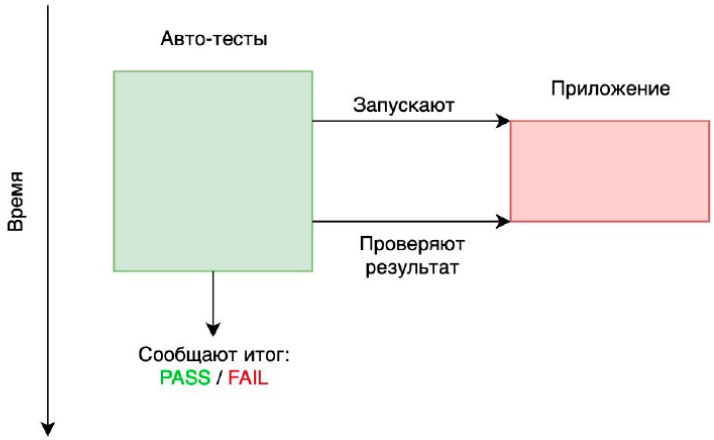
AUTOTESTS & MAVEN

АВТОТЕСТЫ

Фактически, мы просто по определённым правилам напишем код на Java (автотесты), который будет вызывать другой код на Java (наше приложение) с определёнными параметрами и проверять возвращаемый результат.



TESTABILITY

Testability — степень, с которой система пригодна для тестирования (определение нечёткое и не определяет численной характеристики).

Наше приложение обладает низкой Testability (нужно руками менять значения в коде и запускать).

Q: как же повысить Testability нашего приложения?
A: посмотрим на ваш предыдущий опыт в ручном тестировании.

ООП

ООП (объектно-ориентированное программирование) — это подход к моделированию реального мира в программировании, когда мы всё **описываем в виде объектов**, обладающих **свойствами** и **определёнными функциями**.

СОСТОЯНИЕ

Текущее значение всех свойств объекта называется **состоянием**.

В зависимости от состояния может меняться поведение объекта:

- если система неисправна, то методы не перемещают лифт;
- если перевозимая масса выше, чем допустимая, — то лифт остаётся с открытыми дверями и никуда не едет;
- если лифт едет вниз, то он не реагирует на вызовы верхних этажей;

КЛАССЫ И ОБЪЕКТЫ

Для того, чтобы получить объект, с которым можно работать, нам нужно сделать два шага:

1. Описать этот объект (какие свойства и методы у него будут).
2. Создать объект из этого описания.

Все классы должны называться с большой буквы: `BonusService`, а не `bonusService`.

МЕТОДЫ

33

Методы — это функции*, которые будут у созданного объекта.
У метода есть:

1. **Имя** («Позвонить« и т.д. — но на английском).
2. **Входные параметры** (amount типа `long`, `registered` типа `boolean`).
3. **Тип возвращаемого результата** (в нашем случае — `long`).

СОЗДАНИЕ ОБЪЕКТА

37

Q: что такое `new BonusService()`?
A: пока нам нужно запомнить, что это создание объекта из класса (описания объекта).

Представьте, что вы запускаете Калькулятор из панели Пуск.

Каждый раз, когда вы нажимаете на пункт Калькулятор,

создаётся **новое окно**

Калькулятора.
Каждое новое окно — это отдельныйобъект.
Тоже самое происходит и у нас: `new BonusService()` — это создание объекта (только не калькулятора, а нашего, который описан в классе `BonusService`).

Ключевые клавиатурные сокращения*:

1. Ctrl + F8 — установка/снятие точки остановки.
2. Shift + F9 — запуск подотладчиком.
3. F8 — исполнение следующей строки (без захода в метод) в режиме отладки.
4. F7 — исполнение следующей строки (с заходом в метод) в режиме отладки.

БИБЛИОТЕКА VS FRAMEWORK

Библиотека — это код (набор классов в Java), который вы подключаете к своему проекту и используете так, как хотите.
Фреймворк — это библиотека, определяющая правила, по которым вы должны писать код (в противном случае, вы не получите желаемого).

Maven и Gradle часто называют Project Management Tool —
инструмент управления проектом.

Умеет:

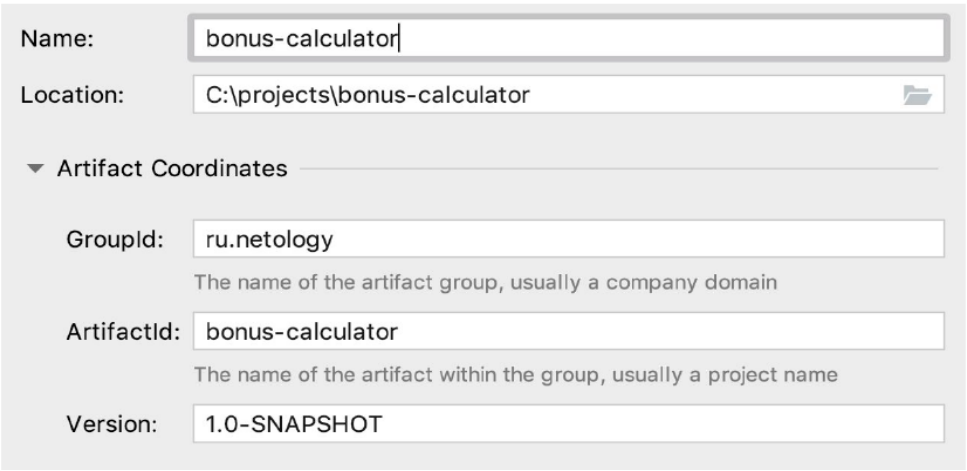
4. настраивать ваше приложение;
5. управлять зависимостями;

- компилировать/тестировать/генерировать документацию;
- собирать исполняемые файлы;
- и много чего другого.

Широкий спектр возможностей обеспечивается благодаря системе плагинов— отдельных модулей, которые можно подключать к Maven для выполнения
специальных задач.

ARTIFACT COORDINATES

Нужно обязательно заполнить название проекта и открыть панельку Artifact Coordinates:



Maven исповедует концепцию **Convention over Configuration** — вместо того, чтобы всё настраивать, используй общепринятыесоглашения.

Автотест — это просто метод:

1. На месте возвращаемого типа стоит `void` — это значит метод ничего не возвращает (не нужен `return`).
2. Скобки для параметров пусты — значит метод не принимает никаких входных данных.
3. Над методом стоит конструкция `@org.junit.jupiter.api.Test` — это аннотация.

Как мы говорили, метод не может существовать сам по себе, поэтому он написан в классе.

1. Когда мы запускаем тесты, запускается JUnit (как раньшезапускался наш `Main`).
2. JUnit ищет все классы в каталоге `src/test/java` над методами которых стоит `@Test`.
3. Для каждого метода (с `@Test`) — создаёт объект из класса и вызывает метод (это и есть тест).
4. Для каждого вызова — проверяет результат.

JUnit использует специальные методы (`assert`'ы) для того, чтобы определить, тест прошёл или нет.

Assert'ов достаточно много, но сегодня нам хватит одного: `assertEquals`.

`assertEquals` — это метод, который принимает два параметра, сравнивает их и, если они не равны, «роняет»тест.

Это очень важно при написании тестов: **обязательно проверяйте при написании, что ваши тесты падают при неправильных значениях**.

Потому что мы встречали очень много тестов, которые всегда «зелёные», т.к.
ничего не проверяют, либо проверяют не то.

Очень важно, чтобы имена тестовых методов (над которыми стоит `@Test`) были говорящими.
Когда у вас будет много тестов, это поможет четко понимать, что именно "упало" в тестируемой вамипрограмме.
Есть много разных подходов к именованию, например:

1. `whenRegisteredAndBonusUnderLimit_thenBonusIsNotLimited`
2. `testBonusIsNotLimitedIfRegisteredAndBonusUnderLimit`

   Идеального решения нет — вы должны использовать ту схему, которая будет принята в вашей команде.

Важно: когда вы нажимаете `Ctrl + Shift + F10` внимательно следите, где находится ваш курсор:

1. если внутри метода, то запустится только этот метод (один тест)
2. если внутри класса (но вне методов), то запустятся все методы внутри класса

`mvn clean test` сначала всё почистит, а затем запустит тесты.

Ключевые клавиатурные сокращения*:

1. `Ctrl + Shift + T` на имени класса — генерация теста.
2. `Shift + Shift` — глобальный поиск по командам и возможностям IDEA (и не только).
3. `Ctrl + Ctrl` — Run Anything (возможность запуск команд Maven).