Maven

Trener: Marcin Górecki

Gdańsk, 7.02.2018

Praktycznie niemożliwe we współczesnej firmie jest tworzenie dobrej jakości oprogramowania bez automatyzacji procesu budowania, reużywalności kodu, automatyzacji testów etc.

Maven jako narzędzie do zarządzania zależnościami

Nie piszecie sami aplikacji na swoje telefony

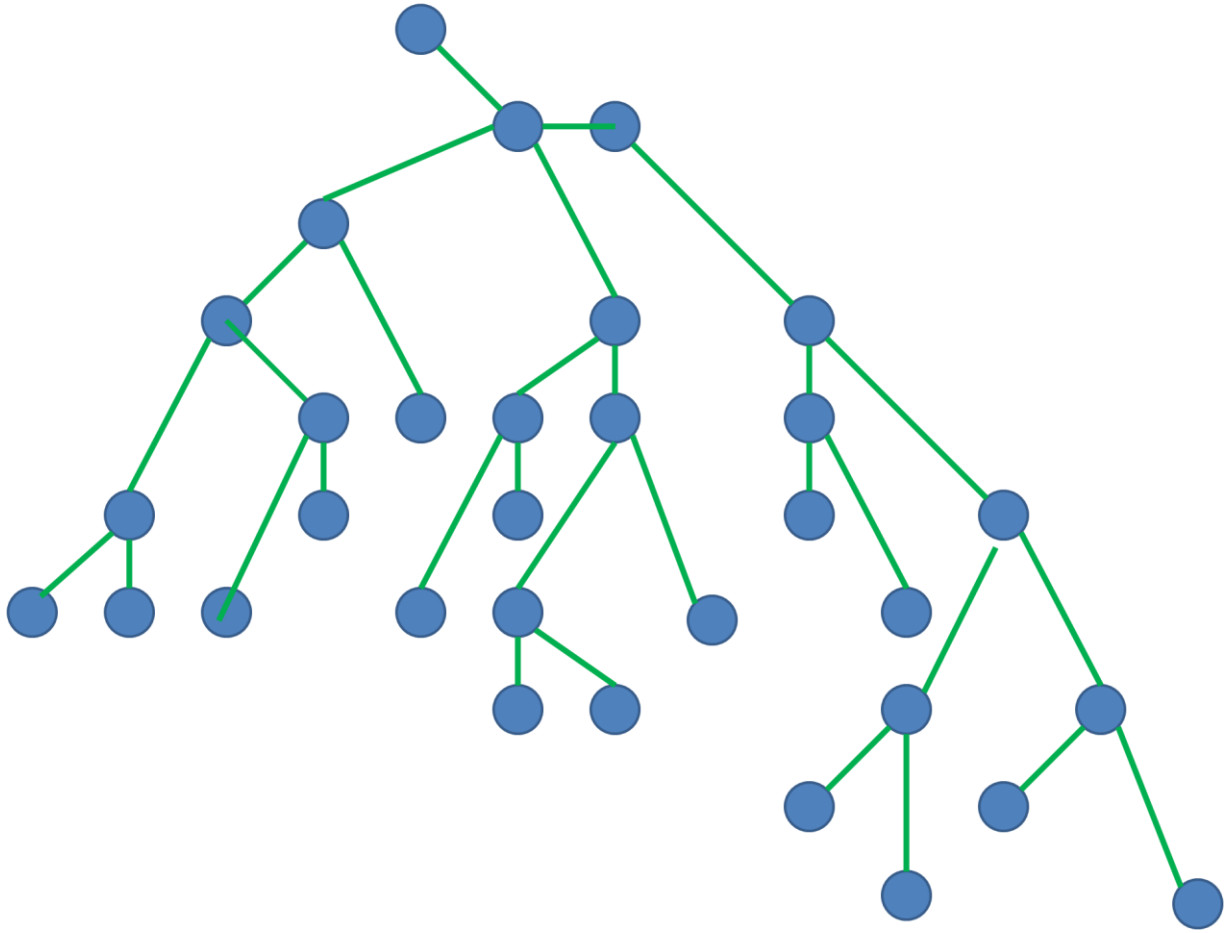
Maven – moduły, których używa Twoja aplikacja

- Moduł dostępu do bazy danych (ORM)
- Moduł obliczeniowy matematyczne dostarczony przez R&D
- Obsługa MVC i servletów
- Moduł logowania
- Klient obsługi płatności
- Webservices
- Wsparcie testowania
- Code candies

pom.xml – moduły których używasz, czyli „zależności”

```
<dependency>
        <groupId>commons-collections</groupId>
        <artifactId>commons-collections</artifactId>
</dependency>

<dependency>
        <groupId>joda-time</groupId>
        <artifactId>joda-time</artifactId>
</dependency>

<dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
</dependency>

<dependency>
        <groupId>net.sourceforge.javacsv</groupId>
        <artifactId>javacsv</artifactId>
</dependency>

<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-annotations</artifactId>
</dependency>
```

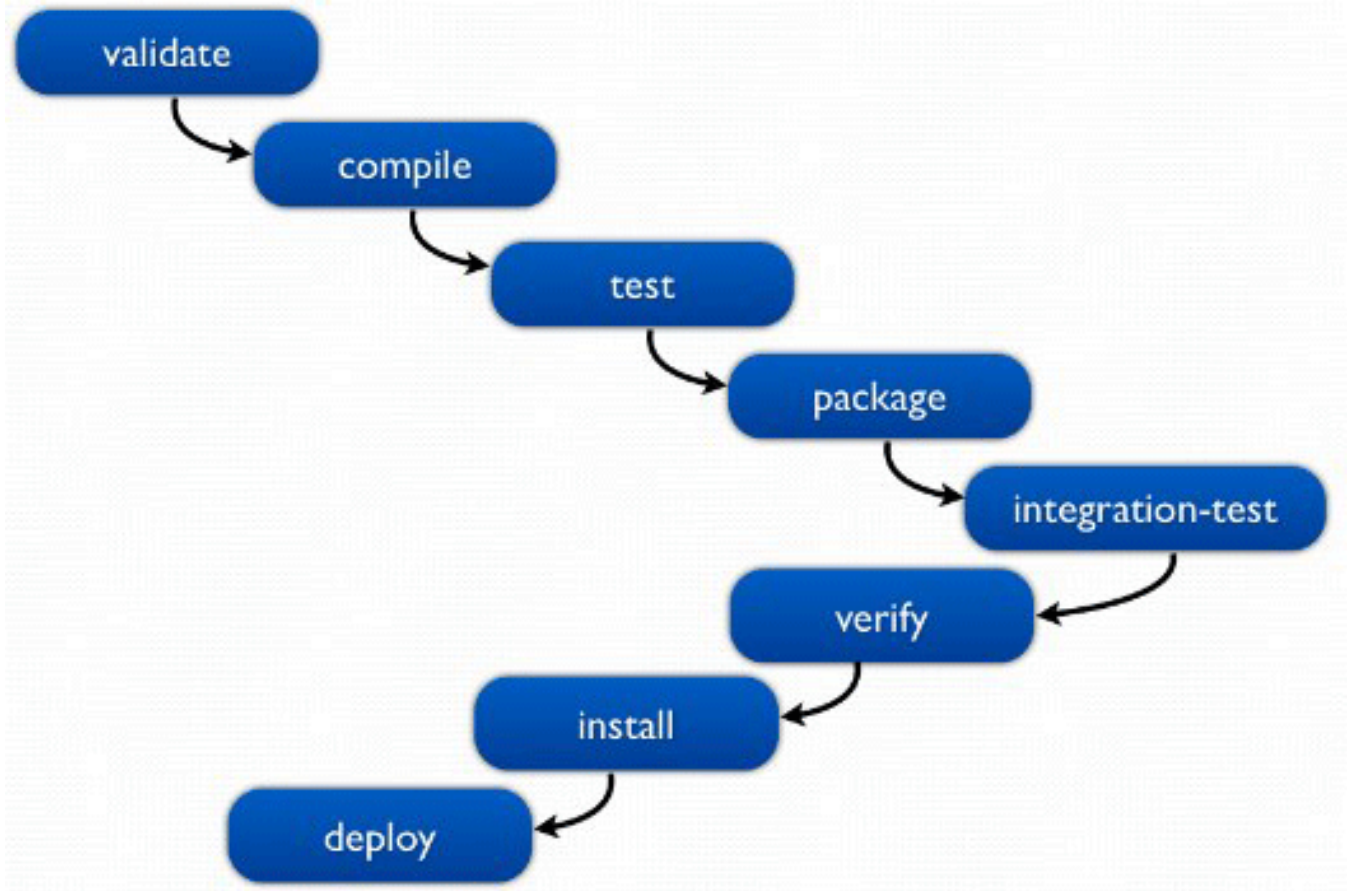
Jak wyglądają prawdziwe aplikacje



Jak wyglądają prawdziwe aplikacje

- Korzystają z modułów innych programistów
- Zależności między własnymi modułami aplikacji
- DRY – Don’t Repeat Yourself
- Automatyczne generowanie kodu
- Różne wersje aplikacji dla środowisk DEV/PROD - profile
- Quality assurance – Automatyczne uruchamianie testów
- Continous integration

„Taśma produkcyjna” albo Build Lifecycle



Build Lifecycle

- *validate* : czy dobrze zdefiniowaliśmy jak chcemy budować samochód
- *compile* : ściagamy podzespoły od dostawców, składamy gotowe części w całość
- *test* : ściągamy od dostawców pomoce do testowania (np. mierniki poziomu spalin, mierniki grubości lakieru, wyważarki kół), budujemy narzędzia do sprawdzania jakości pojedynczych podzespołów (czy kręcą się koła etc.)
- *package* : przygotowujemy gotowy produkt (naklejamy folie ochronne, wkładamy w opakowanie)

Build Lifecycle

- *verify* : testy integracyjne – czy podzespoły ze sobą prawidłowo współpracują – uruchamiamy silnik i próbujemy się przejechać

- *install* : wysyłamy gotowy produkt do magazynu naszej firmy, tak aby był dostępny do użytku wewnętrznego. Inne działy firmy mogą użyć tego samochodu

- *mvn release* : produkt jest gotowy – nadajemy mu finalną wersję i wprowadzamy na rynek

MAVEN

- Narzędzie do automatyzowania budowania aplikacji
- Nie wymaga instalacji
- Wymaga zdefiniowania zmiennej JAVA_HOME
- Wymaga zdefiniowania zmiennej MAVEN_HOME
- Wymaga dodania MAVEN_HOME/bin do zmiennej PATH
- Konfiguracja maven: MAVEN_HOME/conf/settings.xml

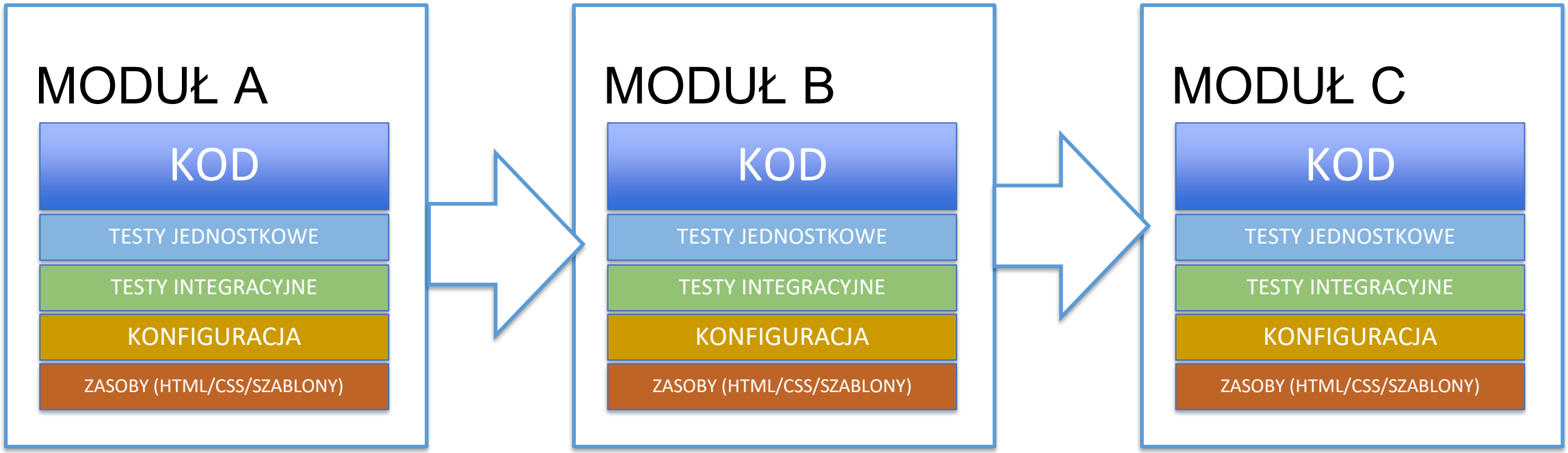
Jakie problemy rozwiązuje maven?

Proste projekty, złożone tylko z klas wyglądają tak:

- Main.java
- Pakiet
  - KlasaA.java
  - KlasaB.java
  - KlasaC.java

Co jeśli projekt jest złożony?

Co jeśli projekt jest jeszcze bardziej złożony?



Ćwiczenie 1

```
C:\Users\lukaszd>mvn --version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T17:41:47+01:00)
Maven home: c:\Users\lukaszd\apps\apache-maven-3.3.9\bin\..
Java version: 1.8.0_45, vendor: Oracle Corporation
Java home: c:\Program Files\Java\jdk1.8.0_45\jre
Default locale: pl_PL, platform encoding: Cp1250
OS name: "windows 8.1", version: "6.3", arch: "amd64", family: "dos"
```

- Sprawdź czy, i jakiej wersji jest zainstalowany maven

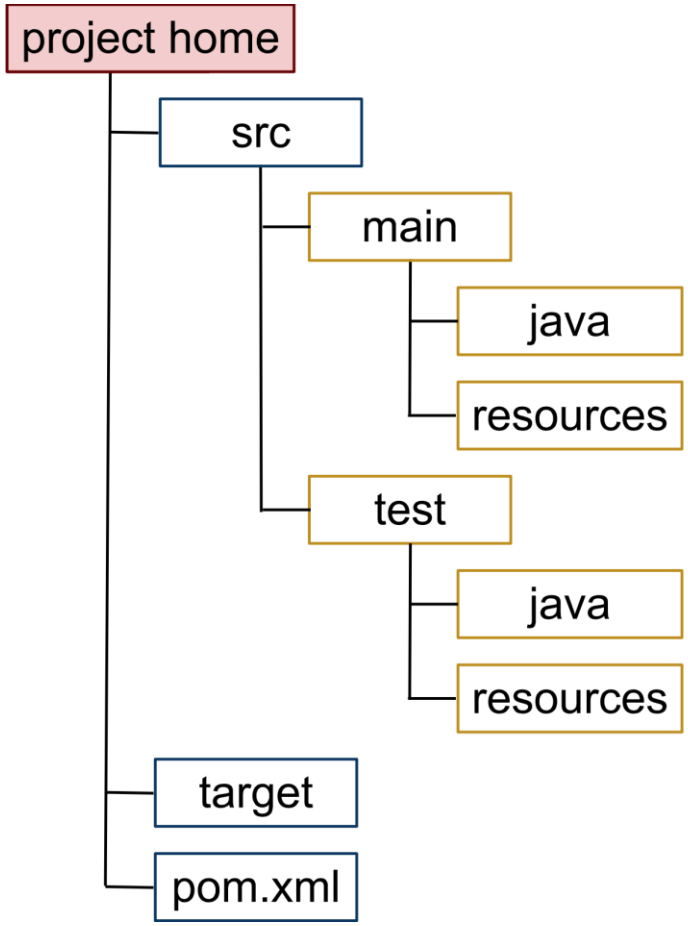
Archetypy - szablony aplikacji

mvn archetype:generate
-D**groupId**=nazwa-grupy
-D**artifactId**=nazwa-artefaktu
-D**archetypeArtifactId**=maven-archetype-quickstart
-D**interactiveMode**=false

Ćwiczenie 2

- Stworzyć przykładowy projekt używając wiersza poleceń oraz z poziomu IntelliJ
- Projekt powinien nazywać się: **my-application**
- Grupa: **infoshare**
- Użyj archetypu: **maven-archetype-quickstart**

Maven Directory Layout



Pluginy

- Większość funkcjonalności uruchamiamy poprzez plugin
- Każdy plugin zawiera w sobie listę celów (goal), które może wykonać
- Składnia: mvn [plugin-name]:[goal-name]
- Przykłady:

```
mvn compiler:compile
mvn compiler:testCompile
mvn surefire:test
mvn jar:jar
mvn archetype:generate
```

Fazy i Build Lifecycle - definicje

- Aby nie uruchamiać za każdym razem każdego pluginu oddzielnie zdefiniowana została ich sekwencja - Build Lifecycle
- Build Lifecycle jest sekwencją zdefiniowanych faz
- Każda faza (Build Phase) ma przypisany zestaw wywołań pluginów z goalami
- Fazy uruchamiają się w ustalonej kolejności
- Uruchomienie danej fazy oznacza uruchomienie wszystkich poprzednich faz w Build Lifecycle

Fazy w Default Build Lifecycle

- **validate** - sprawdzenie, czy projekt jest poprawnie skonfigurowany
- **compile** - kod źródłowy jest kompilowany
- **test** - uruchamiane są testy jednostkowe
- **package** - budowana jest paczka z aplikacją (JAR/WAR/EAR)
- **integration-test** - przeprowadzane są testy integracyjne
- **verify** - sprawdzenie, czy paczka jest poprawna
- **install** - paczka umieszczana jest w repozytorium lokalnym - może być używana przez inne projekty jako zależność
- **deploy** - paczka umieszczana jest w repozytorium zdalnym (opublikowana)

Build Lifecycle - jeszcze raz

- Lifecycle to lista nazwanych faz (ang. Phase), które są wykonywane do osiągnięcia celu
- Do faz przypisane są przypisane sa pluginy, np.:

  compile:

  - **resources:resources**
  - **compiler:compile**
- Wywołanie którejś z faz skutkuje wywołaniem wszystkich poprzednich
- Wywołując mvn, możemy podać dwie fazy, np.: mvn clean install

Ćwiczenie 3 - Build Lifecycle

- Zbuduj stworzoną aplikację wywołując każdą fazę po kolei
- Sprawdź co zmienia się w katalogu target po każdej z faz
- Zobacz co zawiera plik **pom.xml**

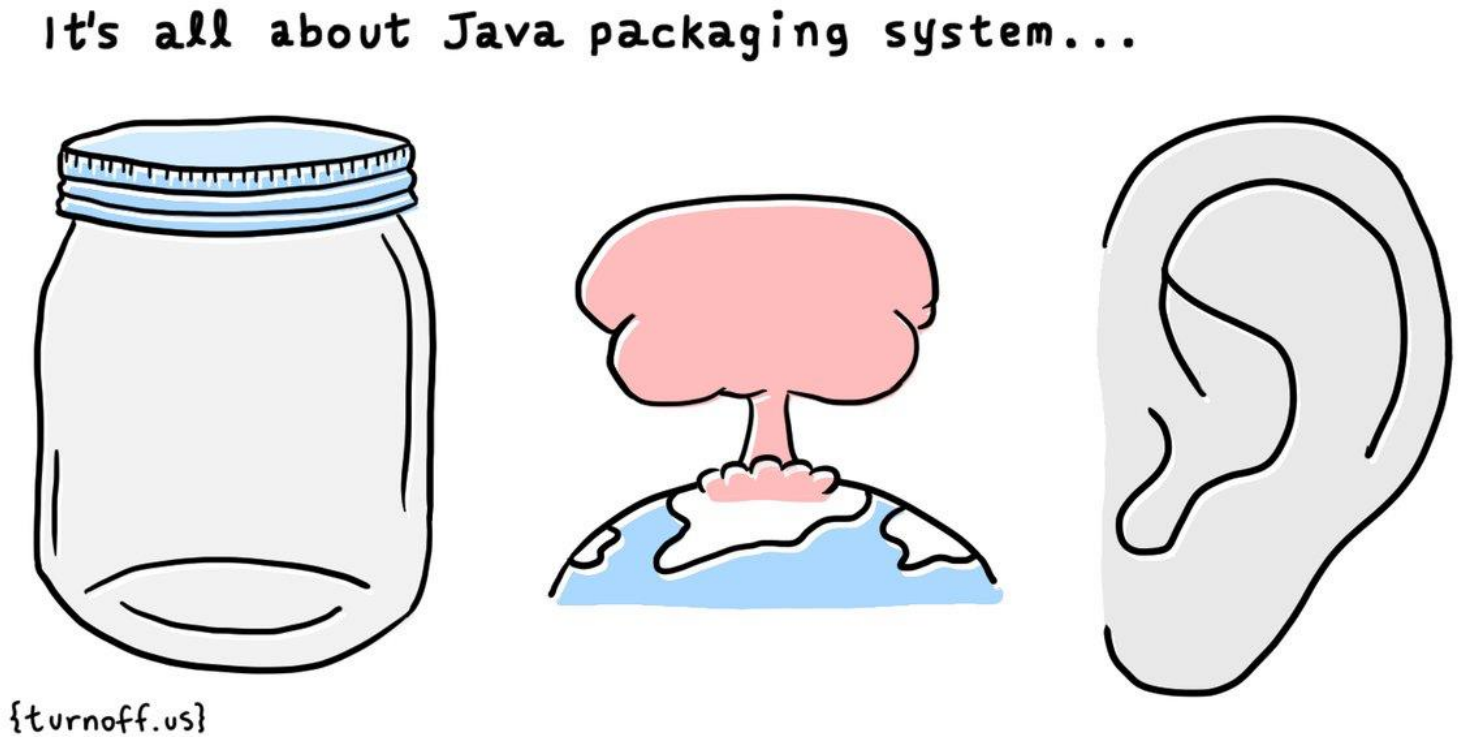
Plik pom.xml - konfiguracja modułu

- **groupId** – identyfikator organizacji / firmy
- **artifactId** – identyfikator (nazwa) modułu / projektu
- **version** – wersja

Często oznaczana jako SNAPSHOT – wersja rozwojowa, której nie można używać produkcyjnie

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.marcingorecki</groupId>
    <artifactId>ChartAnalysis</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
```

pom.xml - parametr packaging



- jar
- war
- ear

  ...
- pom

Ćwiczenie 4 - Moduły

- Zmień w aplikacji my-application atrybut packaging na „**pom**"
- Usuń katalogi **src** oraz **target**
- Wewnątrz projektu my-application utwórz moduł my-jar.
- Ma to być moduł JAR utworzony archetypem **maven-archetype-quickstart**.
- Sprawdź zawartość pom.xml – w szczególności :
  - parent
  - version

Ćwiczenie 5 - Moduły

- Wewnątrz projeku my-application utwórz moduł my-web. Ma to być moduł WAR utworzony archetypem **maven-archetype-webapp**.
- Sprawdź zawartość pom.xml – w szczególności :
  - parent
  - version
  - packaging
- Zbuduj projekt i sprawdź zawartość: /my-web/target/my-web

Ćwiczenie 6 - Zależności

- Dodaj do my-application zależność do:

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.21</version>
</dependency>
```

- Zbuduj projekt i sprawdź zawartość: /my-web/target/my-web/WEB-INF/lib

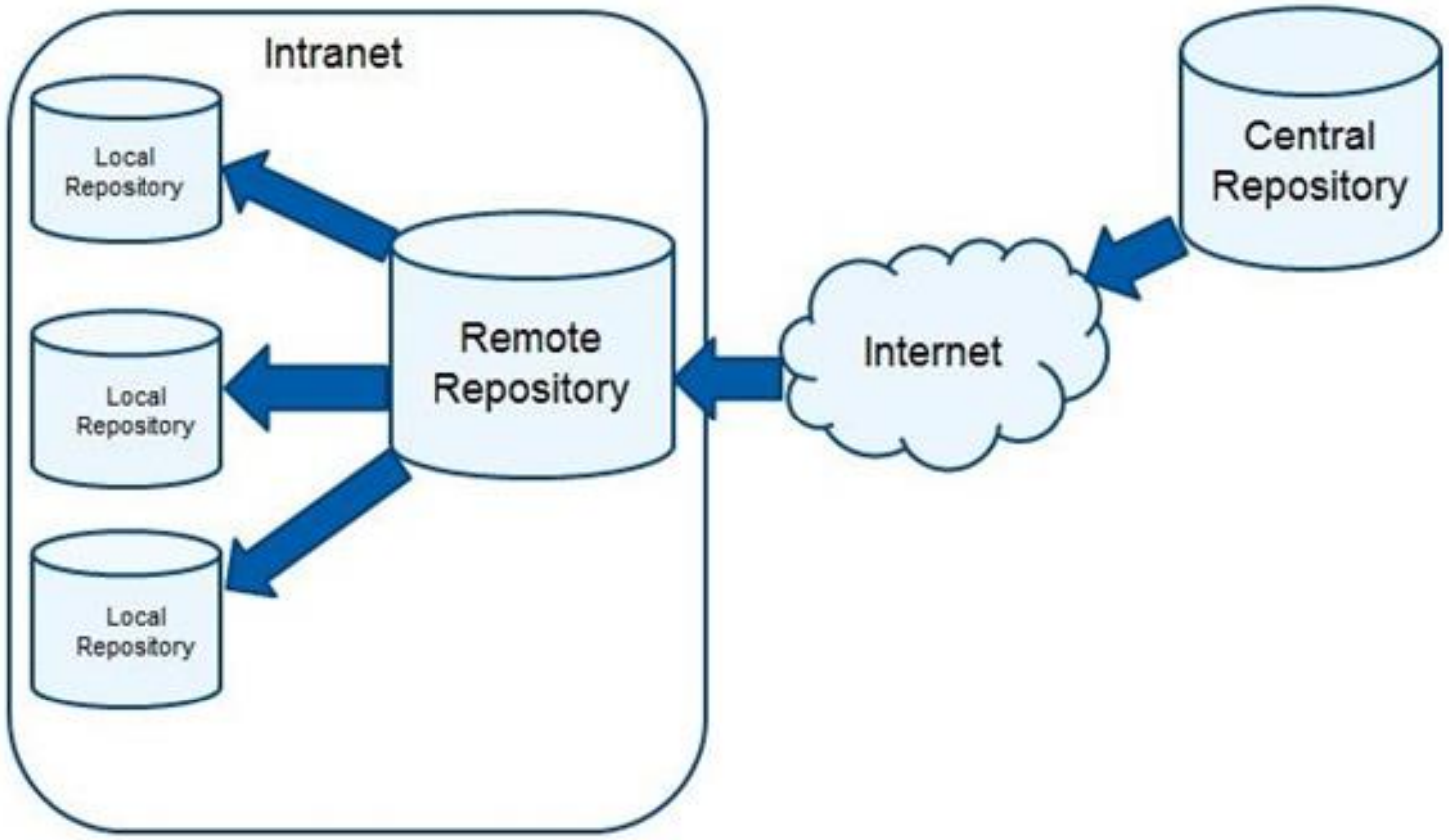
POM – zależności – scope

- compile - (domyślna) w trakcie kompilacji, przechodnia
- provided - kompilacja, testy, nieprzechodnia (dostarcza kontener)
- runtime - wymagana do uruchomienia, nie do kompilacji
- test - kompilacja oraz uruchomienie testów
- system - dostarczenie zależności przez system, poza repozytorium

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

Repozytoria – publiczne i lokalne

- Konfiguracja repozytoriów znajduje się w pliku settings.xml



Ćwiczenie 7 - Zależności z innych repozytoriów

- Dodaj do my-jar zależność do:

```
<dependency>
    <groupId>org.primefaces</groupId>
    <artifactId>primefaces</artifactId>
    <version>3.0</version>
</dependency>
```

- Zbuduj projekt

Repozytoria – konfiguracja repozytorium

```
<repositories>
    <repository>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
        <id>central</id>
        <name>Central Repository</name>
        <url>http://repo.maven.apache.org/maven2</url>
    </repository>
</repositories>
```

Ćwiczenie 8 - Konfiguracja repozytoriów

- Dodaj do pom.xml / settings.xml:

```
<repository>
  <id>prime-repo</id>
  <name>PrimeFaces Maven Repository</name>
  <url>http://repository.primefaces.org</url>
  <layout>default</layout>
</repository>
```

- Zbuduj projekt, sprawdź zawrtość my-web/target/my-web/WEB-INF/lib

Ćwiczenie 9 - Testy

Wracamy do faz budowania projektu

- Otwórz plik */my-jar/src/test/java/*/AppTest.java*
- Zamień *assertTrue(true)*; na *assertTrue(false)*
- Co się stało?
- Przywróć testy do działania

Maven i testy jednostkowe

Za testy odpowiedzialny jest plugin **surefire**

- Testy jednostkowe są uruchamiane przy każdym budowaniu projektu w fazach > package
- *mvn test* przed *package*
- Plugin uruchamia testy, które znajdują się w klasach o nazwie *Test.java

Ćwiczenie 10 - Testy

- Skompiluj i uruchom za pomocą mavena tylko testy jednostkowe

Maven i testy integracyjne

Za testy integracyjne odpowiedzialny jest plugin **failsafe**

- Testy integracyjne są uruchamiane przy każdym instalowaniu projektuw repozytorium
- *mvn verify* przed *install*
- Plugin uruchamia testy, które znajdują się w klasach o nazwie *IT.java, IT*.java, *ITCase.java

Ćwiczenie 11 - Testy integracyjne

- Skopiuj plik AppTest.java -> AppIT.java
- Zamień *assertTrue(true)*; na *assertTrue(false)*
- Skonfiguruj wtyczkę failsafe; uruchom fazy clean package i clean install

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-failsafe-plugin</artifactId>
        <version>2.18.1</version>
        <executions>
            <execution>
                <goals>
                <goal>integration-test</goal>
                <goal>verify</goal>
            </goals>
            </execution>
        </executions>
    </plugin>
</plugins>
```

Dziękuję za uwagę