Технологии программирования

Системы сборки


Что значит: «код запускается не только в IDE»?

Вопрос 1: Из чего состоит приложение?

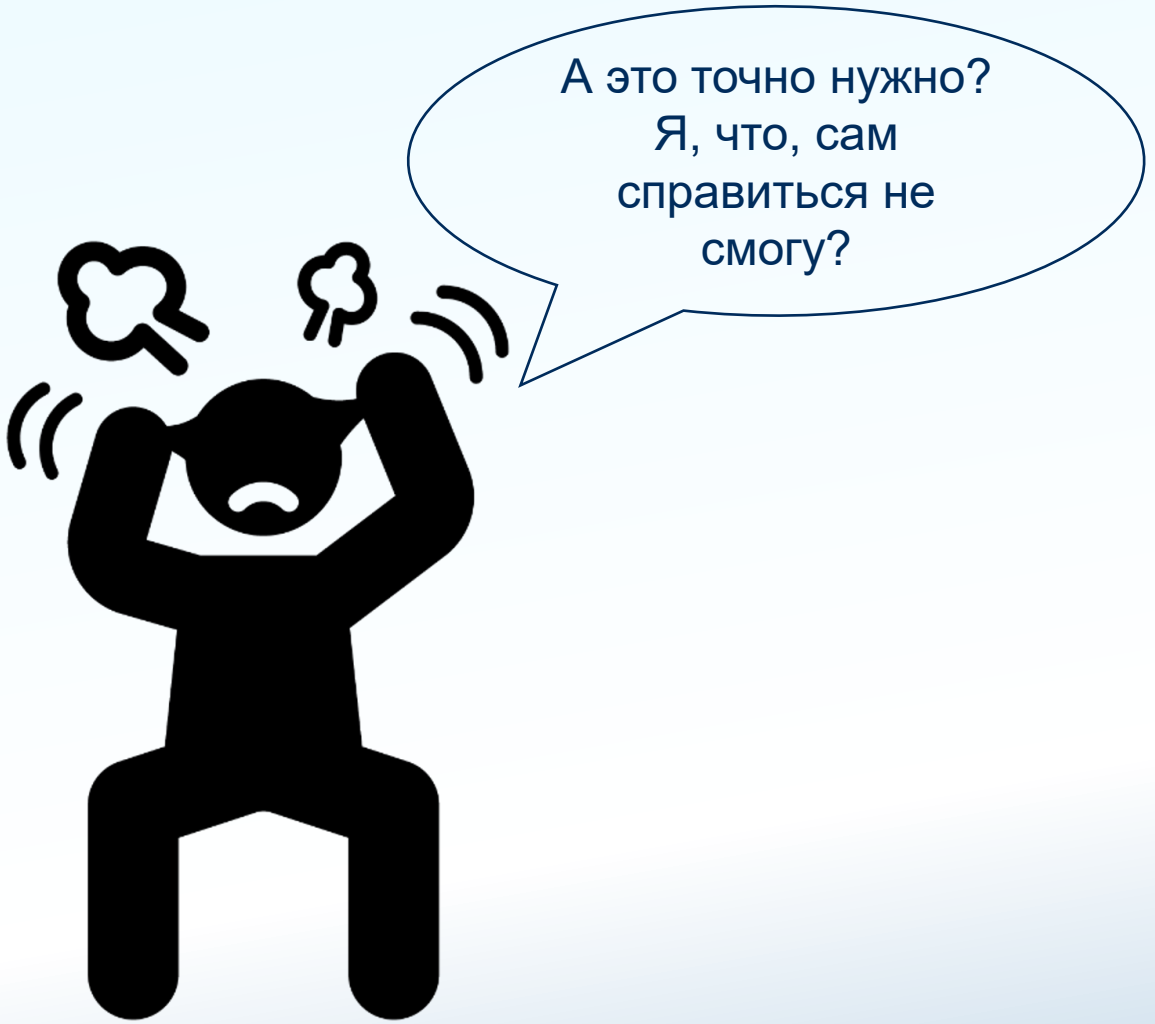
Вопрос 2: Как дать возможность использовать приложение?

Инструменты сборки и управления зависимостями

Что это такое?

ПО, обеспечивающее автоматизацию сборки проекта:

- компилирует код
- запускает тесты и определяет процент покрытия кода
- выполняет статический анализ
- собирает исполняемые файлы из исходного кода
- управляет зависимостями проекта: скачивает, добавляет к собранному проекту для его работоспособности.




Apache Maven Project
http://maven.apache.org/

Что это такое?

Система сборки и управления зависимостями со всей функциональностью подобного ПО.

Maven основывается на объектной модели проекта (POM). Кроме файлов, содержащих код, и необходимых библиотеках также в эту модель входят файлы конфигурации, информация о разработчиках, дефектах, организации-разработчике, лицензиях и т.д.

Зависимость проекта

Зависимость – это какая-либо библиотека, которая необходима данному проекту для корректной работы. Библиотека представляет из себя, например: обычный Java-проект, который собрали и распространили с помощью Maven.

Идентификатор состоит из 3 частей:

- groupId
- artifactId
- version

Самый известный для вас на данный момент пример – это org.junit.jupiter:junit-jupiter-api:5.9.2.

Область видимости зависимости

Каждую зависимость можно подключить к конкретному этапу сборки проекта.

Например, можно подключить зависимость так, чтобы она существовала в проекте только на фазе тестирования, самый яркий пример такой зависимости все тот же org.junit.jupiter:junit-jupiter-api:5.9.2.

То есть, данная библиотека будет находится в составе проекта только на этапе тестирования, а после того, как цикл прошел фазу тестирования будет исключена из проекта.

В Maven есть 6 областей видимости: compile, provided, runtime, test, system, import.

Область видимости зависимости

compile – область видимости по умолчанию;

provided – аналогично типу compile, но во время выполнения зависимость подставляет JDK;

runtime – указывает, что зависимость не требуется во время компиляции, и будет подставлена именно во время выполнения;

test – указывает, что зависимость требуется только во время компиляции и выполнения;

system – устарела;

import – подставляет зависимости, указанные в отдельном блоке dependencyManagement в POM.

«Жизненный цикл сборки»

Maven основан на концепции жизненного цикла сборки. Пользователю нужно указать одну из фаз жизненного цикла, а Maven на основе конфигурации POM сам сделает все, что нужно и доведет проект до необходимого состояния.

Цикл Maven состоит из следующих этапов: clean, validate, compile, test, package, verify, install, site, deploy.

Которые на самом деле делятся на три отдельных жизненных цикла Maven: clean, default, site.

Цикл по умолчанию

validate – проверить доступность всей необходимой информации для сборки проекта;

compile – скомпилировать исходный код;

test-compile – скомпилировать исходный код тестов;

test – запустить unit-тесты;

package – упаковать скомпилированный исходный код в необходимый формат (war, jar и т.д.);

integration-test – развернуть пакет для запуска интеграционных тестов;

verify – проверить собранный пакет на работоспособность;

install – установить проект в локальный репозиторий;

deploy – копировать полностью проверенный и собранный проект в удаленный репозиторий.

Цикл очистки

pre-clean – выполнить процессы, необходимые до очистки проекта;

clean – удалить все файлы, предыдущей сборки;

post-clean – выполнить процессы, необходимые для завершения очистки проекта.

Цикл генерации сайта

pre-site – выполнить процессы, необходимые до создания сайта проекта;

site – генерировать сайт проекта;

post-site – выполнить процессы, необходимые для завершения создания сайта и подготовки к развертыванию сайта;

site-deploy – развернуть сайт на указанный веб-сервер.

Цели Maven

На самом деле, когда мы даем команду к приведению состояния проекта к какой-то фазе, выполняется не сама фаза, а конкретные цели, которые в ней заложены. То есть, исполняемыми «методами» являются именно цели.

Например, есть фаза Maven site, мы можем ее вызвать командой mvn site, но на самом деле выполняется цель этой фазы, которая называется так mvn site:site.

Фазы – это обертка, а для выполнения команд используются цели.

Посмотреть, какие цели закреплены за каждой фазой мы можем командой:

mvn help:describe -Dcmd=PHASENAME.

Плагины Maven

Плагины Maven – это вторая после зависимостей сущность данного инструмента. Если зависимость выступает статической библиотекой, которая просто присутствует или отсутствует в проекте, то плагин – это, грубо говоря, метод, который совершает какие-то действия.

На самом деле, фазы – это и есть плагины, просто вшитые в Maven по умолчанию. Также есть еще множество различных плагинов для различных целей. А еще вы можете написать собственный плагин, необходимый именно вашему проекту.

А, если быть совсем точными, то плагины – это группа целей, так как мы уже знаем, что за выполнение команд отвечают именно цели.

Пример

org.apache.maven.plugins:maven-jar-plugin:2.6

Данный плагин отвечает за генерацию документации в формате JavaDoc из исходного кода вашего проекта.

У плагинов ровно также есть groupId, artifactId, version.

Данный плагин отвечает за генерацию документации в формате JavaDoc из исходного кода вашего проекта.

POM

POM – это одновременно и объектная модель объекта, и файл pom.xml, в котором хранится метаинформация о проекте, информация о зависимостях и описаны необходимые плагины.

Ознакомиться с примером pom.xml можете самостоятельно:

https://drive.google.com/file/d/14maqANo6l4DXARTZiRsMBTkMHkR4ZndK/view?usp=sharing

Gradle

Что это такое?

Система сборки и управления зависимостями со всей функциональностью подобного ПО и открытым исходным кодом.

– помогает автоматизировать широкий спектр сценариев сборки ПО, используя встроенные функции, сторонние плагины или пользовательскую логику сборки;
– предоставляет высокоуровневый, декларативный и выразительный язык сборки, который упрощает чтение и написание логики сборки;
– обеспечивает надежные результаты, используя такие оптимизации, как инкрементальные сборки, кэширование сборок и параллельное выполнение;
– поддерживает Android, Java, Kotlin Multiplatform, Groovy, Scala, JavaScript и C/C++.

Gradle

Зависимость

Зависимость в терминологии Gradle – это: библиотека; проект, (многомодульный проект); локальный файл, (.jar) - необходимые данному проекту для корректной работы.

Идентификатор состоит из 3 частей:

- groupId
- artifactId
- version

```
dependencies {
    testImplementation(platform("org.junit:junit-bom:5.10.0"))
    testImplementation("org.junit.jupiter:junit-jupiter")
    compileOnly("org.projectlombok:lombok:1.18.36")
}
```

Области видимости зависимостей

implementation – зависимости, необходимые, как для компиляции, так и для выполнения;

api – открытый “implementation”;

compileOnly – зависимости, необходимые только для компиляции;

compileOnlyApi – открытый “compileOnly”;

runtimeOnly – зависимости, необходимые только для выполнения;

testImplementation – зависимости, необходимые для компиляции и выполнения тестов;

testCompileOnly – зависимости, необходимые только для компиляции тестов;

testRuntimeOnly – зависимости, необходимые только для выполнения тестов.

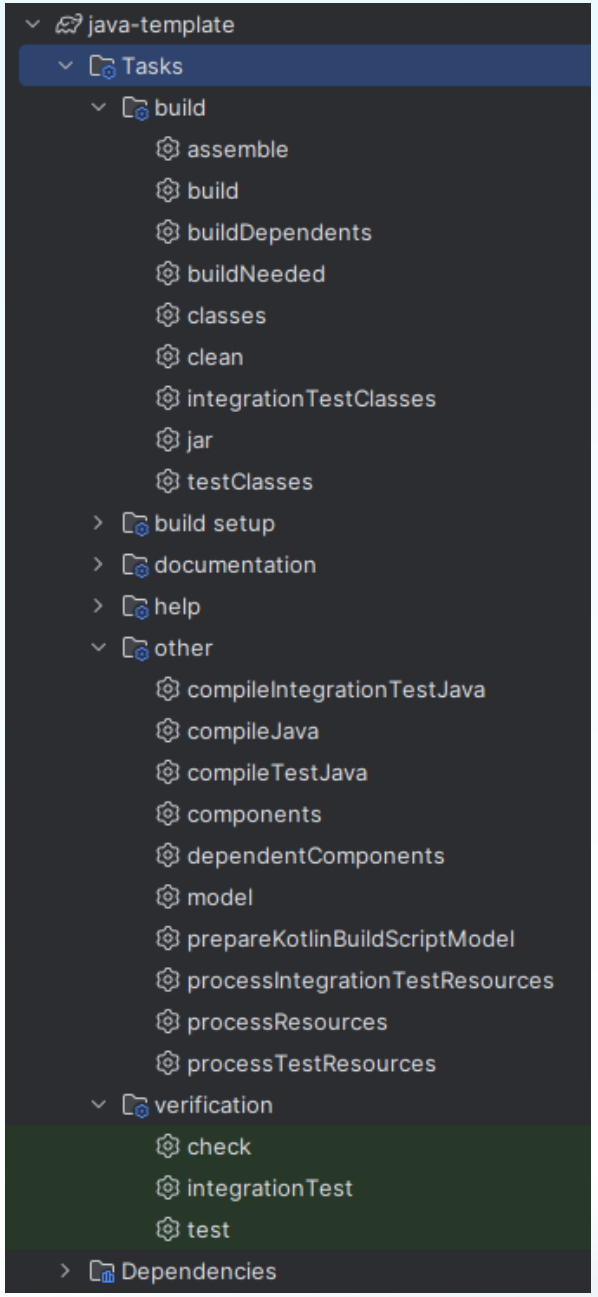
Gradle

Репозитории зависимостей

Gradle не имеет собственных репозиториев и в качестве источника зависимостей использует Maven и Ivy репозитории.

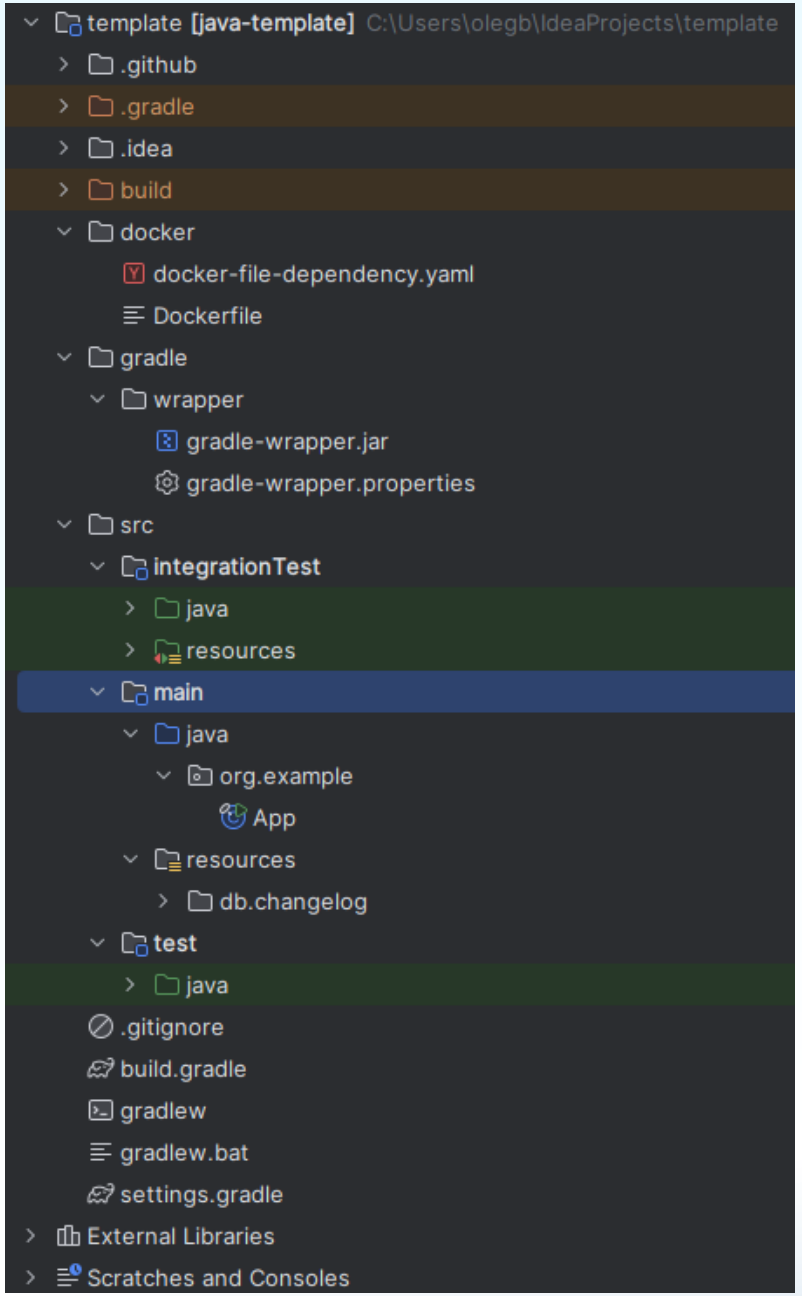
При этом интерфейс для работы с репозиториями не отличается на базовом уровне, более развёрнуто об отличиях параметров вы можете узнать по ссылкам MavenArtifactRepository и IvyArtifactRepository.

«Цикл» сборки Gradle



Gradle

Структура проекта



Gradle

Wrapper

Это скрипт, который запускает задачи Gradle с объявленной версией. Если объявленная версия не установлена, Wrapper устанавливает требуемую.

Преимущества:

– создание проектов с помощью Wrapper локально позволяет вам предварительно не устанавливать Gradle;
– у каждого участника команды и на конвейерах CI проект собирается под одной и той же версией Gradle;
– легкое обновление до новой версии Gradle путем изменения настроек Wrapper.

Конфигурация wrapper

```
#Sat Feb 07 12:09:30 MSK 2026
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-8.12-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

build.gradle

```
plugins {
    id 'java'
    id 'idea'
}

group = 'org.example'
version = '1.0-SNAPSHOT'

java {
    toolchain { JavaToolchainSpec it ->
        languageVersion = JavaLanguageVersion.of(21)
    }
}

repositories {
    mavenCentral()
}

dependencies {
    testImplementation platform('org.junit:junit-bom:5.10.0')
    testImplementation 'org.junit.jupiter:junit-jupiter'
    testImplementation 'org.testcontainers:junit-jupiter:1.19.7'
    testImplementation 'org.testcontainers:postgresql:1.19.7'
    testImplementation 'org.postgresql:postgresql:42.7.4'
    testRuntimeOnly 'org.slf4j:slf4j-simple:2.0.13'
}
```

Gradle

Gradle

- производительность, (инкрементальные сборки, кэш);
- гибкость;
- более глубокое управление зависимостями;
- поддержка нескольких языков программирования;
- доменно-специфический язык программирования: DSL.

Maven

- проще в освоении;
- набор поведений и настроек по умолчанию.

Если не лень…

Доклад Евгения Борисова. Power of Gradle.

https://github.com/mihailaleksseev/conferenceAbstracts/blob/main/%2B%2B%202013%20Gradle%20%7C%20%D0%95%D0%B2%D0%B3%D0%B5%D0%BD%D0%B8%D0%B8%CC%86%20%D0%91%D0%BE%D1%80%D0%B8%D1%81%D0%BE%D0%B2%20%E2%80%94%20Power%20of%20Gradle.pdf