# Один модуль — *хорошо*, несколько — *лучше.*

# Егор Пиший

Java Developer · PVS-Studio

Разрабатываю статические анализаторы для JavaScript/TypeScript и Java.
Работаю с Gradle каждый день.
Выступаю на конференциях.

github.com/Pupcheg  t.me/supcheg

КТО МЫ

# PVS-Studio

Статический анализатор кода. Находит ошибки и потенциальные уязвимости до того, как они попадут в прод.

- Анализатор для C, C++, C# и Java
- Разрабатываем новые анализаторы для JavaScript/TypeScript и Go

pvs-studio.ru/
training_gradle

ВЕБИНАР 1 ИЗ 5

# О *курсе*

*После курса вы будете* **уверенно работать** *с Gradle в реальных проектах.*

> Kotlin DSL · мультимодульность · Convention Plugins · агрегация тестов

ВЕБИНАР 1 ИЗ 5

# О *курсе*

*После курса вы будете* **уверенно работать** *с Gradle в реальных проектах.*

> Kotlin DSL · мультимодульность · Convention Plugins · агрегация тестов

01 **Практика с первого вебинара.** Каждый урок — лайфкодинг с нуля до рабочего проекта.

# О *курсе*

*После курса вы будете* **уверенно работать** *с Gradle в реальных проектах.*

> Kotlin DSL · мультимодульность · Convention Plugins · агрегация тестов

| | |
|---|---|
| 01 | **Практика с первого вебинара.** Каждый урок — лайфкодинг с нуля до рабочего проекта. |
| 02 | **Современный стек.** Java 25, Gradle 9, Kotlin DSL, JUnit 6. |

# О *курсе*

*После курса вы будете* **уверенно работать** *с Gradle в реальных проектах.*

> Kotlin DSL · мультимодульность · Convention Plugins · агрегация тестов

| | |
|---|---|
| 01 | **Практика с первого вебинара.** Каждый урок — лайфкодинг с нуля до рабочего проекта. |
| 02 | **Современный стек.** Java 25, Gradle 9, Kotlin DSL, JUnit 6. |
| 03 | **Готовый шаблон.** В конце вебинара у нас получится шаблон, с которым будет быстро и удобно создавать мультимодульные проекты. |

СЕГОДНЯ

# Что *разберём?*

СЕГОДНЯ

# Что *разберём?*

01 **Зачем вообще модули**

Мотивация, сравнение с монолитом

СЕГОДНЯ

# Что *разберём?*

01 **Зачем вообще модули**

Мотивация, сравнение с монолитом

02 **Когда это оправдано**

Три чётких критерия

СЕГОДНЯ

# Что *разберём?*

01 **Зачем вообще модули**

Мотивация, сравнение с монолитом

02 **Когда это оправдано**

Три чётких критерия

03 **Жизненный цикл Gradle**

Инициализация → конфигурация → выполнение

СЕГОДНЯ

# Что *разберём?*

01 **Зачем вообще модули**
Мотивация, сравнение с монолитом

02 **Когда это оправдано**
Три чётких критерия

03 **Жизненный цикл Gradle**
Инициализация → конфигурация → выполнение

04 **Лайфкодинг**
Собираем простейший мультимодульный проект

# Зачем вообще *модули?*

| АСПЕКТ | МОНОЛИТ · ОДИН BUILD.GRADLE.KTS | МУЛЬТИ-МОДУЛЬ · НЕСКОЛЬКО BUILD.GRADLE.KTS |
|---|---|---|

# Зачем вообще *модули?*

| АСПЕКТ | ● **МОНОЛИТ** · ОДИН BUILD.GRADLE.KTS | ● **МУЛЬТИ-МОДУЛЬ** · НЕСКОЛЬКО BUILD.GRADLE.KTS |
|---|---|---|
| Переиспользование | – копипаста между проектами | ✓ выносится в либу, подключается везде |

# Зачем вообще *модули?*

| АСПЕКТ | ● **МОНОЛИТ** · ОДИН BUILD.GRADLE.KTS | ● **МУЛЬТИ-МОДУЛЬ** · НЕСКОЛЬКО BUILD.GRADLE.KTS |
|---|---|---|
| Переиспользование | – копипаста между проектами | ✓ выносится в либу, подключается везде |

ПРИМЕР · РАЗДЕЛЬНАЯ ПУБЛИКАЦИЯ

# Зачем вообще *модули?*

| АСПЕКТ | ● **МОНОЛИТ** · ОДИН BUILD.GRADLE.KTS | ● **МУЛЬТИ-МОДУЛЬ** · НЕСКОЛЬКО BUILD.GRADLE.KTS |
|---|---|---|
| Переиспользование | – копипаста между проектами | ✓ выносится в либу, подключается везде |
| Сборка / кэш | – компилируется всё целиком | ✓ только то, что реально изменилось |

# *Сборка / кэш* в действии

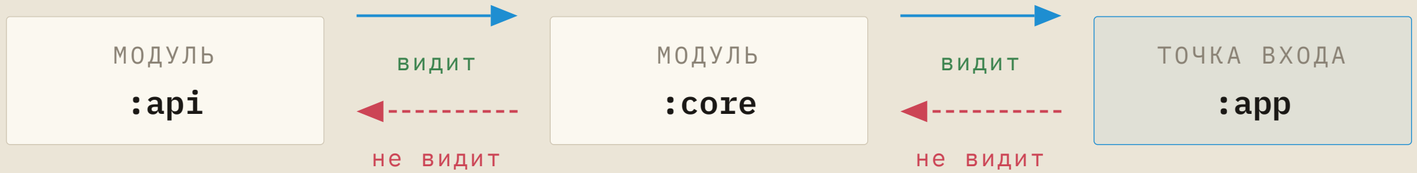
```
$ ./GRADLEW :APP:BUILD
> Task :core:compileJava UP-TO-DATE
> Task :core:jar UP-TO-DATE
> Task :app:compileJava
> Task :app:jar

BUILD SUCCESSFUL in 1s
2 actionable tasks: 2 executed, 2 up-to-date
```

# Зачем вообще *модули?*

| АСПЕКТ | ● **МОНОЛИТ** · ОДИН BUILD.GRADLE.KTS | ● **МУЛЬТИ-МОДУЛЬ** · НЕСКОЛЬКО BUILD.GRADLE.KTS |
|---|---|---|
| Переиспользование | – копипаста между проектами | ✓ выносится в либу, подключается везде |
| Сборка / кэш | – компилируется всё целиком | ✓ только то, что реально изменилось |
| Границы зависимостей | – любой класс видит любой | ✓ явно в `dependencies { }` |

# *Границы* зависимостей

ГРАФ ЗАВИСИМОСТЕЙ · КТО КОГО ВИДИТ



# Когда это *оправдано?*

*Не нужно* **дробить всё подряд.**

# Когда это *оправдано?*

*Не нужно* **дробить всё подряд.**

01 **Переиспользование.** Код нужен в нескольких местах — в другом модуле, в другом проекте, в тестах.

# Когда это *оправдано?*

*Не нужно* **дробить всё подряд.**

| | |
|---|---|
| 01 | **Переиспользование.** Код нужен в нескольких местах — в другом модуле, в другом проекте, в тестах. |
| 02 | **Границы слоёв.** Команда хочет чётко разграничить архитектурные слои и запретить лишние связи. |

# Когда это *оправдано?*

*Не нужно* **дробить всё подряд.**

| | |
|---|---|
| 01 | **Переиспользование.** Код нужен в нескольких местах — в другом модуле, в другом проекте, в тестах. |
| 02 | **Границы слоёв.** Команда хочет чётко разграничить архитектурные слои и запретить лишние связи. |
| 03 | **Независимая публикация.** Артефакт уезжает в Maven / внутренний репозиторий отдельно от основного приложения. |

# Жизненный цикл *Gradle*

# Жизненный цикл *Gradle*

ФАЗА 1

## Инициализация

Gradle читает `settings.gradle.kts` и узнаёт, какие модули существуют в проекте и как они называются.

РЕЗУЛЬТАТ

список Project-объектов

# Жизненный цикл *Gradle*

ФАЗА 1

## Инициализация

Gradle читает `settings.gradle.kts` и узнаёт, какие модули существуют в проекте и как они называются.

РЕЗУЛЬТАТ

список Project-объектов

ФАЗА 2

## Конфигурация

Выполняются все `build.gradle.kts`, регистрируются задачи и строится граф их зависимостей.

РЕЗУЛЬТАТ

граф задач (DAG)

# Жизненный цикл *Gradle*

ФАЗА 1

## Инициализация

Gradle читает `settings.gradle.kts` и узнаёт, какие модули существуют в проекте и как они называются.

РЕЗУЛЬТАТ

список Project-объектов

ФАЗА 2

## Конфигурация

Выполняются все `build.gradle.kts`, регистрируются задачи и строится граф их зависимостей.

РЕЗУЛЬТАТ

граф задач (DAG)

ФАЗА 3

## Выполнение

Gradle запускает только те задачи, которые нужны для требуемой цели, и только те, чей вход изменился.

РЕЗУЛЬТАТ

собранные артефакты

ПРАКТИКА

# Идём на *лайфкодинг*

ВПЕРЕДИ ЕЩЁ 4 ВЕБИНАРА

# Что *дальше?*

02

ВЕБИНАР 2 · СЛЕДУЮЩИЙ

### Зависимости между модулями

Разница `api` и `implementation`, плагины `java-library` / `java-application`, конфигурации `compileOnly` и `runtimeOnly`.

03

ВЕБИНАР 3

### Version Catalog и BOM

Централизованное управление версиями через `libs.versions.toml`. BOM вместо явных версий (JUnit 6), мотивация к Convention Plugins.

04

ВЕБИНАР 4

### Convention Plugins: убираем дублирование

Сравнение `subprojects {}` и `build-logic`. Precompiled script plugin, применение во всех модулях.

05

ВЕБИНАР 5

### Тесты и агрегация отчётов

Единый отчёт по тестам и покрытию из всех модулей. Плагины `test-report-aggregation` и `jacoco-report-aggregation`.

ВПЕРЕДИ ЕЩЁ 4 ВЕБИНАРА

# Что *дальше?*

02

ВЕБИНАР 2 · СЛЕДУЮЩИЙ

### Зависимости между модулями

Разница `api` и `implementation`, плагины `java-library` / `java-application`, конфигурации `compileOnly` и `runtimeOnly`.

03

ВЕБИНАР 3

### Version Catalog и BOM

Централизованное управление версиями через `libs.versions.toml`. BOM вместо явных версий (JUnit 6), мотивация к Convention Plugins.

04

ВЕБИНАР 4

### Convention Plugins: убираем дублирование

Сравнение `subprojects {}` и `build-logic`. Precompiled script plugin, применение во всех модулях.

05

ВЕБИНАР 5

### Тесты и агрегация отчётов

Единый отчёт по тестам и покрытию из всех модулей. Плагины `test-report-aggregation` и `jacoco-report-aggregation`.

ВПЕРЕДИ ЕЩЁ 4 ВЕБИНАРА

# Что *дальше?*

02

ВЕБИНАР 2 · СЛЕДУЮЩИЙ

### Зависимости между модулями

Разница `api` и `implementation`, плагины `java-library` / `java-application`, конфигурации `compileOnly` и `runtimeOnly`.

03

ВЕБИНАР 3

### Version Catalog и BOM

Централизованное управление версиями через `libs.versions.toml`. BOM вместо явных версий (JUnit 6), мотивация к Convention Plugins.

04

ВЕБИНАР 4

### Convention Plugins: убираем дублирование

Сравнение `subprojects {}` и `build-logic`. Precompiled script plugin, применение во всех модулях.

05

ВЕБИНАР 5

### Тесты и агрегация отчётов

Единый отчёт по тестам и покрытию из всех модулей. Плагины `test-report-aggregation` и `jacoco-report-aggregation`.

ВПЕРЕДИ ЕЩЁ 4 ВЕБИНАРА

# Что *дальше?*

02

ВЕБИНАР 2 · СЛЕДУЮЩИЙ

### Зависимости между модулями

Разница `api` и `implementation`, плагины `java-library` / `java-application`, конфигурации `compileOnly` и `runtimeOnly`.

03

ВЕБИНАР 3

### Version Catalog и BOM

Централизованное управление версиями через `libs.versions.toml`. BOM вместо явных версий (JUnit 6), мотивация к Convention Plugins.

04

ВЕБИНАР 4

### Convention Plugins: убираем дублирование

Сравнение `subprojects {}` и `build-logic`. Precompiled script plugin, применение во всех модулях.

05

ВЕБИНАР 5

### Тесты и агрегация отчётов

Единый отчёт по тестам и покрытию из всех модулей. Плагины `test-report-aggregation` и `jacoco-report-aggregation`.

ВПЕРЕДИ ЕЩЁ 4 ВЕБИНАРА

# Что *дальше?*

02

ВЕБИНАР 2 · СЛЕДУЮЩИЙ

### Зависимости между модулями

Разница `api` и `implementation`, плагины `java-library` / `java-application`, конфигурации `compileOnly` и `runtimeOnly`.

03

ВЕБИНАР 3

### Version Catalog и BOM

Централизованное управление версиями через `libs.versions.toml`. BOM вместо явных версий (JUnit 6), мотивация к Convention Plugins.

04

ВЕБИНАР 4

### Convention Plugins: убираем дублирование

Сравнение `subprojects {}` и `build-logic`. Precompiled script plugin, применение во всех модулях.

05

ВЕБИНАР 5

### Тесты и агрегация отчётов

Единый отчёт по тестам и покрытию из всех модулей. Плагины `test-report-aggregation` и `jacoco-report-aggregation`.

КОНЕЦ ВЕБИНАРА 1

# Вопросы *и ответы*

Спрашиваете — отвечаем