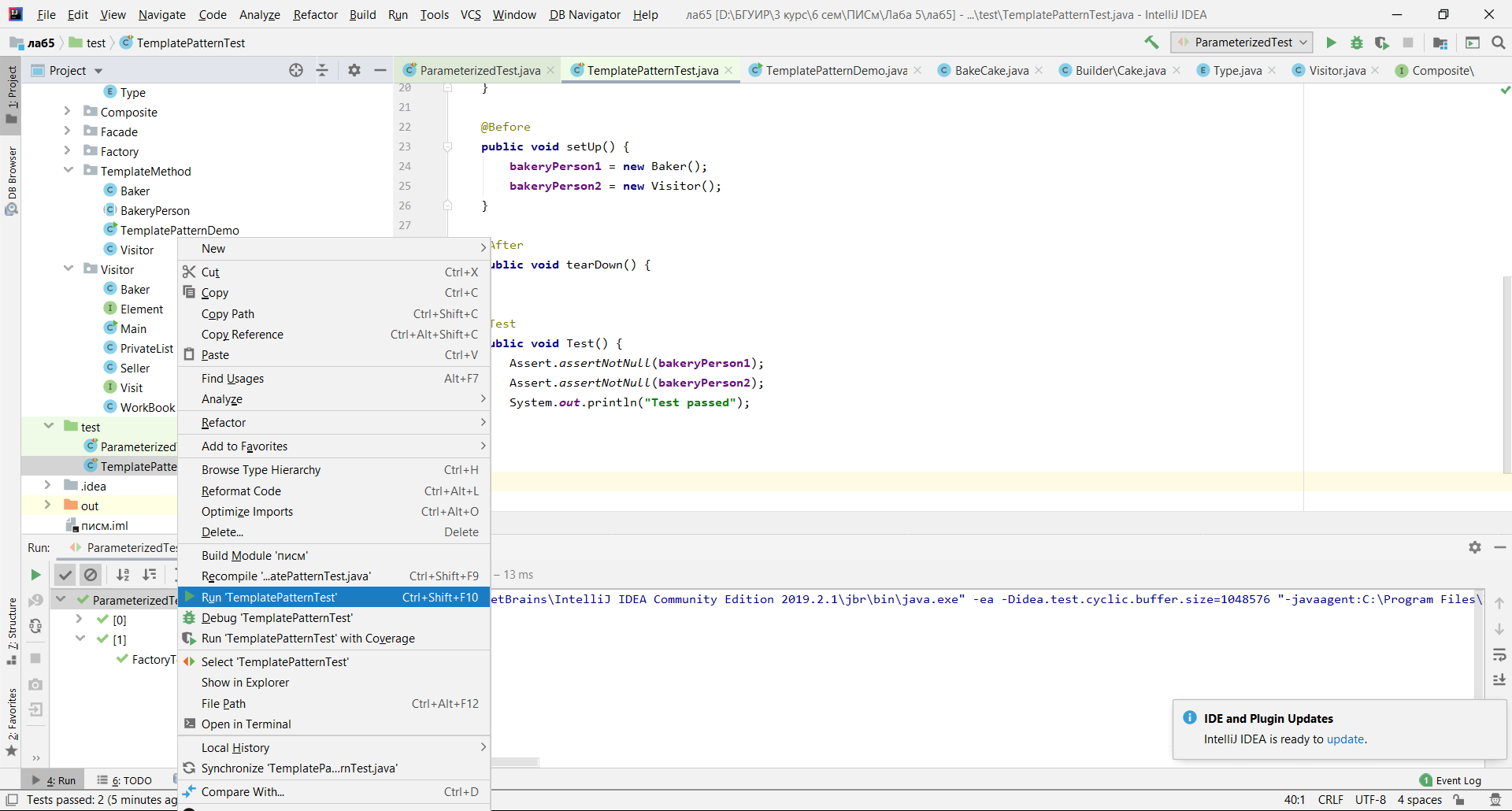


Рисунок 4 – Запуск тестирования

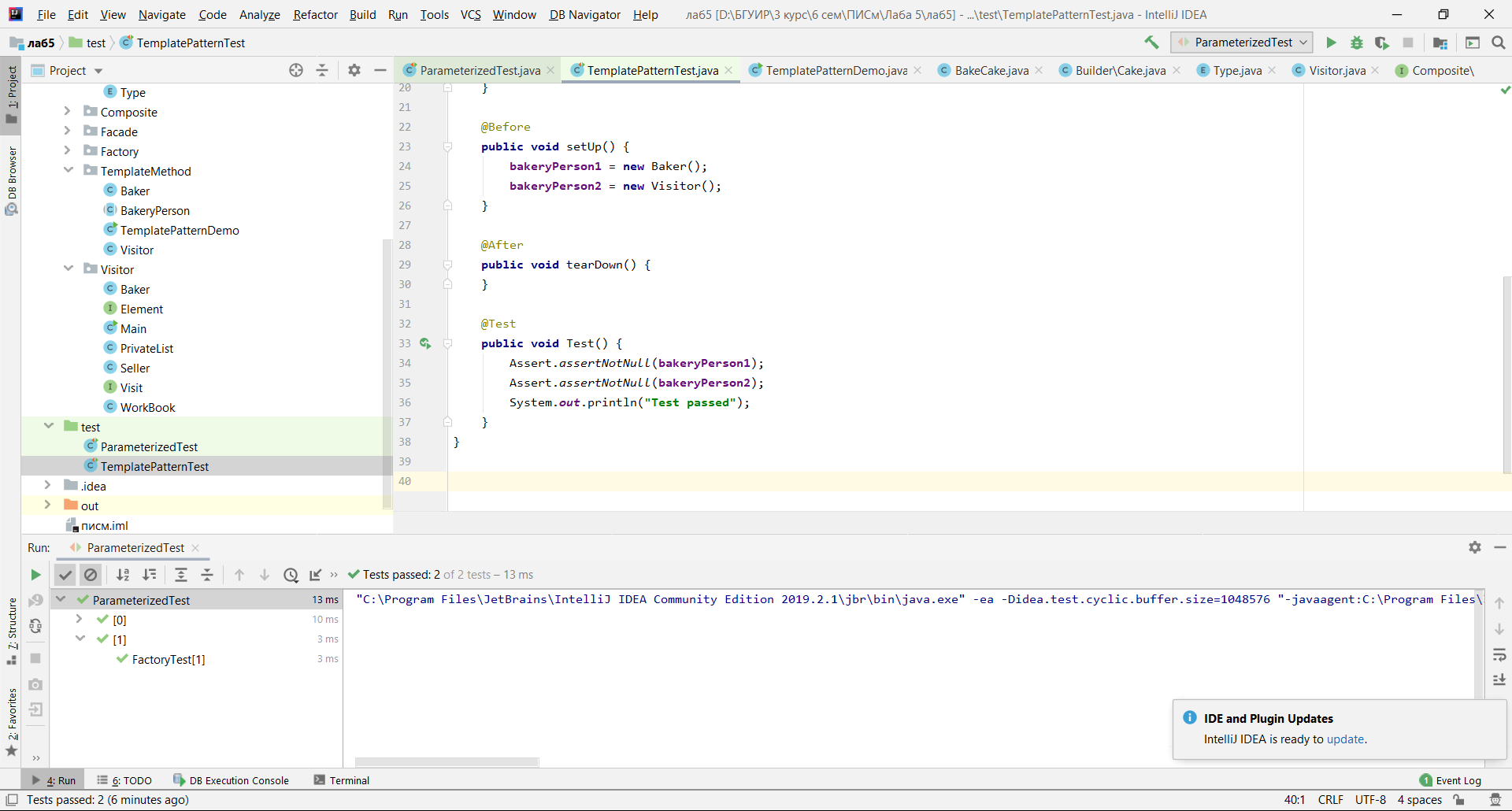


Рисунок 5 – Результат тестирования

Аналогично создаем параметризированный Unit-тест.

Код:

**import** Factory.Factory;  
**import** org.junit.Assert;  
**import** org.junit.Test;  
**import** org.junit.runner.RunWith;  
**import** org.junit.runners.Parameterized;  
**import** org.junit.runners.Parameterized.Parameters;  
  
**import** java.util.Arrays;  
**import** java.util.Collection;  
  
@RunWith(Parameterized.**class**)  
**public class** ParameterizedTest {  
  
 @Parameters  
 **public static** Collection<Object[]> data() {  
 **return** Arrays.*asList*(**new** Object[][] {  
 { **"custom"**, **"Factory.CustomCake"**}, { **"regular"**, **"Factory.RegularCak"** }  
 });  
 }  
  
 **private final** Factory **factory**;  
 **private final** String **input**;  
 **private final** String **expected**;  
  
 **public** ParameterizedTest(String input, String expected) {  
 **this**.**factory** = **new** Factory();  
 **this**.**input** = input;  
 **this**.**expected** = expected;  
 }  
  
 @Test  
 **public void** FactoryTest() {  
 Assert.*assertEquals*(**expected**, **factory**.getCurrentCake(**input**).toString().substring(0, 18));  
 }  
}

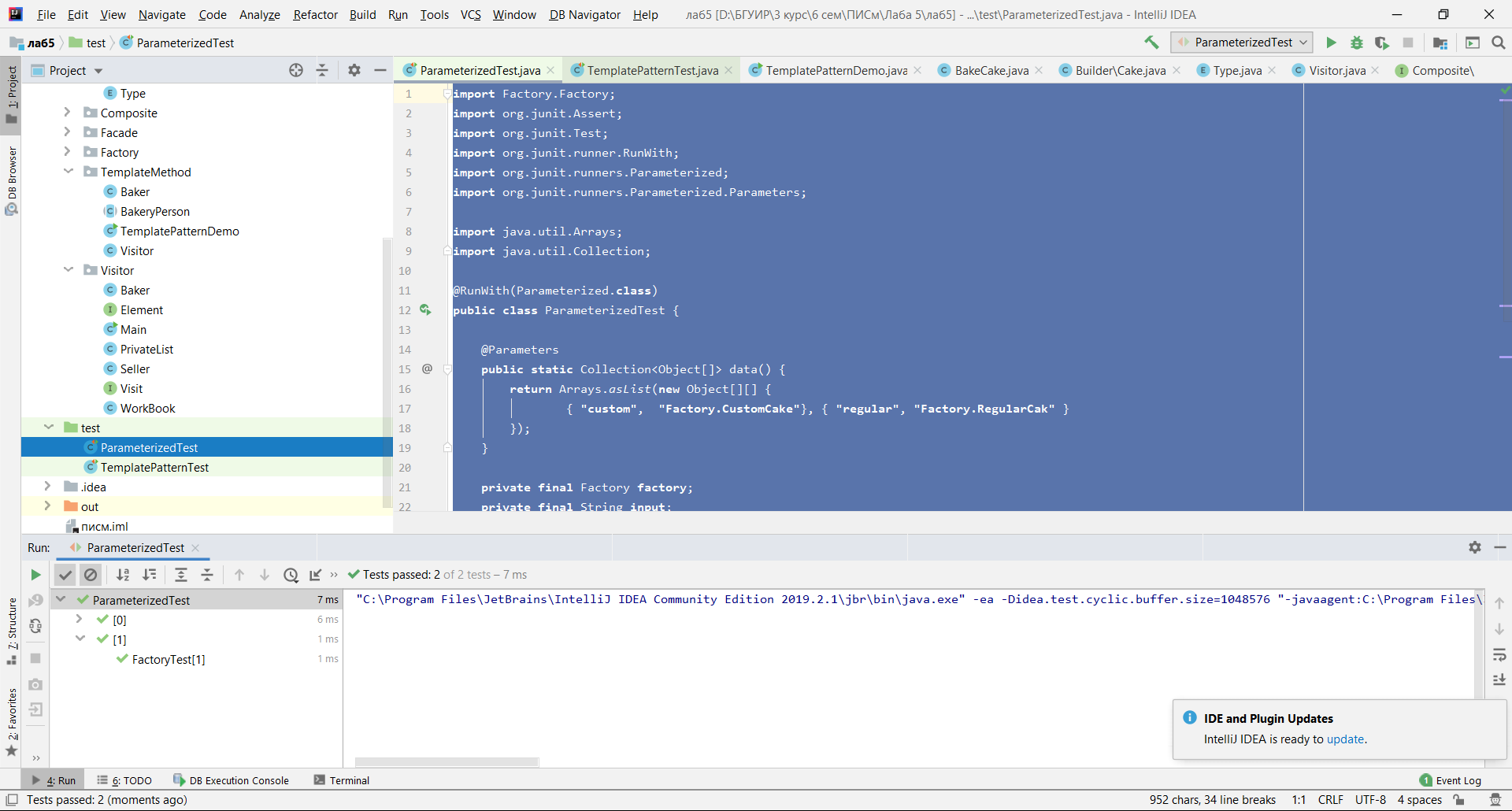


Рисунок 6 – Результат тестирования

**Выводы:** В данной лабораторной работе я частично реализовала методы классов из UML диаграмм информационной системы, разобрала Unit-тесты для паттернов поведения, структурных паттернов и порождающих паттернов.