Организация индивидуального Java-проекта

Репозиторий, Gradle и первичный Project Proposal

Косило Павел Андреевич

5 сентября 2026

Содержание

Цель и задачи занятия

Цель

Запустить индивидуальный проект как инженерный артефакт: репозиторий, воспроизводимая сборка и осмысленный черновик Proposal.

Задачи

- разобрать этапы жизненного цикла ПО и что из них делаем сегодня;
- собрать toolchain: JDK, IDEA, Gradle, Git;
- зафиксировать правила работы с AI;
- показать три пути и семестровый минимум;
- выбрать тему и сформулировать проблему.

Проект и жизненный цикл

Курс вокруг индивидуального проекта

Сквозная система на семестр. Лабораторные — отдельные упражнения; проект — одна предметная область от проблемы до работающего кода.

Сегодня

- проектный Git-репозиторий и remote;
- Gradle-проект с wrapper;
- черновик Project Proposal.

Дальше

- модель предметной области;
- код, тесты, итерации;
- то, что можно показать и защитить.

Жизненный цикл ПО

Этап	Вопрос	Артефакт
Анализ	Какую проблему решаем и для кого?	Проблема, аудитория, Proposal
Проектирование	Какие сущности и границы v1?	Модель, scope in/out
Реализация	Как это устроено в коде?	Репозиторий, сборка, исходники
Тестирование	Что считается работающим?	Тесты, критерии приёмки
Поставка	Как воспроизвести у другого?	README, wrapper, remote
Сопровождение	Что менять дальше?	История коммитов, итерации

Где мы сегодня

Inception + bootstrap. Ещё нет доменной модели в коде — есть решение, *что* строить и *чем* это собирать.

Анализ

- выбрать одну из 30 тематик;
- сформулировать проблему;
- назвать целевую аудиторию.

Bootstrap

- проектный репозиторий на GitHub или GitLab;
- Gradle Wrapper в проекте;
- README и успешный `./gradlew build`.

Инструменты

Четыре инструмента курса

Инструмент	Зачем	Сегодня
JDK	Компилятор и стандартная библиотека Java	JDK 21 LTS
IntelliJ IDEA	Редактирование, запуск, отладка	Открыть как Gradle-проект
Gradle	Воспроизводимая сборка и зависимости	Wrapper + Kotlin DSL
Git	История, remote, совместная работа	Репозиторий и первый коммит

Gradle — основной инструмент практики. Maven не требуется устанавливать: его достаточно узнавать в чужих проектах.

JDK

JDK — комплект разработчика: компилятор, стандартная библиотека, инструменты. **JRE** — только среда выполнения; для курса его недостаточно.

Что внутри

- `javac` — компиляция;
- `java` — запуск;
- стандартная библиотека.

Ориентир

- **JDK 21 LTS**;
- toolchain задаём в Gradle, а не «какая Java стоит у меня»;
- IDEA подхватит версию из проекта.

IntelliJ IDEA

Основная IDE курса. Проект открываем как Gradle, а не как «папку с .java».

Как открывать

- Open / New Project — Gradle;
- использовать **Gradle Wrapper**;
- дождаться импорта, затем `build`.

Не коммитить

- каталог `.idea/`;
- файлы `*.iml`;
- локальные run-конфигурации, если они не общие.

Gradle: зачем сборка и wrapper

Не собирать руками через `javac`. Сборка знает исходники, тесты, зависимости и версию JDK.

- layout как у Maven: `src/main/java`, `src/test/java`;
- **wrapper** (`gradlew` / `gradlew.bat`) — та же версия Gradle у всех;
- задачи: `build`, `test`; `run` — когда подключите Application plugin.

Команда успеха

```
./gradlew build
```

На Windows — `gradlew.bat build`.

Минимальный Gradle Kotlin DSL

`settings.gradle.kts`

```
rootProject.name = "task-tracker"
```

Имя проекта — осмысленное, по теме, не `untitled`.

`build.gradle.kts`

```
plugins {
    java
}

java {
    toolchain {
        languageVersion.set(
            JavaLanguageVersion.of(21))
    }
}
```

Maven: знать в лицо

В индустрии часто Maven. На курсе практическая сборка выполняется на Gradle; Maven нужен как ознакомительный промышленный аналог.

Что увидите

- файл `pom.xml` вместо `build.gradle.kts`;
- тот же `src/main/java`;
- фазы: `compile`, `test`, `package`.

Как читать

- координаты: groupId, artifactId, version;
- зависимости — в XML, не в Kotlin DSL;
- IDEA откроет Maven-проект так же, как Gradle.

Практическое задание — только Gradle. Чужой корпоративный репозиторий вы можете встретить уже на Maven.

Git с первого дня

История и remote — часть проекта, не «потом зальём».

Локально

- `git init` (или New Project в IDEA);
- осмысленные коммиты;
- не коммитить сборку и кэш IDE.

Remote проекта

- GitHub **или** GitLab — на ваш выбор;
- приватный репозиторий допустим;
- это не контур сдачи лабораторных.

Гигиена репозитория

`.gitignore` — не в Git

```
.gradle/
build/
.idea/
*.class
*.iml
```

Коммитить: `gradlew`, `gradlew.bat`, `gradle/wrapper/`.

README

- что за проект и какая тематика;
- как собрать: `./gradlew build`;
- какая версия JDK ожидается.

README читает человек и AI: без него проект — чёрный ящик.

Лабораторные и проект — разные контуры

Не смешивать. Сдача заданий и проектная деятельность идут отдельно.

Лабораторные работы

- сдаются в курсе на Manytask2;
- сайт: manytask2.org;
- это не репозиторий семестрового проекта.

Индивидуальный проект

- отдельный GitHub или GitLab;
- Gradle, README, Proposal;
- ведётся вне контура лабораторных.

AI и ответственность

AI-friendly разработка

Структура помогает и человеку, и модели. Чем предсказуемее проект, тем полезнее подсказки.

- Gradle Wrapper: одна команда сборки без «а какая у вас версия».
- Kotlin DSL и стандартный layout — меньше двусмысленности.
- README и Proposal задают проблему, аудиторию и сущности.
- Имена пакетов и классов по домену, не `Main2` и `Test1`.

AI: правила

Можно

- черновик `build.gradle.kts` и README;
- заготовка тестов и `.gitignore`;
- вопросы «почему не собирается»;
- разбор ошибки сборки по логу.

Нельзя

- сдавать код, который вы не читали;
- коммитить то, чего не понимаете;
- подменять Proposal сгенерированным текстом без своей задачи;
- считать «у модели получилось» критерием готовности.

Ответственность за результат

Отвечает разработчик

- сборка зелёная у другого человека;
- смысл домена и границы v1;
- текст Proposal — ваша формулировка;
- содержимое каждого коммита.

Не снимает ответственность

- IDEA «сама создала проект»;
- Gradle «должен работать»;
- AI «написал за меня»;
- «у меня на машине запускалось».

Практика и Proposal

Практическое задание

Каркас проекта, не лабораторная на Manytask2. До следующей пары соберите репозиторий, а не доменную логику.

1. Личный Git-репозиторий проекта на GitHub или GitLab.
2. Gradle-проект с Kotlin DSL и **закоммиченным wrapper**.
3. Файл `.gitignore` (сборка, кэш IDE — вне Git).
4. README: тема, как собрать, какая JDK.
5. Успешная сборка: `./gradlew build`.

Критерий готовности

Готово, если

- клон с remote собирается без ручных шагов;
- в репозитории нет `build/` и `.gradle/`;
- README позволяет повторить сборку;
- выбран номер тематики (или согласована своя).

Ещё не готово

- проект только локально, без remote;
- wrapper не в Git — «поставьте Gradle»;
- README из двух слов;
- тема не выбрана, Proposal пустой.

Тридцать предметных тематик

Выбрать одну — или согласовать свою. Сущности в таблице — старт модели, не готовая архитектура.

- Сначала проблема и аудитория, потом сущности.
- Не брать «всё сразу»: биллинг и аналитика — чаще за пределами v1. Мобильное — опциональный продвинутый путь, не вместо web-клиента.
- Имена сущностей можно уточнить, если это следует из вашей задачи.
- Тема на семестр: сменить её позже будет дорого.

Тематики 1–10

№	Предметная тематика	Примеры сущностей
1	Система управления задачами и проектами	Project, Task, User, Comment
2	Электронная библиотека	Book, Author, Reader, Loan
3	Интернет-каталог товаров	Product, Category, Supplier, Price
4	Система управления учебным расписанием	Student, Course, Teacher, Lesson
5	Электронный журнал успеваемости	Student, Discipline, Grade, Teacher
6	Система бронирования аудиторий	Room, User, Booking, Schedule
7	Система заявок технической поддержки	Request, User, Category, Status
8	Складской учёт	Product, Warehouse, Stock, Movement
9	Система управления закупками	Supplier, Product, Order, Purchase
10	Личный финансовый менеджер	Account, Transaction, Category, Budget

Тематики 11–20

№	Предметная тематика	Примеры сущностей
11	Система учёта домашних расходов	Household, Expense, Category, Member
12	Система управления мероприятиями	Event, Participant, Venue, Registration
13	Система продажи билетов	Event, Ticket, Customer, Payment
14	Электронная очередь на обслуживание	Client, Service, Operator, Queue
15	Система записи на услуги	Client, Service, Employee, Appointment
16	Система управления фитнес-клубом	Client, Trainer, Subscription, Visit
17	Система управления гостиницей	Room, Guest, Booking, Payment
18	Система бронирования мест в хостеле	Room, Guest, Reservation, Bed
19	Система аренды автомобилей	Car, Customer, Rental, Tariff
20	Система управления автосервисом	Car, Customer, ServiceOrder, Mechanic

Тематики 21–30

№	Предметная тематика	Примеры сущностей
21	Система музейного каталога	Exhibit, Author, Collection, Location
22	Электронный архив документов	Document, Folder, Author, Tag
23	Система управления коллекцией фильмов	Movie, Director, Genre, Rating
24	Система управления музыкальной коллекцией	Track, Album, Artist, Genre
25	Платформа рецептов и кулинарных коллекций	Recipe, Ingredient, Category, User
26	Система заказа еды	Restaurant, Dish, Customer, Order
27	Система управления доставкой	Order, Customer, Courier, Route
28	Система общественного транспорта	Route, Stop, Vehicle, Schedule
29	Система учёта спортивных соревнований	Competition, Team, Participant, Result
30	Система управления волонтёрскими мероприятиями	Volunteer, Event, Task, Organization

Своя тема

Список из 30 — ориентир, не клетка.

- Можно предложить свою тему и **согласовать** её с преподавателем, если хотите сделать что-то сверх предложенного перечня.
- Для некоторых студентов по итогам предыдущего семестра своя тема — **обязательное требование** преподавателя.
- В этом случае не берите строку из таблицы «по умолчанию»: сначала разговор с преподавателем, затем Proposal.

Три пути проектной деятельности

Backend и Web обязательны на всех уровнях. Desktop и Mobile добавляются по уровню и не заменяют общий минимум.

Уровень	Backend	Web-клиент	Desktop	Mobile
Базовый	Java + Spring Boot + REST + PostgreSQL	TypeScript + React	—	—
Стандартный	Java + Spring Boot + REST + PostgreSQL	TypeScript + React	Java Swing	—
Продвинутый	Java + Spring Boot + REST + PostgreSQL	TypeScript + React	Java Swing	Android / Kotlin

Базовый путь: Backend + Web. Стандартный добавляет Desktop, продвинутый — Desktop и Mobile. Web-клиент использует TypeScript, HTML/CSS и React.

Положительная оценка: предметная область и Java

Планка семестра, а не объём реализации первой пары.

Предметная область

- 5–7 взаимосвязанных сущностей;
- CRUD для ключевой сущности;
- поиск / фильтрация;
- минимум два предметных бизнес-правила.

Java-модель

- объектная модель: интерфейсы и generics;
- Collections и Stream API;
- обработка исключений и regex-валидация;
- минимум два обоснованно применённых паттерна;
- события / Observer;
- сценарий работы с XML.

Положительная оценка: стек

Базовый путь закрывает минимум. Стандартный добавляет Swing, продвинутый — Android/Kotlin.

Обязательно

- REST API на Spring Boot;
- PostgreSQL;
- unit- и интеграционные тесты;
- logging;
- Docker и CI;
- web-клиент на TypeScript, HTML/CSS и React.

Сверх минимума

- Java Swing — стандартный путь;
- Android / Kotlin — продвинутый;
- Web обязателен на всех путях и не заменяется desktop- или mobile-клиентом.

Первичный Project Proposal

Короткий документ `PROPOSAL.md` в репозитории. README содержит краткое описание и ссылку на него.

1. **Тема:** номер и название из списка **или** согласованная своя.
2. **Проблема:** 1–2 абзаца, что ломается сегодня без системы.
3. **Аудитория:** кто пользователь, в каком контексте.
4. **Сущности:** ориентир 5–7 взаимосвязанных; можно опереться на таблицу.
5. **Путь:** базовый / стандартный / продвинутый; указать обязательный для него стек.
6. **Общий стек:** Spring Boot, REST, PostgreSQL, unit- и интеграционные тесты, logging, Docker, CI, TypeScript/HTML/CSS/React.
7. **Scope v1:** что входит и что сознательно откладывается.

Образец: тематика 1

Система управления задачами и проектами

Проблема. В студенческой команде договорённости живут в чате: неясно, кто владелец задачи и что должно быть готово к дедлайну.

Аудитория. Команды 3–6 человек без отдельного менеджера.

v1

Project, Task, User, Comment. Статусы задачи, комментарии. Путь: базовый; Backend + Web.

Не в v1

Биллинг, диаграмма Ганта. Мобильное — опциональный плюс, не обязанность v1.

Дополнительные материалы

Linux (WSL)

Дополнительно, ШАД. Linux-окружение в Windows, если основной ОС нет.

Ссылка

disk.yandex.ru/i/K3vdvIM96NCBNg

Не обязательный минимум этой лекции: достаточно терминала, Git и `./gradlew`.

QR-код на материал

Linux (практикум)

Дополнительно, ШАД. Практикум по Linux: терминал, файлы, процессы.

Ссылка

disk.yandex.ru/i/wtrav6JS7W5IGw

Пригодится, если командная строка ещё непривычна.
Ядро лекции — Git и Gradle.

QR-код на материал

Docker

Дополнительно, ШАД. Контейнеры: позже — единообразное окружение и поставка.

Ссылка

`disk.yandex.ru/i/X6CUFHpGe_ZAhg`

Сейчас можно отложить. К проекту Docker не требуется на этой неделе.

QR-код на материал

Итог

- Проект начинается с жизненного цикла: проблема и аудитория важнее кода.
- Инструменты: JDK 21, IDEA, **Gradle** (wrapper), Git.
- Лабораторные — Manytask2; проект — отдельный репозиторий.
- Положительная оценка — базовый стек (Spring Boot, PostgreSQL, React); Swing и Android — надстройка.
- AI ускоряет черновик; **результат ваш**.

Планируемый результат

Git, GitHub/GitLab, Gradle, IDEA и первичный Project Proposal.

Вопросы?