**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное

учреждение высшего образования

**Национальный исследовательский**

**Томский политехнический университет**

Инженерная школа информационных технологий и робототехники

Отделение информационных технологий

Отчет по лабораторной работе №11 по дисциплине

**«Язык Kotlin и основы разработки»**

Отладка. Юнит-тестирование

****

Выполнил:

Студент группы 1А22 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.К. Кравцов

Проверил:

Ст. преп. ОИТ ИШИТР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.А. Дорофеев

Томск 2025

# Задание

Выберите какую-либо область, в которой можно делать расчёты: геометрия, физика, химия, экономика, …

Разработайте класс, содержащий несколько методов для каких-либо вычислений в выбранной области.

Разработайте тесты для проверки правильности работы каждого из этих методов, на обычных значениях, и на граничных значениях.

Запустите тесты, убедитесь, что всё считается правильно. Зафиксируйте это в отчёте скриншотами и описанием.

Допустите намеренную ошибку в одном или нескольких методах исходного класса.

Запустите тесты и убедитесь, что все неверные вычисления приводят к падению соответствующих тестов. Зафиксируйте это в отчёте скриншотами и описанием какие именно тесты и assert-методы вызвали ошибки и почему.

# Ход работы

1. Создан проект Lab11 на основе Empty Views Activity.
2. Выбрана область электротехники.
3. Создан класс `ElectronicsCalculator` с методами для расчётов:

package ru.olegkravtsov.lab11  
  
class ElectronicsCalculator {  
  
 /\*\*  
 \* Расчёт напряжения по закону Ома для участка электрической цепи: U = I \* R  
 \*/  
 fun calculateVoltage(current: Double, resistance: Double): Double {  
 return current \* resistance  
 }  
  
 /\*\*  
 \* Расчёт мощности постоянного тока: P = I^2 \* R  
 \*/  
 fun calculatePower(current: Double, resistance: Double): Double {  
 return current \* current \* resistance  
 }  
  
 /\*\*  
 \* Расчёт общего сопротивления последовательной активной цепи  
 \*/  
 fun calculateSeriesResistance(vararg resistances: Double): Double {  
 return resistances.sum()  
 }  
  
 /\*\*  
 \* Расчёт общего сопротивления параллельной активной цепи  
 \*/  
 fun calculateParallelResistance(vararg resistances: Double): Double {  
 if (resistances.isEmpty()) return 0.0  
 var sum = 0.0  
 for (r in resistances) {  
 if (r == 0.0) return 0.0 // Защита от деления на ноль  
 sum += 1.0 / r  
 }  
 return 1.0 / sum  
 }  
  
 /\*\*  
 \* Расчёт реактивного сопротивления дросселя: Xl = 2 \* π \* f \* L  
 \*/  
 fun calculateInductiveReactance(frequency: Double, inductance: Double): Double {  
 return 2 \* Math.PI \* frequency \* inductance  
 }  
}

1. Создан класс ElectronicsCalculatorTest в ветке (test) с комплексными тестами:

package ru.olegkravtsov.lab11  
  
import org.junit.Assert.\*  
import org.junit.Before  
import org.junit.Test  
  
class ElectronicsCalculatorTest {  
  
 private lateinit var calculator: ElectronicsCalculator  
  
 @Before  
 fun init() {  
 calculator = ElectronicsCalculator()  
 }  
  
 @Test  
 fun calculateVoltage\_isCorrect() {  
 // Обычные значения  
 assertEquals(12.0, calculator.calculateVoltage(2.0, 6.0), 0.001)  
 assertEquals(0.0, calculator.calculateVoltage(0.0, 10.0), 0.001)  
  
 // Граничные значения  
 assertEquals(1000000.0, calculator.calculateVoltage(1000.0, 1000.0), 0.001)  
 assertEquals(0.001, calculator.calculateVoltage(0.001, 1.0), 0.001)  
 }  
  
 @Test  
 fun calculatePower\_isCorrect() {  
 // Обычные значения  
 assertEquals(24.0, calculator.calculatePower(2.0, 6.0), 0.001)  
 assertEquals(0.0, calculator.calculatePower(0.0, 10.0), 0.001)  
  
 // Граничные значения  
 assertEquals(1.0E6, calculator.calculatePower(1000.0, 1.0), 0.001)  
 }  
  
 @Test  
 fun calculateSeriesResistance\_isCorrect() {  
 // Обычные значения  
 assertEquals(15.0, calculator.calculateSeriesResistance(5.0, 10.0), 0.001)  
 assertEquals(30.0, calculator.calculateSeriesResistance(5.0, 10.0, 15.0), 0.001)  
  
 // Граничные значения  
 assertEquals(0.0, calculator.calculateSeriesResistance(), 0.001) // пустой список  
 assertEquals(5.0, calculator.calculateSeriesResistance(5.0), 0.001) // один элемент  
 assertEquals(0.0, calculator.calculateSeriesResistance(0.0, 0.0, 0.0), 0.001) // нулевые сопротивления  
 }

@Test  
 fun calculateParallelResistance\_isCorrect() {  
 // Обычные значения  
 assertEquals(3.333, calculator.calculateParallelResistance(5.0, 10.0), 0.001)  
 assertEquals(2.0, calculator.calculateParallelResistance(6.0, 6.0, 6.0), 0.001)  
  
 // Граничные значения  
 assertEquals(0.0, calculator.calculateParallelResistance(), 0.001) // пустой список  
 assertEquals(5.0, calculator.calculateParallelResistance(5.0), 0.001) // один элемент  
 assertEquals(0.0, calculator.calculateParallelResistance(0.0, 10.0), 0.001) // одно сопротивление равно 0  
 }  
  
 @Test  
 fun calculateInductiveReactance\_isCorrect() {  
 // Обычные значения  
 assertEquals(62.831, calculator.calculateInductiveReactance(50.0, 0.2), 0.001)  
 assertEquals(0.0, calculator.calculateInductiveReactance(0.0, 10.0), 0.001)  
 assertEquals(0.0, calculator.calculateInductiveReactance(50.0, 0.0), 0.001)  
  
 // Граничные значения (высокая частота)  
 assertEquals(6283.185, calculator.calculateInductiveReactance(1000.0, 1.0), 0.001)  
 }  
}

1. Тесты были запущены. Все тесты в результате выполнены успешно (рис. 1).

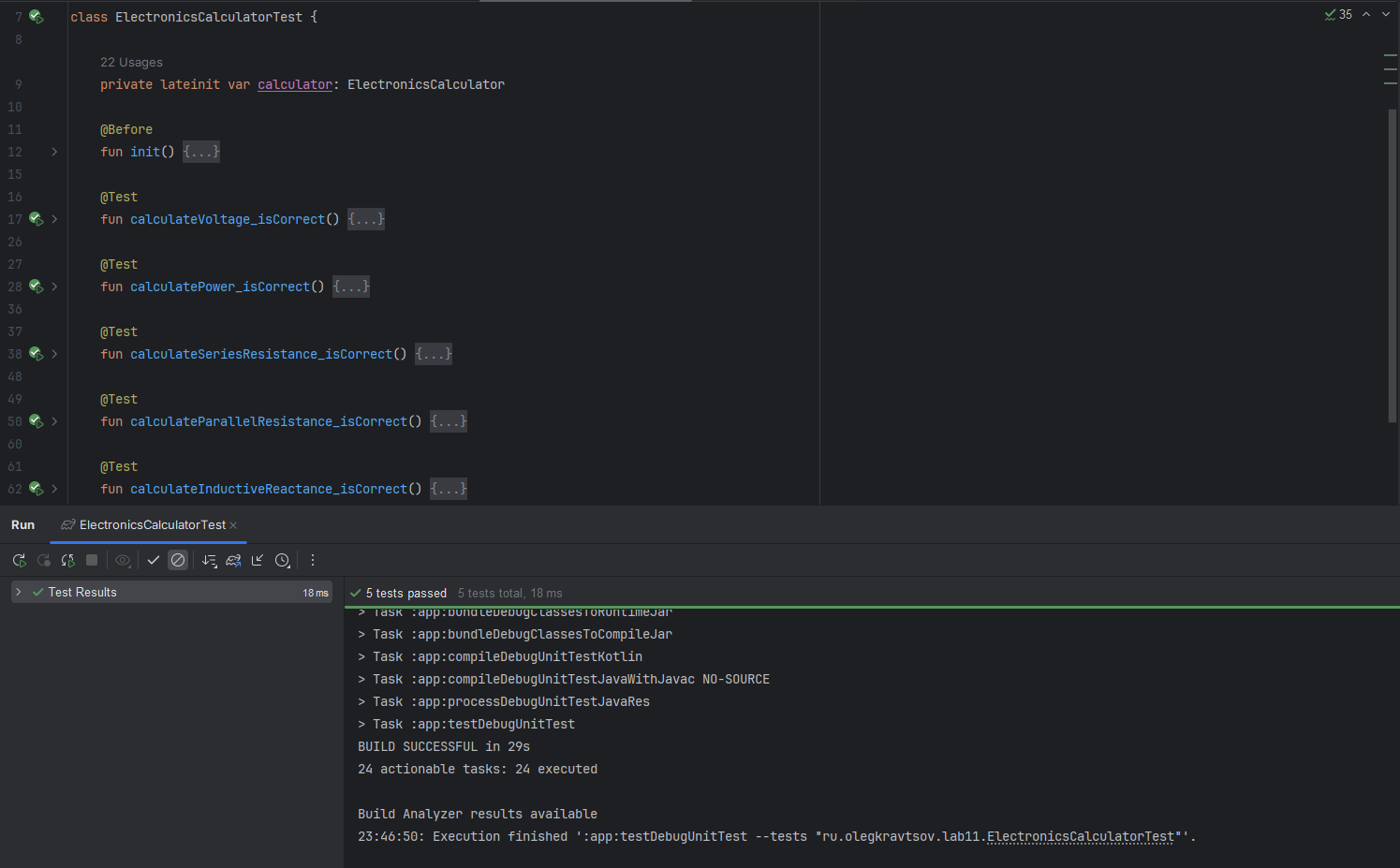


Рисунок 1 – Успешное выполнение всех тестов

1. В класс ElectronicsCalculator намеренно внесена ошибка в метод calculateVoltage (замена умножения на сложение):

fun calculateVoltage(current: Double, resistance: Double): Double {  
 return current + resistance // должно быть \*  
}

1. Тесты были запущены повторно. Результат — неудачное выполнение (рис. 2). Тест calculateVoltage\_isCorrect упал на assert-методах assertEquals. Ожидаемое значение: 12.0, фактическое: 8.0 (2.0 + 6.0 вместо 2.0 \* 6.0). Ошибка возникла из-за неправильной математической операции. Все остальные тесты прошли успешно, так как ошибка была локализована только в одном методе.

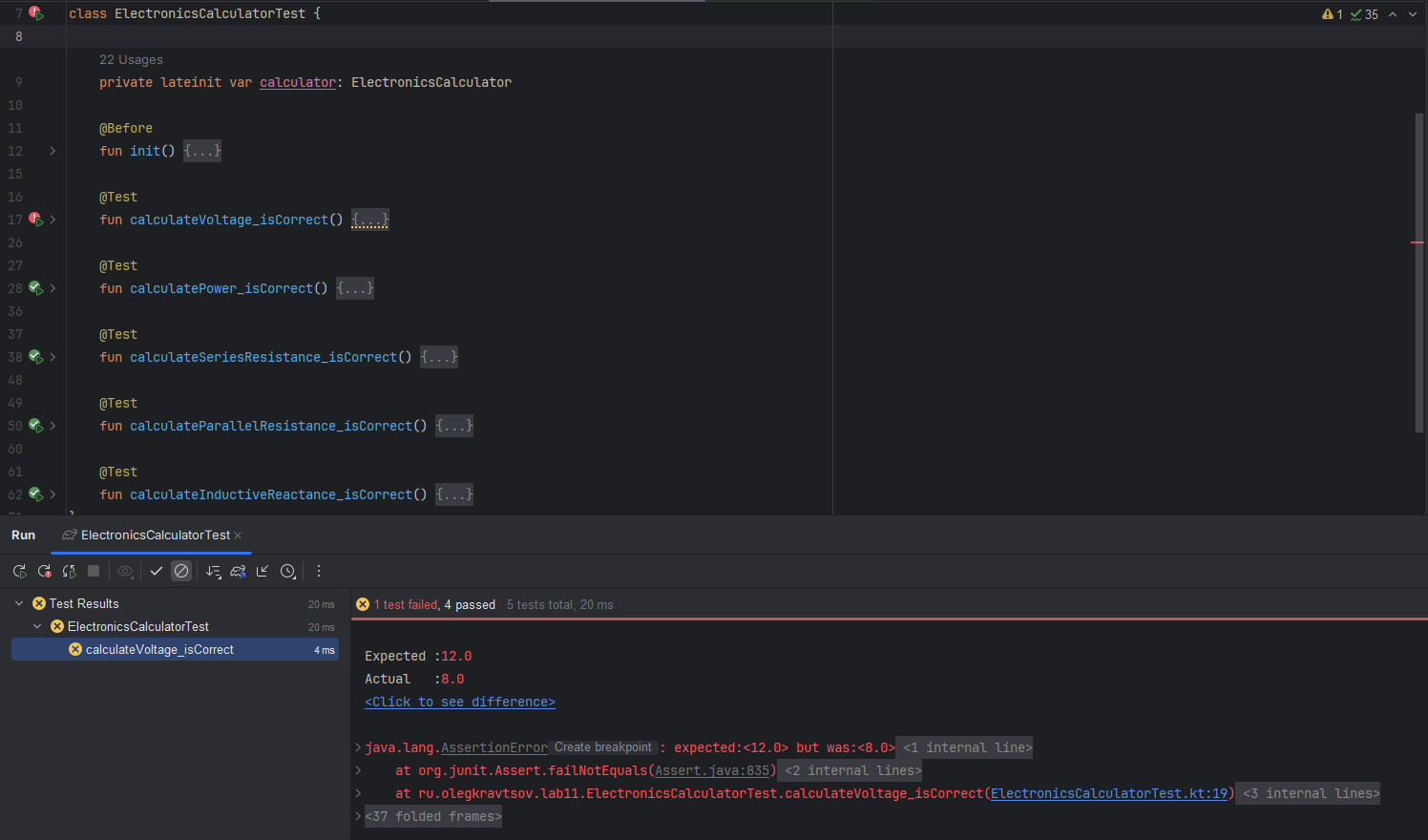


Рисунок 2 — Неудачное выполнение тестов после внесения ошибки

**Результат работы**

Разработан класс ElectronicsCalculator для расчётов в электротехнике, содержащий 5 методов:

* Расчёт напряжения по закону Ома
* Расчёт мощности
* Расчёт общего сопротивления последовательной цепи
* Расчёт общего сопротивления параллельной цепи
* Расчёт реактивного сопротивления катушки

Созданы комплексные юнит-тесты, покрывающие обычные и граничные значения параметров. Тесты успешно выполняются при корректной реализации методов и обнаруживают ошибки при их наличии.

# Выводы

В ходе выполнения лабораторной работы были освоены принципы юнит-тестирования в Android-приложениях. Разработан класс для вычислений в области электротехники с методами, основанными на физических законах. Созданы комплексные тесты, использующие assert-методы для проверки корректности расчётов на обычных и граничных значениях.

Продемонстрирована эффективность юнит-тестирования для обнаружения ошибок в логике программы — намеренно внесённая ошибка была сразу обнаружена соответствующими тестами. Тестирование доказало свою практическую ценность как инструмент обеспечения качества кода и предотвращения регрессионных ошибок при дальнейшей разработке.

Полученные навыки позволяют создавать надёжный и сопровождаемый код с автоматизированной проверкой его корректности, что соответствует современным стандартам промышленной разработки программного обеспечения.