

# Основы Gradle

## Цель работы

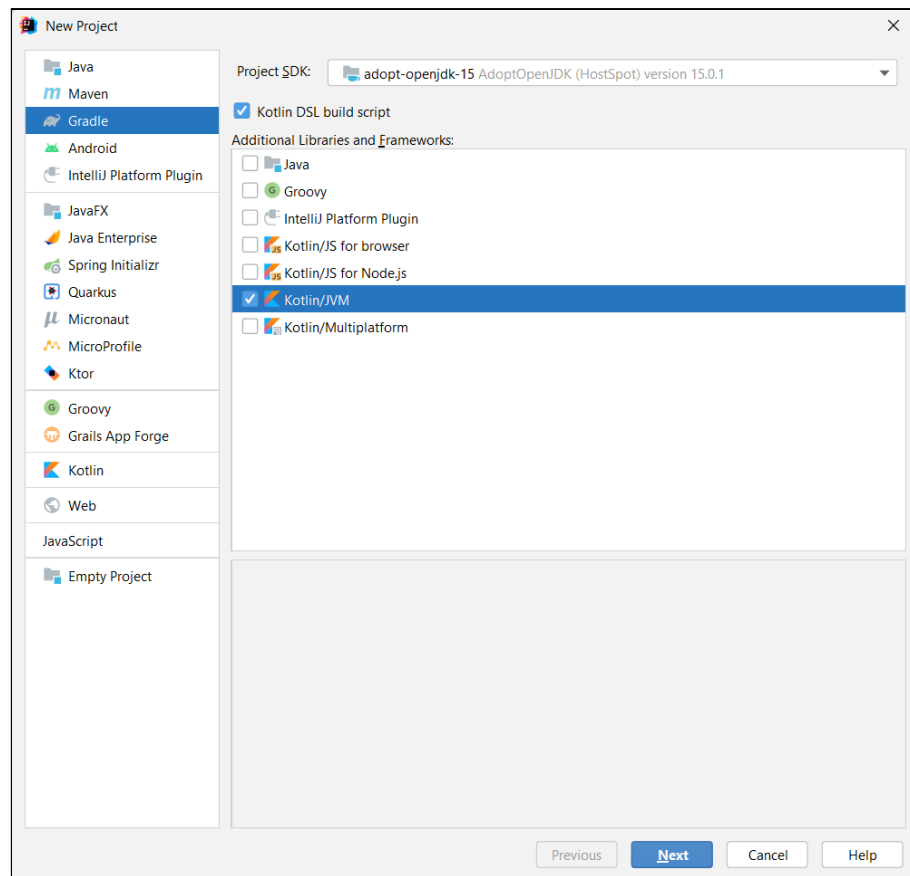
- Обучиться приемам работы с системой автоматической сборки Gradle

## Дополнительные материалы

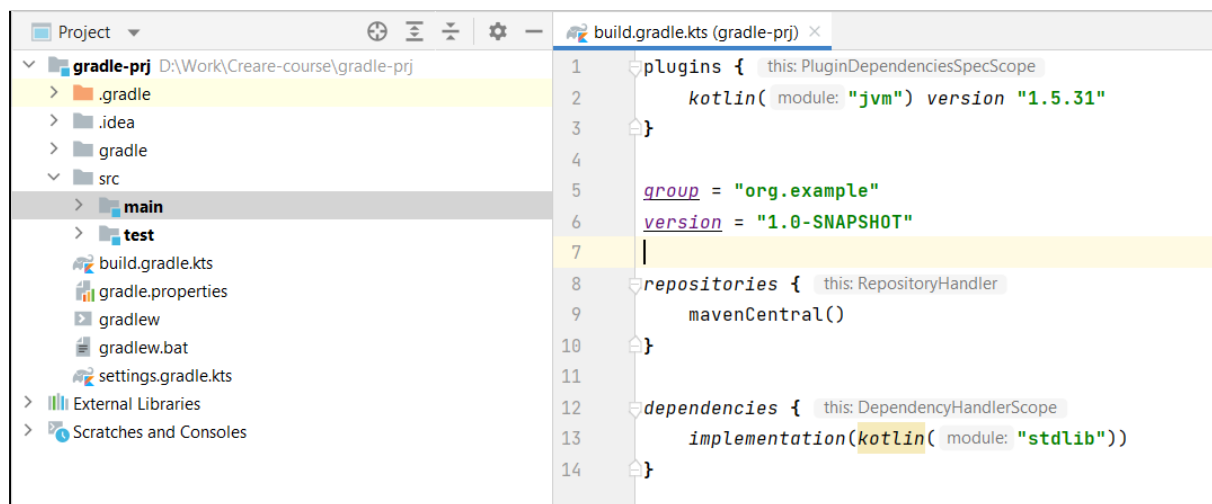
## Практическая работа

### **Создание много-модульного проекта**

1. Открываем IntelliJ Idea, которая была установлена ранее.
2. После открытия Idea нажимаем на кнопку создания проекта New Project.

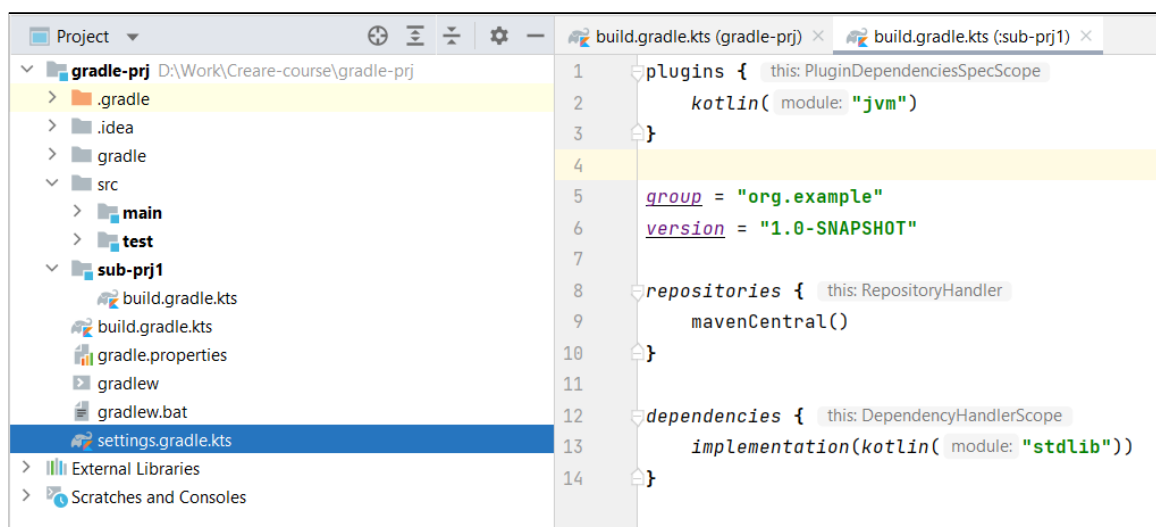
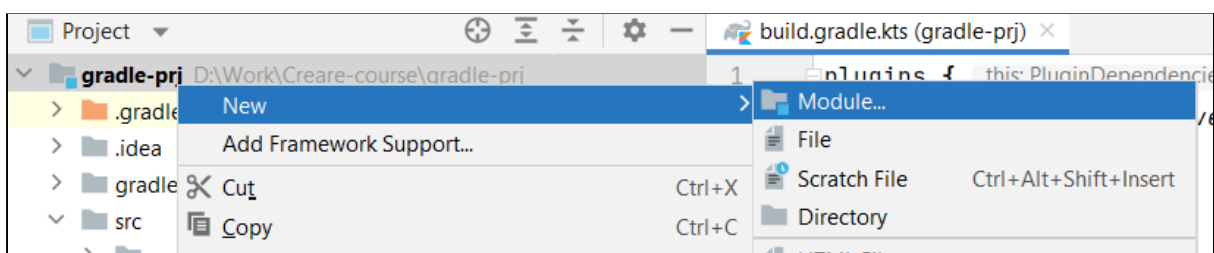


3. В качестве проекта выбираем gradle. Далее выбираем в качестве DSL, Kotlin. Для этого нужно выбрать пункт **Kotlin DSL build script**. А также выбираем Kotlin/JVM. И нажимаем **далее**.



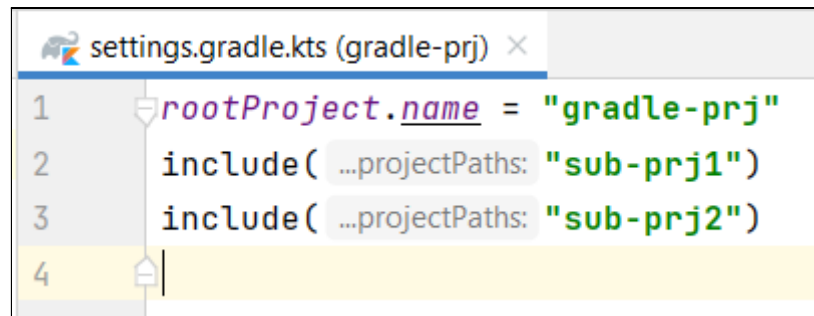
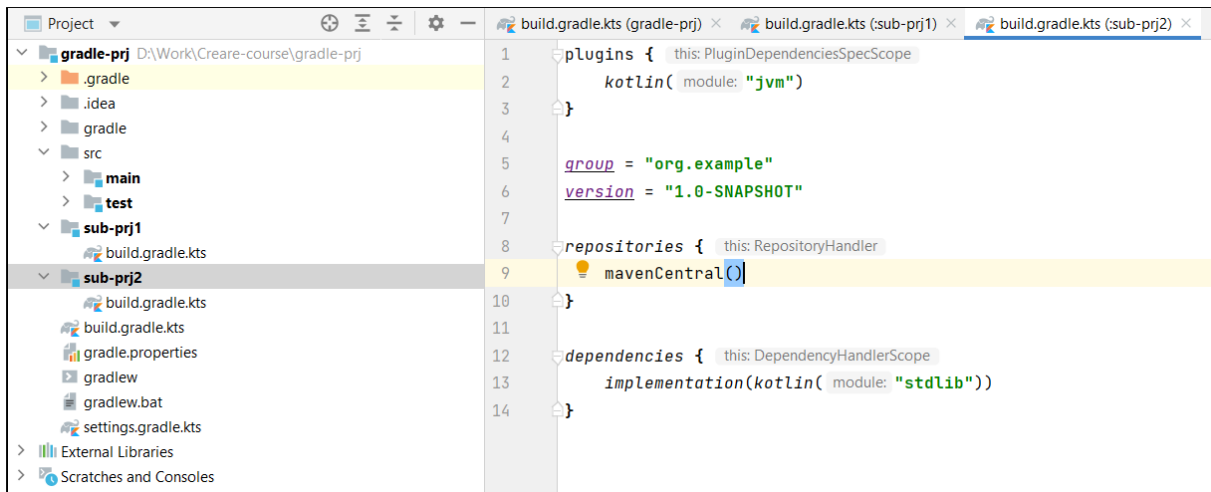
4. Так должен выглядеть проект после создания. Версия плагина kotlin указывается только в главном build.gradle проекта, поэтому после создания модуля нужно будет убрать автоматически созданную версию, чтобы избежать ошибок.

5. Теперь надо создать несколько дополнительных модулей в проекте. Для этого выбираем основной проект и кликаем ПКМ и выбираем модуль в качестве создания.



6. Вот так должен выглядеть проект после создания.

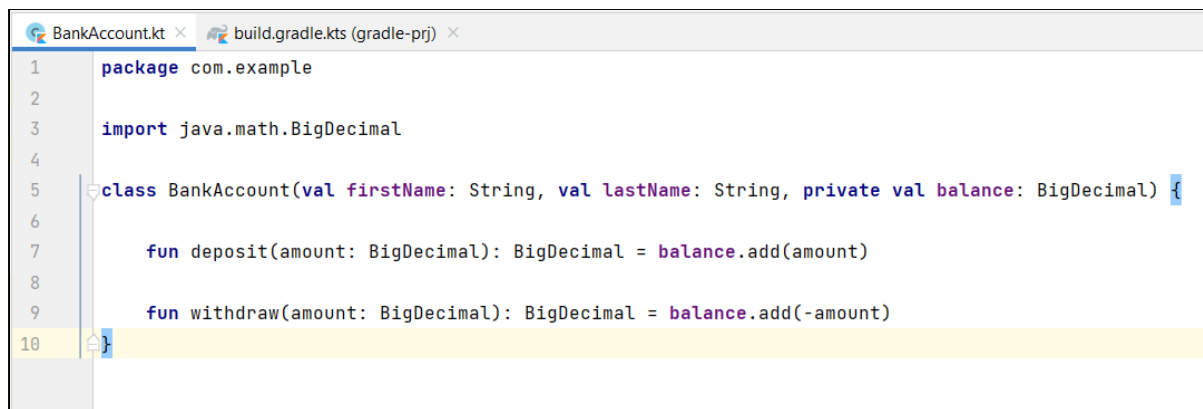
7. Теперь таким же образом создаем еще один модуль.



8. Файл settings.gradle все добавится автоматически. Но если он пуст, то новые модули требуется добавить самостоятельно, прописав include.

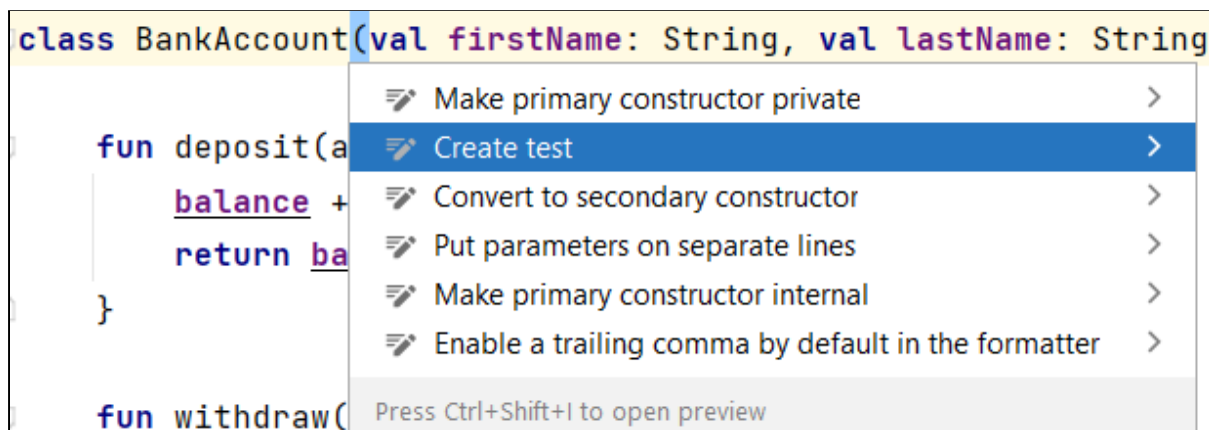
## Создание Junit тестов

1. Для тестов нужен gradle проект, поэтому его нужно создать также, как и в предыдущем примере. Для создания тестов нужен объект для тестирования, поэтому создадим его. Это будет банковский аккаунт с несколькими методами. В нем есть имя, фамилия и баланс. Методы для увеличения и уменьшения баланса.



```
1 package com.example
2
3 import java.math.BigDecimal
4
5 class BankAccount(val firstName: String, val lastName: String, private val balance: BigDecimal) {
6
7     fun deposit(amount: BigDecimal): BigDecimal = balance.add(amount)
8
9     fun withdraw(amount: BigDecimal): BigDecimal = balance.add(-amount)
10 }
```

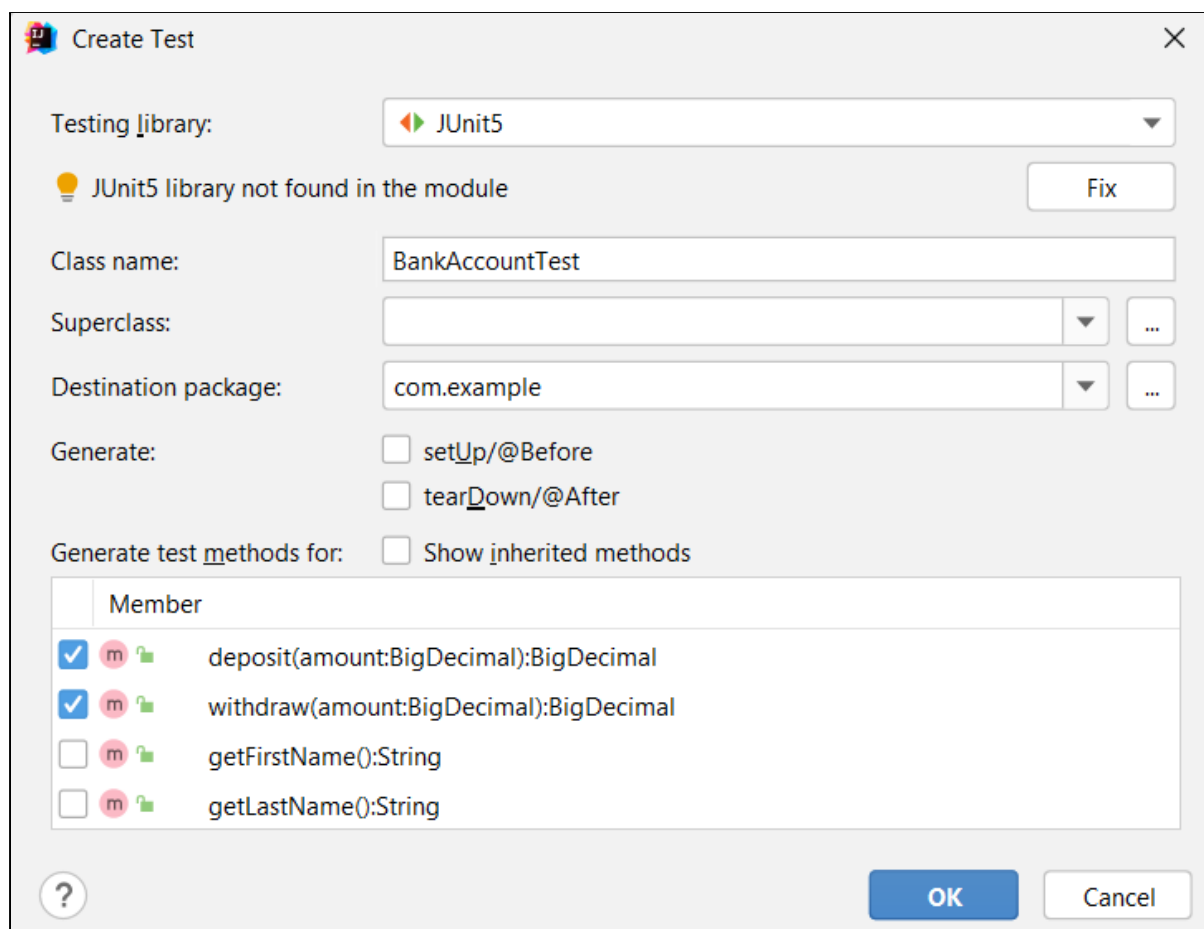
2. Сочетанием клавиш **alt+enter**, можно вызвать меню в котором будет возможность создать тесты для класс BankAccount.



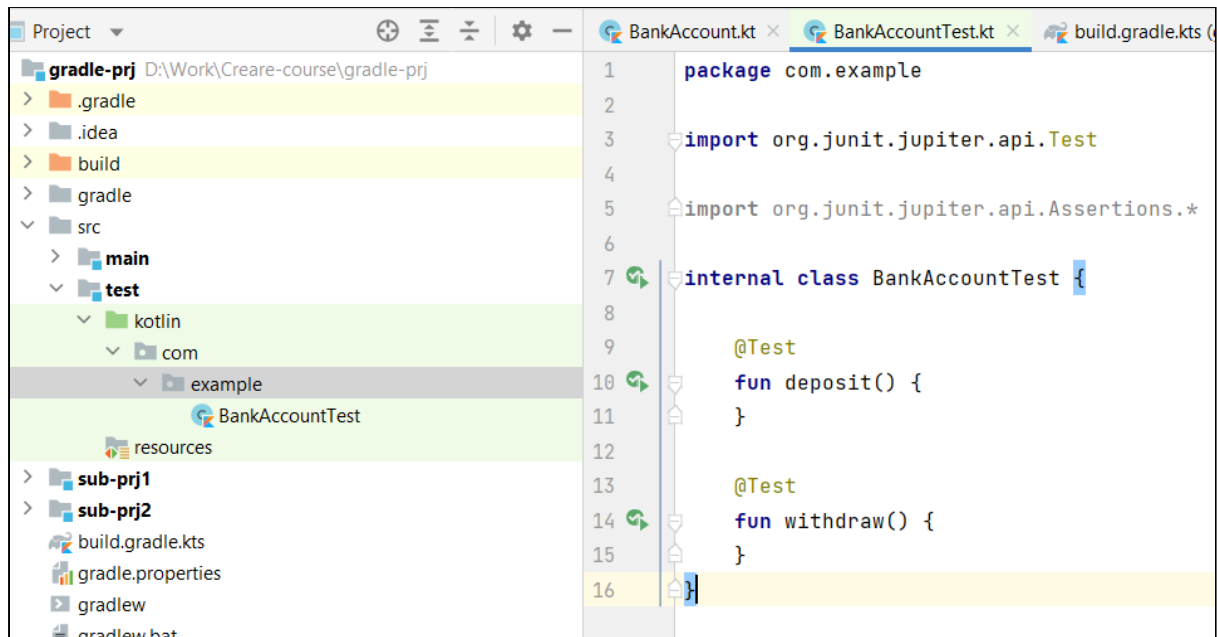
```
class BankAccount(val firstName: String, val lastName: String
    fun deposit(a
        balance +
        return ba
    }
    fun withdraw(
```

- Make primary constructor private >
- Create test >**
- Convert to secondary constructor >
- Put parameters on separate lines >
- Make primary constructor internal >
- Enable a trailing comma by default in the formatter >

Press Ctrl+Shift+I to open preview

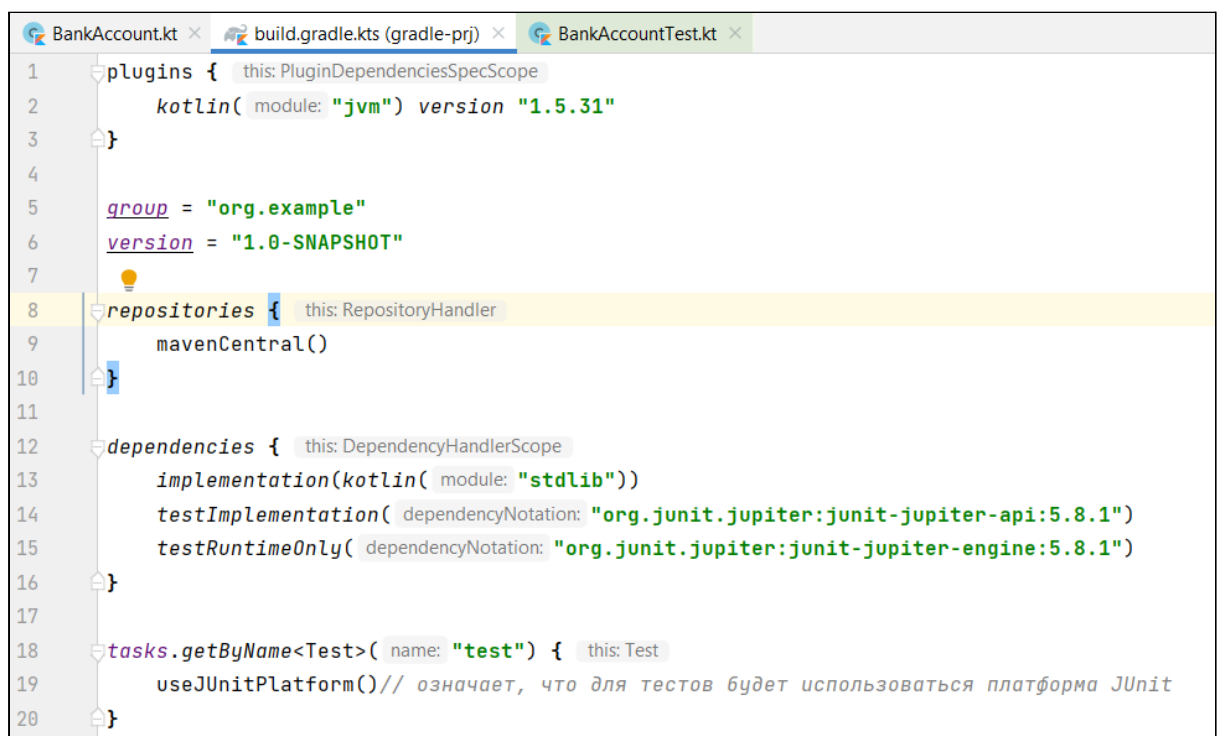


3. Выбираем как будет называть пакет для класса тестирования, который создается в директории test(). Затем выбираем нужные методы и продолжаем.



4. У нас создается класс для в котором будет проходить тестирования.

5. Также нужно добавить зависимости в gradle проект для тестов и выбрать в качестве тестов платформу JUnit.



6. Теперь можно перейти обратно к классу и написать тесты. Для этого мы создаем экземпляр класса, который собираемся проверять и в каждом методе пишем соответствующий тест для метода проверяемого класса, как на рисунке. У JUnit есть класс `Assertions`, который представляет собой набор служебных методов поддерживающих утверждение условий в тестах. Более подробно этот класс можно изучить на по [ссылке](#).



```
9
10 internal class BankAccountTest {
11
12     var bankAccount = BankAccount( firstName: "Mark", lastName: "Petrov", BigDecimal( val: 500))
13
14     @Test
15     fun deposit() {
16         var balance = bankAccount.deposit(BigDecimal( val: 200))
17         Assertions.assertEquals(balance, BigDecimal( val: 700))
18     }
19
20     @Test
21     fun withdraw() {
22         var balance = bankAccount.withdraw(BigDecimal( val: 200))
23         Assertions.assertEquals(balance, BigDecimal( val: 300))
24     }
25 }
```

7. После создания тестов запускаем их с помощью консоли (как на рисунке). Чтобы ее вызвать нужно нажать сочетание клавиш `ctrl+ctrl` и прописать команду `gradle test`. Либо можно запустить тесты сразу из класс, нажимая кнопку `run` рядом с названием класса.





Test Results	19 ms	Starting Gradle Daemon...
BankAccountTest	19 ms	Gradle Daemon started in 1 s 532 ms
withdraw()	17 ms	
getFirstName()	1 ms	> Task :compileKotlin UP-TO-DATE
getLastName()	0 ms	> Task :compileJava NO-SOURCE
deposit()	1 ms	> Task :processResources NO-SOURCE
		> Task :classes UP-TO-DATE
		> Task :compileTestKotlin
		'compileTestJava' task (current target is 15) and 'compileTestKotlin' task (current target is 1.8) jvm target compatibility should
		> Task :compileTestJava NO-SOURCE

8. После этого все тесты запустятся и можно будет увидеть результаты. Если какие-то тесты не прошли, то появится отчет и причина не прохождения тестов.

## Практические задания

1. Перенести в созданный проект сборки предыдущих проектов.
2. Не углубляясь в JUnit написать простые тесты для заданий, которые были в прошлой лабораторной работе (либо телефонная книга, либо бинарное дерево).
3. Добавить какую-нибудь зависимость, например на slf4j+logback. Нужно написать небольшую логику, которая будет использовать данную зависимость.

## Теоретические вопросы

1. Какие особенности и функции есть в gradle ?
2. Что такое DSL ?
3. Основные DSL в Gradle ?
4. Из чего состоит проект gradle ?
5. Что такое wrapper ?
6. Что такое плагин ?

7. Какие есть типы плагинов и чем они отличаются ?
8. Что такое doFirst и doLast в пользовательской задаче ?
9. Что такое пользовательская задача ?
10. Что такое Source set ?