



СИСТЕМА СБОРКИ MAVEN, УПРАВЛЕНИЕ ЗАВИСИМОСТЯМИ, АВТОТЕСТЫ НА JUNIT5



Оксана Мельникова



ОКСАНА МЕЛЬНИКОВА

Software testing engineer



ПЛАН ЗАНЯТИЯ

1. [Автотесты](#)
2. [Библиотеки](#)
3. [Управление зависимостями](#)
4. [Maven](#)
5. [JUnit Jupiter](#)
6. [Логи](#)
7. [Итоги](#)



АВТОТЕСТЫ

АВТОТЕСТЫ

На прошлой лекции мы выделили в нашем приложении класс BonusService:

```
public class BonusService {  
    public long calculate(long amount, boolean registered) {  
        int percent = registered ? 3 : 1;  
        long bonus = amount * percent / 100 / 100;  
        long limit = 500;  
        if (bonus > limit) {  
            bonus = limit;  
        }  
        return bonus;  
    }  
}
```

АВТОТЕСТЫ

Кроме того, мы даже написали псевдо-тесты, которые выводят значения на экран:

```
public static void main(String[] args) {  
    ...  
    long bonusBelowLimitForRegistered = service.calculate(1000_60, true);  
    System.out.println(bonusBelowLimitForRegistered);  
    ...  
}
```

Но нам приходилось проверять эти значения глазами самостоятельно.

АВТОТЕСТЫ

А что если сделать вот так?

```
public static void main(String[] args) {  
    BonusService service = new BonusService();  
  
    // подготавливаем данные:  
    long amount = 1000_60;  
    boolean registered = true;  
    long expected = 30;  
  
    // вызываем целевой метод:  
    long actual = service.calculate(amount, registered);  
  
    // производим проверку (сравниваем ожидаемый и фактический):  
    // если true – то PASS  
    // если false – то FAIL  
    boolean passed = expected == actual;  
  
    // выводим результат  
    System.out.println(passed);  
}
```



АВТОТЕСТЫ

Почти хорошо, но есть несколько моментов:

1. Если тест "упадёт" — непонятно, какой упал (у нас же их будет много).
2. Если тест "упадёт" — непонятно, с какими данными он упал.

Это, конечно, можно всё поправить. Но посмотрим сначала на уже существующие решения.



БИБЛИОТЕКИ



БИБЛИОТЕКИ

В мире Java (как и других языков программирования) существуют библиотеки.

Библиотека — это уже готовый, написанный кем-то код, который мы можем подключить к своему проекту и использовать.

Библиотеки не входят в состав JDK и распространяются отдельно, т.к. принадлежат сообществу, компаниям или другим лицам.



БИБЛИОТЕКИ

Аналог библиотек для обычных пользователей — это программы.

Например, вы хотите редактировать фотографии. Для этого вам нужно скачать и установить соответствующую программу, например, Adobe Photoshop.

То же самое в программировании — мы можем скачать и установить библиотеку, позволяющую проще тестировать.

JUNIT, TESTNG

В Java самыми популярными являются:

- [JUnit Jupiter \(JUnit5\);](#)
- [JUnit \(JUnit4\);](#)
- [TestNG.](#)

И именуют они себя не библиотеками, а фреймворками.



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).

Простой пример: вы можете купить для передвижения **самокат (библиотеку)**, а можете пользоваться услугами **метро (фреймворк)**.

При этом в метро определено: в какой вход вы должны входить, по какому эскалатору спускаться, маршруты и интервалы движения поездов.

Самокат же разрешает вам ездить в любую сторону и в любое время без ограничений.



JUNIT JUPITER

Мы начнём нашу работу с JUnit Jupiter (JUnit5)*, а с остальными двумя познакомимся позже.

Для этого нам надо перейти на официальный сайт, разобраться с тем, как скачивать, устанавливать и так для каждого проекта...

СТОП, но это же ручная работа?

Неужели программисты до сих пор не придумали аналога Google Play или AppStore: заходишь, нажимаешь кнопку, оно само скачивается, устанавливается и можно пользоваться?

Примечание*: пока для простоты мы будем использовать JUnit5 и JUnit Jupiter как синонимы, но на следующем курсе разберёмся в отличиях.

РЕПОЗИТОРИИ

В мире Java хранилища кода (в том числе библиотек) называются репозиториями. А хранящиеся в них сущности — артефактами.

Самый популярный репозиторий — Maven Central. А поисковый веб-интерфейс к нему (и не только) — mvnrepository.com:

Found 1924 results

Sort: **relevance** | popular | newest



1. JUnit Jupiter API

org.junit.jupiter » junit-jupiter-api

Module "junit-jupiter-api" of JUnit 5.

Last Release on Jan 6, 2020

4,798 usages

EPL



УПРАВЛЕНИЕ ЗАВИСИМОСТЯМИ



ЗАВИСИМОСТЬ

Если наш проект зависит от какой-либо библиотеки (или фреймворка) — это называется **зависимость**.

В мире Java существует два самых популярных инструмента управления зависимостями*:

1. Maven.
2. Gradle.

Примечание:* есть и другие, но мы будем рассматривать эти два.



PROJECT MANAGEMENT

Maven и Gradle умеют самостоятельно скачивать и устанавливать нужные зависимости (и зависимости зависимостей), а также компилировать ваш проект, запускать автотесты и т.д.

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

Q: т.е. нам нужно установить Maven/Gradle, а уже они скачают зависимости?

A: нет, всё, что нужно, уже идёт в составе IDEA (хотя вы можете скачать и установить отдельно — об этом позже).



MAVEN



MAVEN

[Maven](#) — Project Management Tool.

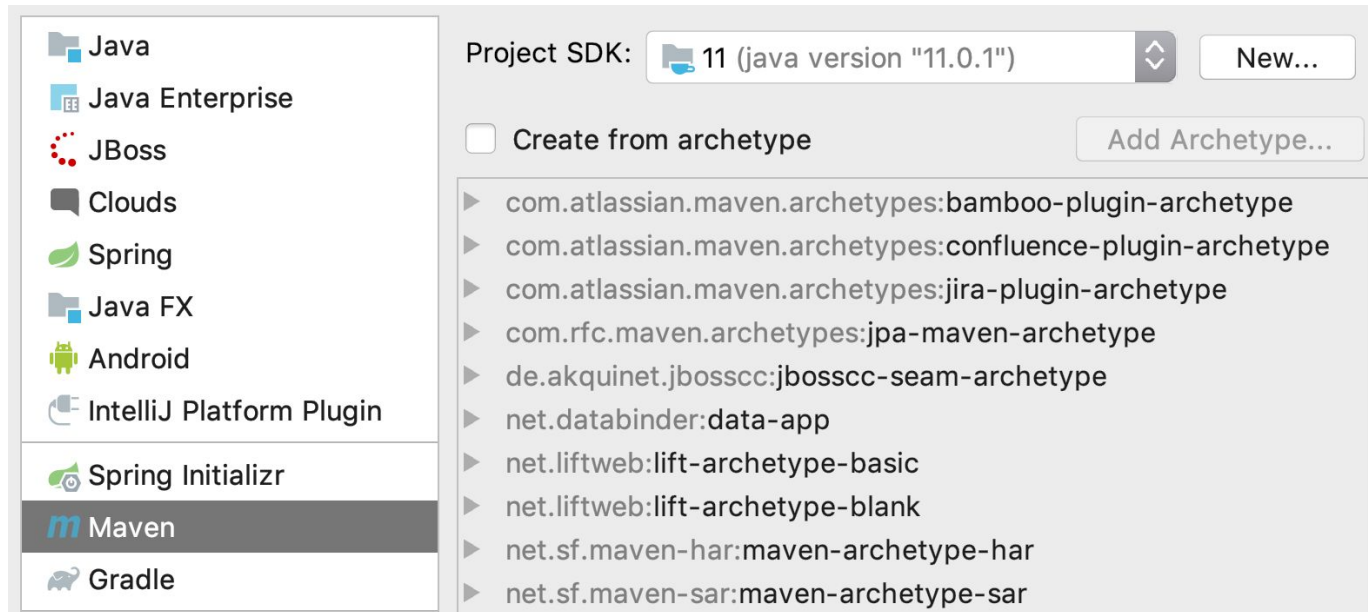
Умеет:

- настраивать ваше приложение;
- 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.

СОЗДАНИЕ ПРОЕКТА

Проще всего использовать конструктор IDEA для создания Maven-проектов:




Далее нажимаете Next (в правой панельке ничего выбирать не нужно).

ARTIFACT COORDINATES

Нужно обязательно заполнить название проекта и открыть панельку Artifact Coordinates:

Name:

Location: 

▼ Artifact Coordinates

GroupId:
The name of the artifact group, usually a company domain

ArtifactId:
The name of the artifact within the group, usually a project name

Version:

ARTIFACT COORDINATES

У каждого создаваемого приложения/библиотеки есть уникальный набор координат:

- `GroupId` — чаще всего reverse domain name* сайта автора (организации);
- `ArtifactId` — имя проекта (или выходного файла);
- `version` — текущая версия.

По этому набору (координатам) его (приложение/библиотеку) можно будет найти в репозиториях наподобие Maven Central (если его публиковать).

В рамках курса мы будем использовать `GroupId ru.netology`, а имя проекта — будет зависеть от проекта, например, `bonus-calculator`. Версия пока будет по умолчанию.

Примечание:* reverse domain name — это имя домена наоборот. Т.е. обычное — `netology.ru`, а reverse — `ru.netology`.

pom.xml

IDEA сгенерирует файл pom.xml (Project Object Model), описывающий наш проект:



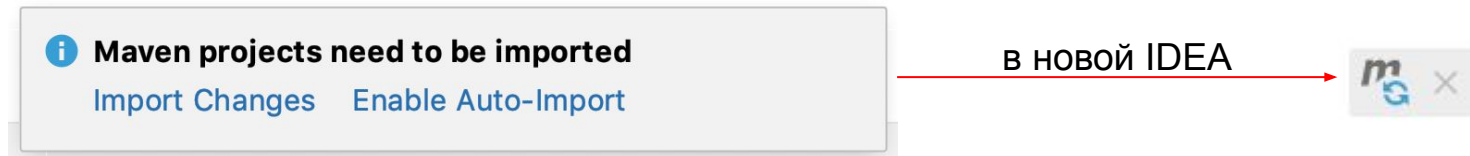
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         <modelVersion>4.0.0</modelVersion>
6
7         <groupId>ru.netology</groupId>
8         <artifactId>bonus-calculator</artifactId>
9         <version>1.0-SNAPSHOT</version>
10
11
12 </project>
13
```

Именно в этом файле будут описаны все настройки нашего проекта (зависимости и т.д.).

AUTO IMPORT

Теперь IDEA будет постоянно синхронизировать наш проект с файлом `pom.xml` (если мы будем редактировать его руками).

Поэтому у вас будет появляться вот такой диалог:



Мы не рекомендуем включать `Enable Auto-Import` (автоматическую синхронизацию), поскольку она не всегда срабатывает.



CONVENTION OVER CONFIGURATION

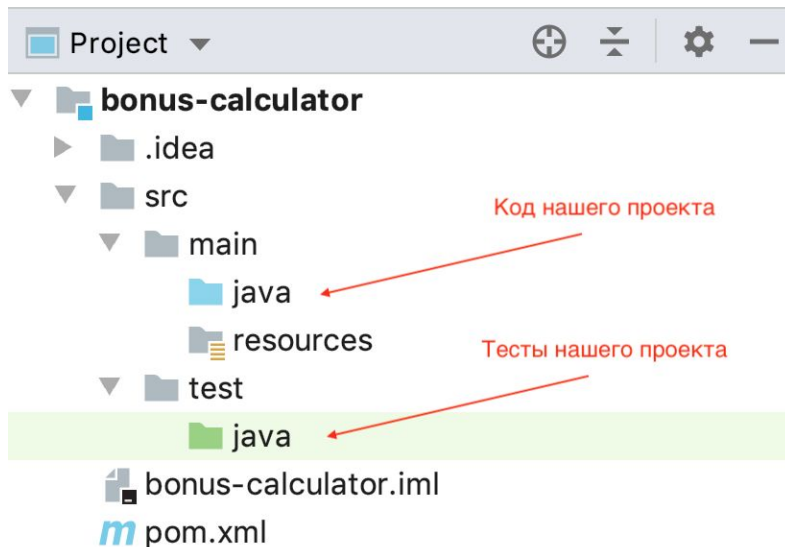
Во многих инструментах, которые мы будем проходить — достаточно много ключевых «философских идей», на базе которых всё строится.

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

Это относится к структуре каталогов в проекте, компиляции, запуску автотестов и т.д.

СТРУКТУРА КАТАЛОГОВ

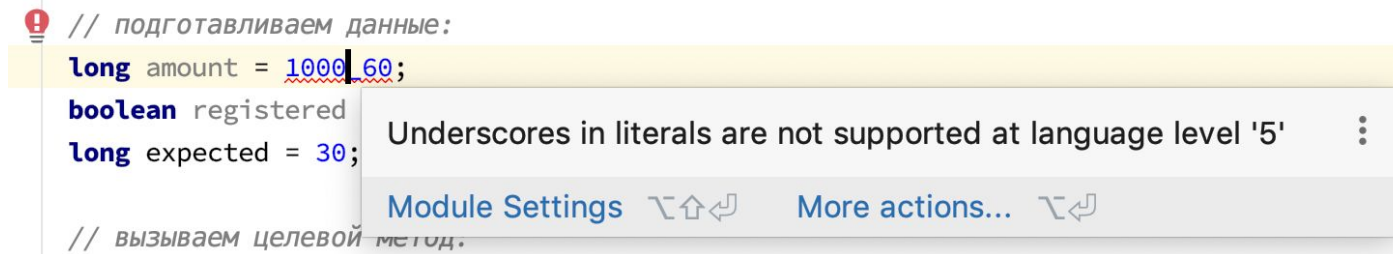
Большинство проектов используют структуру каталогов по умолчанию:



Поместим исходные коды (файлы с расширением `.java`) в каталог `src/main/java` и попробуем запустить.

IDEA поддерживает `Ctrl + V` прямо в панельке `Project`.

JAVA 5

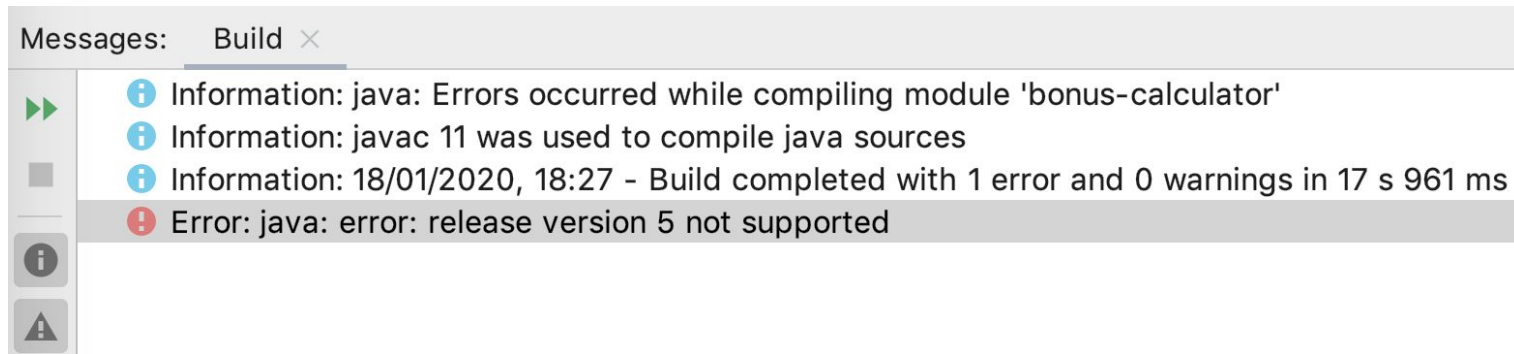


Q: IDEA говорит о каком-то «language level '5'». Что это?

A: по умолчанию Maven (и IDEA на основании этого) считает, что вы работаете в режиме совместимости с Java 5. А там конструкции с подчёркиванием ещё не внедрили в язык.

JAVA 5

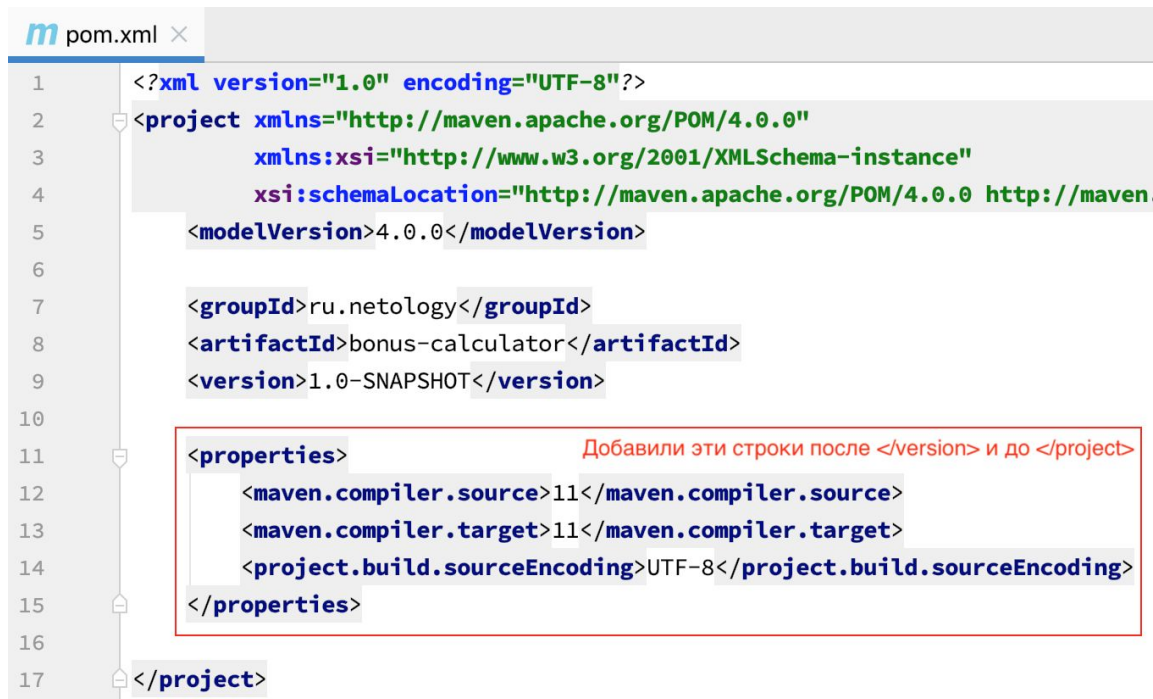
Кроме того, попытка компиляции приведёт к следующему:



Поэтому нам нужно воспользоваться принципом «Convention over Configuration» и сконфигурировать эту часть, поскольку она нас не устраивает.

pom.xml

Для этого отредактируем файл pom.xml следующим образом:



The screenshot shows a code editor with the file 'pom.xml' open. The code is as follows:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
5         <modelVersion>4.0.0</modelVersion>
6
7     <groupId>ru.netology</groupId>
8     <artifactId>bonus-calculator</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <properties>
12         <maven.compiler.source>11</maven.compiler.source>
13         <maven.compiler.target>11</maven.compiler.target>
14         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15     </properties>
16
17 </project>
```

A red rectangular box highlights the content between lines 11 and 15. A red annotation above the box reads: "Добавили эти строки после </version> и до </project>".

`pom.xml`

Если идти построчно, то мы буквально говорим Maven:

1. Исходные коды написаны на Java 11.
2. Компилируем в байткод, совместимый с Java 11.
3. Кодировка исходников — UTF-8.

После этого не забудьте импортировать изменения (предупреждения IDEA об ошибках должны пропасть).

Настоятельно рекомендуем вам ознакомиться с форматом файла `xml` и правилами его редактирования [в материалах к лекциям](#).

MAVEN PANEL

Если вдруг у вас куда-то пропал диалог об импорте изменений, вы можете нажать два раза **Shift** и ввести **reimport**:

🔍 reimport|

Actions

↻ Reimport All Gradle Projects

↻ **Reimport** All Maven Projects


pom.xml

Q: а нельзя ли это автоматизировать?

A: можно, но студенты слишком часто потом натыкаются на эту проблему и не понимают, что делать.

Поэтому мы решили, что вы должны будете запомнить эти строки и вводить их руками, чтобы с вами такого не произошло (либо брать из предыдущего проекта).

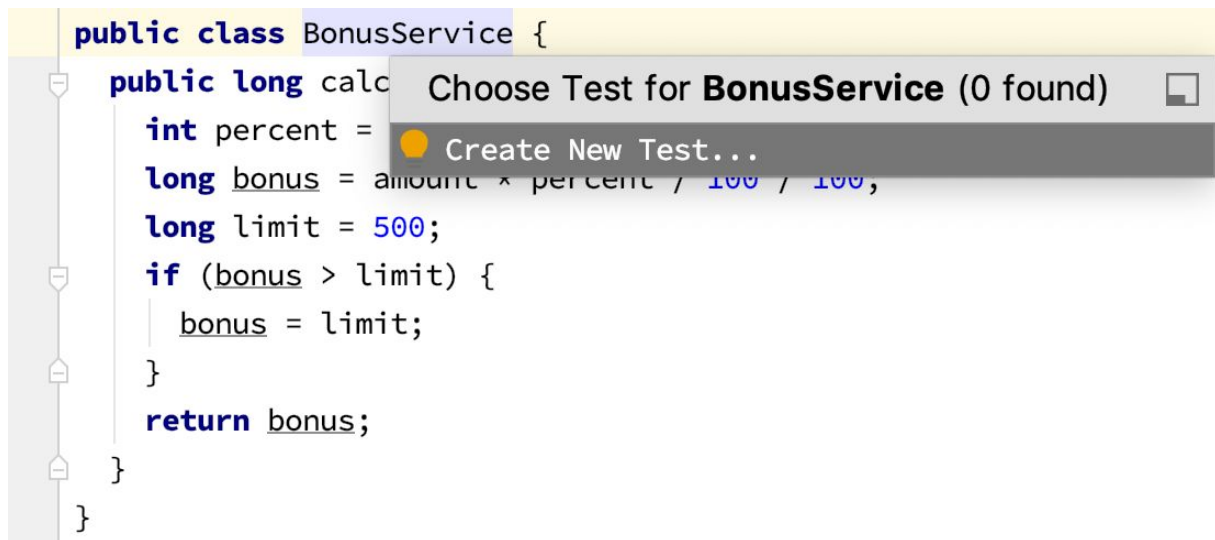


JUNIT JUPITER

АВТОТЕСТЫ

Наконец у нас всё готово для создания наших автотестов.


Переходим в класс `BonusService` ставим курсор на имени класса, нажимаем `Ctrl + Shift + T` и выбираем `Create New Test`:



АВТОТЕСТЫ

Ставим в генераторе флажки:

Testing library: JUnit5

 JUnit5 library not found in the module 1 Fix

Class name:


Superclass: ▼ ...

Destination package: ▼ ...

Generate:

- ☐ setUp/@Before
- ☐ tearDown/@After

Generate test methods for: ☐ Show inherited methods

2		Member
<input checked="" type="checkbox"/>		<code>calculate(amount:long, registered:boolean):long</code>

IDEA сама добавила в наш файл `pom.xml` следующий блок:

```
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.4.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Это означает, что с Maven Central будет выкачан необходимый артефакт (и всё, что нужно для его работы) определённой версии:

- ▼  External Libraries
 - ▶  < 11 >
 - ▶  Maven: org.apiguardian:apiguardian-api:1.0.0
 - ▶  Maven: org.junit.jupiter:junit-jupiter:5.4.2
 - ▶  Maven: org.junit.jupiter:junit-jupiter-api:5.4.2
 - ▶  Maven: org.junit.jupiter:junit-jupiter-engine:5.4.2
 - ▶  Maven: org.junit.jupiter:junit-jupiter-params:5.4.2
 - ▶  Maven: org.junit.platform:junit-platform-commons:1.4.2
 - ▶  Maven: org.junit.platform:junit-platform-engine:1.4.2
 - ▶  Maven: org.opentest4j:opentest4j:1.1.1



ЗАВИСИМОСТИ

Q: но как я узнаю, какие зависимости мне нужны?

A: в данном случае мы специально использовали конструктор IDEA, чтобы она сама правильно всё установила.

В реальной жизни:



- из нашего курса;
- статьи в интернете, блоги, подкасты;
- анализ Maven Central по категориям и чтение документации на конкретные библиотеки.

ВЕРСИИ


В составе IDEA не всегда идёт привязка к самым свежим версиям, поэтому проверяйте последние свежие на <https://mvnrepository.com>





















```
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.6.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

IDEA сама сгенерировала заготовку для автотеста:

```
3   class BonusServiceTest {  
4  
5      @org.junit.jupiter.api.Test  
6   void calculate() {  
7      }  
8  }
```

Их можно запустить с помощью стрелок или **Ctrl + Shift + F10**:

Run:  BonusServiceTest x

												
	  Test Results 68 ms											
	  BonusServiceTest 68 ms											
	 calculate() 68 ms											

АВТОТЕСТ

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.

Посмотрим, как это работает.

КАК ЭТО РАБОТАЕТ

1. Когда мы запускаем тесты, запускается JUnit (как раньше запускался 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.

Пока проверять нечего — тест пустой. Попробуем сделать первую проверку.

ПЕРВЫЙ АВТОТЕСТ

Просто перенесём наш код из `Main` и заменим всего пару строк:

```
@org.junit.jupiter.api.Test
void calculate() {
    BonusService service = new BonusService();

    // подготавливаем данные:
    long amount = 1000_60;
    boolean registered = true;
    long expected = 30;

    // вызываем целевой метод:
    long actual = service.calculate(amount, registered);

    // производим проверку (сравниваем ожидаемый и фактический):
    assertEquals(expected, actual);
}
```

Всё, что поменялось — вместо `System.out.println` добавили `assertEquals`.



ASSERTIONS

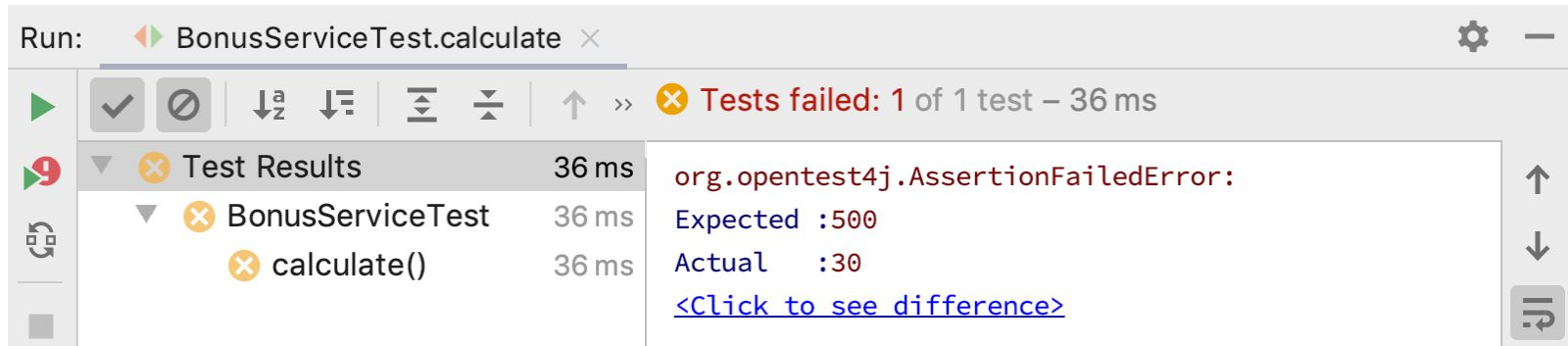
JUnit использует специальные методы (`assert'`ы) для того, чтобы определить, тест прошёл или нет.

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.

РОНЯЕМ ТЕСТ

Заменяем `expected` на заведомо неправильное значение (500) и попробуем заново запустить 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.

ИМЕНОВАНИЕ ТЕСТОВ

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 вами программе.

Есть много разных подходов к именованию, например:

- `whenRegisteredAndBonusUnderLimit_thenBonusIsNotLimited`
- `testBonusIsNotLimitedIfRegisteredAndBonusUnderLimit`
- и т.д.

ИМЕНОВАНИЕ ТЕСТОВ

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

Именованье — большая проблема, т.к. сложно придумать действительно понятное и короткое имя. Поэтому не бойтесь длинных имён — всё приходит с опытом.

Мы с вами будем использовать простую схему:

`shouldCalculateForRegisteredAndUnderLimit`

- `should` — базовое имя
- `Calculate` — имя тестируемого метода (с большой буквы)
- `RegisteredAndUnderLimit` — базовые условия

ВТОРОЙ ТЕСТ

```
@org.junit.jupiter.api.Test
void shouldCalculateForRegisteredAndOverLimit() {
    BonusService service = new BonusService();

    // подготавливаем данные:
    long amount = 1_000_000_60;
    boolean registered = true;
    long expected = 500;

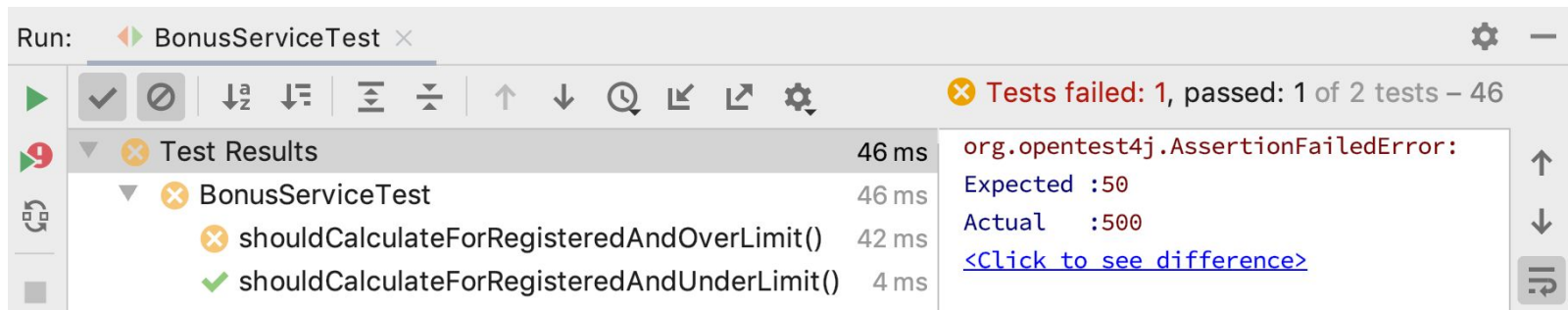
    // вызываем целевой метод:
    long actual = service.calculate(amount, registered);

    // производим проверку (сравниваем ожидаемый и фактический):
    assertEquals(expected, actual);
}
```

Как видите, ничего сложного: мы поменяли название и данные. Теперь у нас два теста.

РОНЯЕМ ТЕСТ

Если в новом тесте установим заведомо неправильный ответ (а вы должны помнить, что надо обязательно проверять, что тест падает при неверных данных):



The screenshot shows the 'Run' window of an IDE with the 'Test Results' tab selected. The window title is 'Run: BonusServiceTest'. The toolbar includes icons for running, passing, failing, and other test actions. The test results are as follows:

Test Name	Duration	Status
Test Results	46 ms	Failed
BonusServiceTest	46 ms	Failed
shouldCalculateForRegisteredAndOverLimit()	42 ms	Failed
shouldCalculateForRegisteredAndUnderLimit()	4 ms	Passed

On the right side of the window, the details for the failed test are shown:

```
org.opentest4j.AssertionFailedError:  
Expected :50  
Actual   :500  
<Click to see difference>
```

Видно, какой тест прошёл и видна общая статистика (1 из 2).

ЗАПУСК ТЕСТОВ

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

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

Q: а нельзя ли запустить сразу все? Вроде же говорили, что Maven это умеет.

ДОБРО ПОЖАЛОВАТЬ В РЕАЛЬНЫЙ МИР

Доступ к командам Maven можно получить с помощью диалога в IDEA:
нажмите два раза **Ctrl** и введите **mvn test***:

Run Anything

 mvn test

В случае Maven приучайтесь сразу пользоваться им из "командной строки", потому что это вам очень сильно поможет при настройке систем автотестирования в дальнейшем (у большинства из них не будет графического интерфейса).

ДОБРО ПОЖАЛОВАТЬ В РЕАЛЬНЫЙ МИР

Всё отлично (галочка зелёная), тесты проходят!

```
m bonus-calculator [test] ×
✓ bonus-calculator [test]: 10 s 807 ms

T E S T S

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.982 s
[INFO] Finished at: 2020-01-18T20:33:25+05:00
[INFO] -----
```

Вопрос к аудитории: или не совсем отлично?



ЛОГИ

ЛОГИ

То, что выводится в панели сбоку — называется логи. От умения читать и разбираться в логах зависит ваше будущее как тестировщика и программиста.

```
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ bonus-calculator ---
[INFO] Surefire report directory: /Users/coursar/bonus-calculator/target/surefire-reports
-----
T E S T S
-----
Results :
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0 [INFO]
-----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.982 s
[INFO] Finished at: 2020-01-18T20:33:25+05:00
[INFO] -----
```

Выделенные строки сообщают нам, что ошибок нет, но и **ни одного теста не запущено!**



ЛОГИ

Т.е. мы попали в ту самую ловушку, о которой говорили: тесты есть, но они ничего не проверяют (они даже не запускаются Maven'ом).

Удостоверимся в этом, попробовав «уронить» тесты — результат всё равно будет BUILD SUCCESS.

В чём проблема?

ЛОГИ

Начнём читать лог с самого верха, собирая информацию о том, что происходит:

```
[INFO] -----< ru.netology:bonus-calculator >-----  
[INFO] Building bonus-calculator 1.0-SNAPSHOT  
[INFO] -----[ jar ]-----
```

Maven сообщает, что собирается «собирать» (build — сборка) наш проект, чтобы затем на нём запустить автотесты.

ЛОГИ

```
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ bonus-calculator ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
```

Maven запускает плагин `maven-resources-plugin` версии 2.6 для копирования ресурсов (файлов из каталога `src/main/resources`).

Файлов там нет, поэтому предсказуемо `Copying 0 resource`.

ЛОГИ

```
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ bonus-calculator ---
[INFO] Nothing to compile – all classes are up to date
[INFO]
```

Maven запускает плагин `maven-compiler-plugin` версии 3.1 для компиляции Java кода.

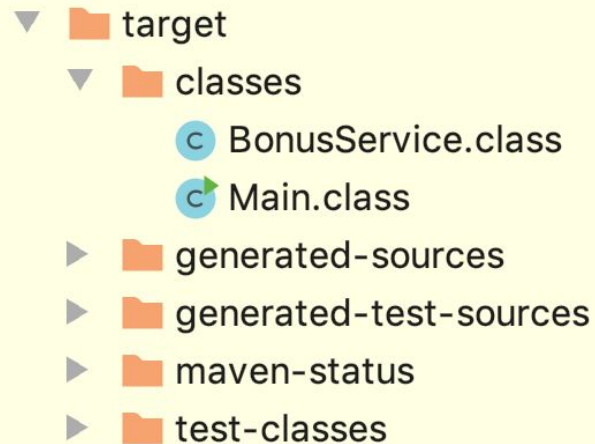
Maven старается делать бóльшую часть работы средствами плагинов — для ресурсов плагин `maven-resources-plugin`, для компиляции кода `maven-compiler-plugin`.

Но этот плагин почему-то говорит, что нечего компилировать: `all classes are up to date`.

ЛОГИ

Дело в том, что Maven достаточно умный — и если вы не меняли исходные коды проекта, то он считает, что и перекомпилировать ничего не нужно.

А скомпилированное хранится в каталоге `target`:



Хорошо бы протестировать на «чистой» системе.



mvn clean

Для очистки результатов предыдущей сборки есть специальная команда `mvn clean`.

Кроме того, Maven позволяет комбинировать команды: `mvn clean test` сначала всё почистит, а затем запустит тесты.

Но это не помогает. Значит идём дальше.

ЛОГИ

```
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ bonus-calculator ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/coursar/bonus-calculator/src/test/resources
[INFO]
```

Maven снова запускает плагин `maven-resources-plugin` версии 2.6 для копирования ресурсов (файлов из каталога `src/test/resources`).

Каталога такого нет, поэтому плагин ничего не делает.

ЛОГИ

```
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ bonus-calculator ---
[INFO] Changes detected – recompiling the module!
[INFO] Compiling 1 source file to /Users/coursar/bonus-calculator/target/test-classes
[INFO]
```

Maven снова запускает плагин `maven-compiler-plugin` версии 3.1 уже для компиляции кода тестов.

Если вы сделали `mvn clean`, то он скомпилирует, если нет — то получим:
`Nothing to compile — all classes are up to date.`

ЛОГИ

```
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ bonus-calculator ---
[INFO] Surefire report directory: /Users/coursar/bonus-calculator/target/surefire-reports
[INFO]
-----
T E S T S
-----
Results :
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.982 s [INFO] Finished at: 2020-01-18T20:33:25+05:00
[INFO] -----
```

Maven запускает плагин `maven-surefire-plugin` версии 2.12.4 для запуска тестов.

И мы видим, что **тестов он не находит**.



ДОБРО ПОЖАЛОВАТЬ В РЕАЛЬНЫЙ МИР!

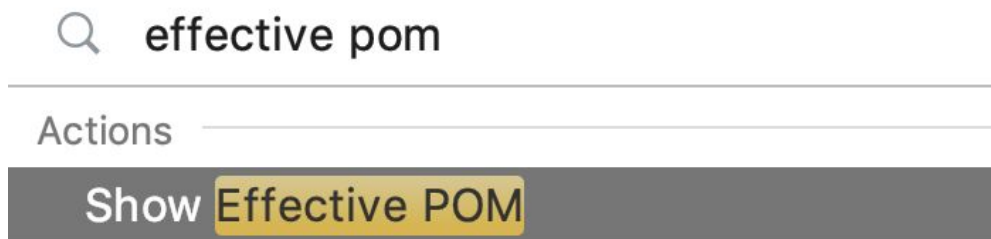
Если мы будем гуглить «maven surefire didn't execute tests» то рано или поздно попадём на ответ, что оказывается это происходит из-за связки JUnit5 и этой версии плагина.

Но откуда Maven берёт версии плагинов и как нам поменять версию?

SUPER POM

На самом деле, Maven хранит свой файл `pom.xml` (Super POM), в котором прописаны все версии, настройки и т.д. И когда он (Maven) работает он «накладывает» наш `pom.xml` на Super POM. Таким образом переопределяются настройки.

Полученный после «наложения» итоговый `pom` называется Effective POM. Мы можем два раза нажать `Shift` и ввести "effective pom":



Именно там будут прописаны все версии.

SUREFIRE PLUGIN

Чуть позже мы с вами научимся узнавать последние версии плагинов и их настройки, сейчас же пока скажем, что нам нужно добавить в наш `pom.xml`:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>2.22.2</version>
    </plugin>
  </plugins>
</build>
```

После этого сделать импорт изменений и запустить `mvn clean test`

SUREFIRE PLUGIN

```
[INFO] -----  
[INFO] T E S T S  
[INFO] -----  
[INFO] Running BonusServiceTest  
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.079 s  
[INFO] Results:  
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 9.107 s  
[INFO] Finished at: 2020-01-18T21:37:42+05:00  
[INFO] -----
```

ВПЕЧАТЛЕНИЯ

Q: эм... как-то всё сложно. Нельзя ли сделать попроще?

A: надо к этому привыкать. Мы сразу вам показываем то, как оно "в реале".

Сейчас никому не нужны «просто программисты». Вы должны знать, уметь настраивать и использовать инструменты, а также разбираться в проблемах.

Это как с автомобилем: недостаточно просто «уметь рулить». Нужно знать:

- как заправлять автомобиль, где, каким топливом;
- как заливать жидкость в бачок омывателя;
- когда менять резину на колёсах;
- как оформлять страховку и т.д.

В мире программирования так же.



ИТОГИ



ИТОГИ

Сегодня была одна из ключевых "нетеоретических" лекций.

Мы обсудили работу с:

- зависимостями;
- автотестами;
- логами;
- проблемами совместимости между различными инструментами.



ИТОГИ

А самое важное: мы обсудили ключевую проблему — можно написать тесты, которые не запускаются или ничего не тестируют.

Галочка при этом будет зелёная.

Поэтому всегда проверяйте, что ваши тесты «падают» при неправильных данных и читайте логи.

Всё нарабатывается с опытом.



HOTKEYS

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).

На Mac OS посмотреть клавиатурные сокращения можно через пункт меню Help -> Keymap Reference.



ДОМАШНЕЕ ЗАДАНИЕ

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задаем в чате Slack!
- Задачи можно сдавать по частям.
- Зачет по домашней работе проставляется после того, как приняты **все задачи**.



Задавайте вопросы и напишите отзыв о лекции!

ОКСАНА МЕЛЬНИКОВА

 Оксана Мельникова