Maven - maven.apache.org

Łukasz Drzewiecki

Gdańsk, 17 września 2016 roku

Wprowadzenie / powtórka

- CLASSPATH
- PATH
- JAR – Java ARchive
- WAR – Web ARchive – aplikacja WEB
  - Wiele JARów
- EAR – Enterprise Archive
  - Wiele JARów
  - Wiele WARów

WAR – struktura

- META-INF
- WEB-INF
  - classes – skompilowane klasy
  - lib – biblioteki należące do CLASSPATH aplikacji web
  - web.xml – na tym etapie nie interesuje nas to☺
- Pliki WEB (HTML, JSP, JSF, …) – na tym etapie nas to nie interesuje

Wprowadzenie / powtórka

- CLASSPATH
- PATH
- JAR – Java ARchive
- WAR – Web ARchive
- EAR – Enterprise ARchive

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

Command: mvn --version

```
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"
```

- Ćwiczenie 1 – sprawdź czy i w jakiej wersji masz zainstalowanego maven

Archetypy – „szablony”

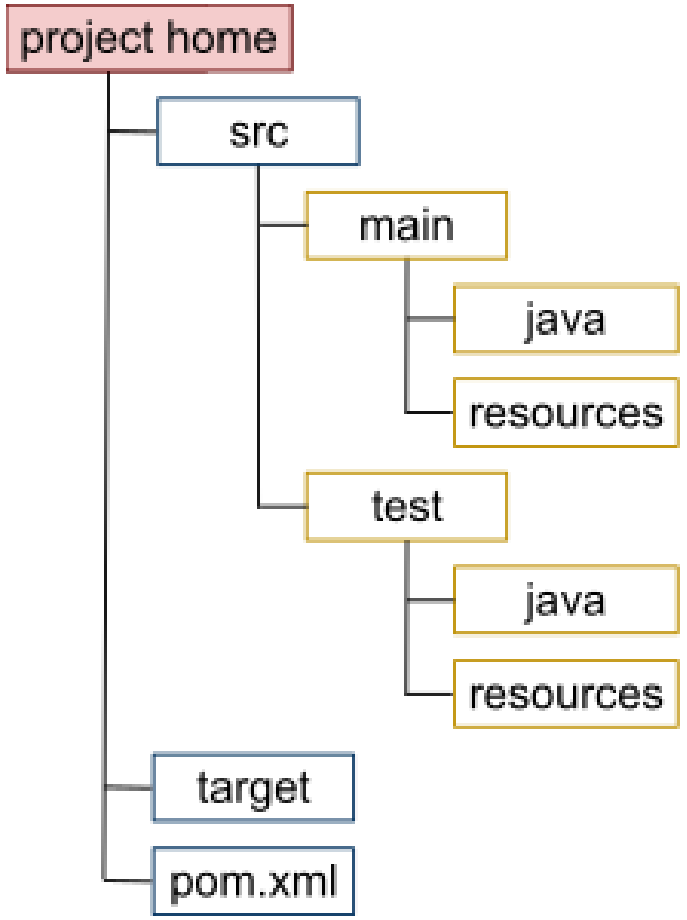
- Archetyp = szablon projektu

```
mvn archetype:generate
-DgroupId=infoshare
-DartifactId=maven-test
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

Ćwiczenie 2

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

Podstawowa struktura katalogów



Podstawowe koncepcje - Pluginy

- Większość funkcjonalności jest zaimplementowana jako PLUGIN
- Każdy plugin zawiera w sobie listę celów (goal), które może wykonać
- Składnia: mvn [plugin-name]:[goal-name]
- Przykłady:
  - mvn compiler:compile
  - mvn surefire:test
  - mvn jar:jar

Podstawowe koncepcje - Phases and Default Lifecycle

1. validate
2. generate-sources
3. process-sources
4. generate-resources
5. process-resources
6. compile
7. process-test-sources
8. process-test-resources
9. test-compile
10. test
11. package
12. install
13. deploy

- Dodatkowe fazy:
  - clean
  - site

Build Lifecycle – najważniejsze fazy

- **validate** - sprawdzenie, czy projekt jest poprawny i czy wszystkie niezbędne informacje zostały określone
- **compile** - kod źródłowy jest kompilowany
- **test** - przeprowadzane są testy jednostkowe
- **package** - budowana jest paczka dystrybucyjna
- **integration-test** - zbudowany projekt umieszczany jest w środowisku testowym, gdzie 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 – cykl budowania aplikacji

- Lista nazwanych faz (ang. Phase), które są wykonywane do osiągnięcia celu
- Do przypisane są przypisane sa pluginy, np.:
  - compiler:compile -> compile
  - surfire:test -> test
- 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 POM.XML

POM – koordynaty - groupId:artifactId:version

- groupId – identyfikator „organizacji”
- artifactId – identyfikator moduły/projektu
- version – wersja – SNAPSHOT – wersja, której nie można ufać.

```
<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>lukaszd</groupId>
  <artifactId>jjdz-materialy-jse2</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
```

POM – packaging

- jar - domyślny
- war
- ear
- pom

Ćwiczenie 4

- Zmień w aplikacji my-application atrybut packaging na pom.
- Usuń 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

- 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/WEB-INF/lib

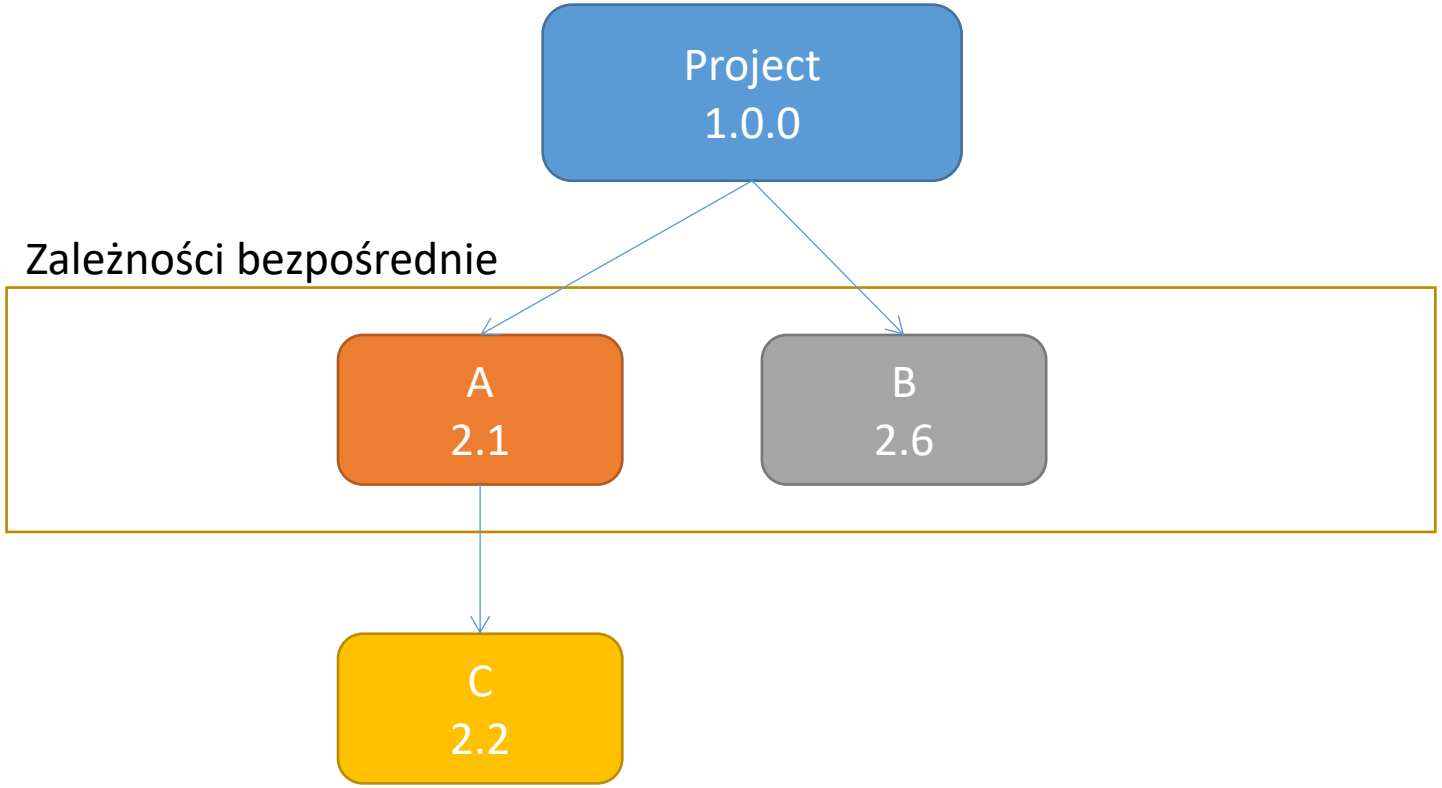
Ćwiczenie 6

- Dodaj do my-jar zależność do:

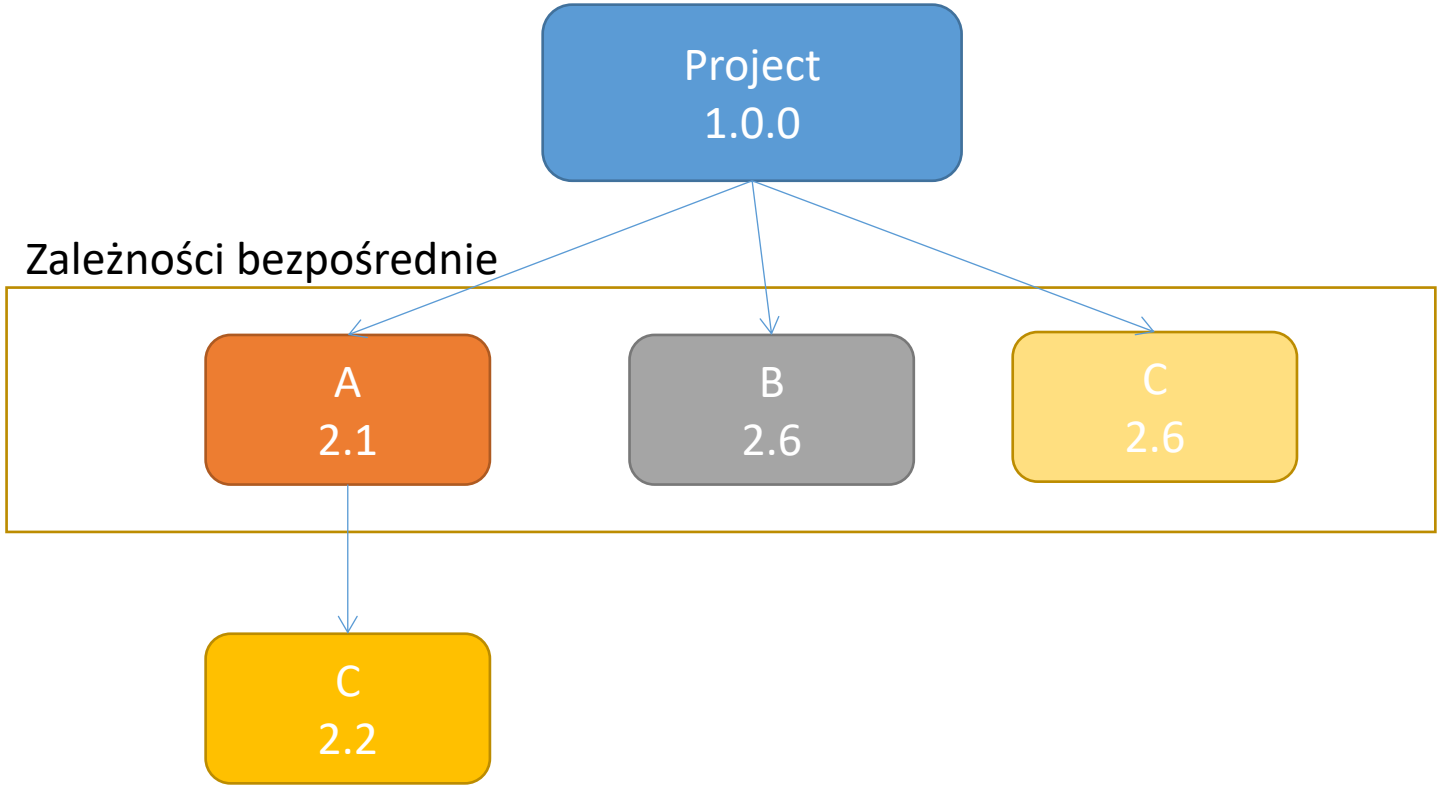
```
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.21</version>
</dependency>
```

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

POM – zależności – przechodniość



POM – zależności – przechodniość



Ćwiczenie 7

- Dodaj do my-jar zależność do:

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.1.3</version>
</dependency>
```

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

POM – zależności – scope

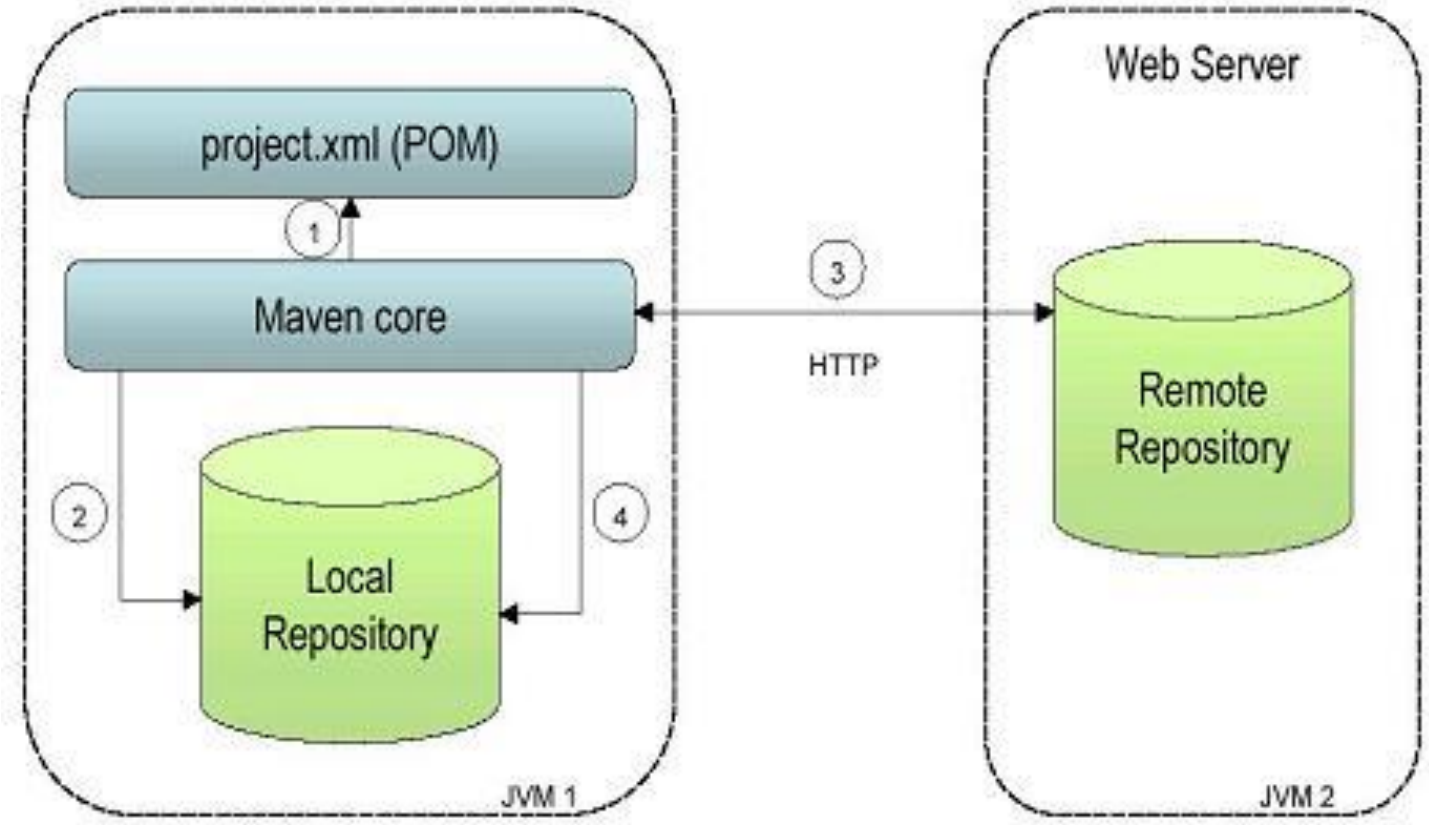
- compile – domyślny – w trakcie kompilacji, przechodnia
- provided – kompilacja, testy, nie przechodnia (dostarcza kontener)
- runtime – testy oraz w trakcie uruchomienia
- test – tylko do testów, kompilacja oraz wywołanie
- system – ręczne dostarczenie zależności

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.8.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

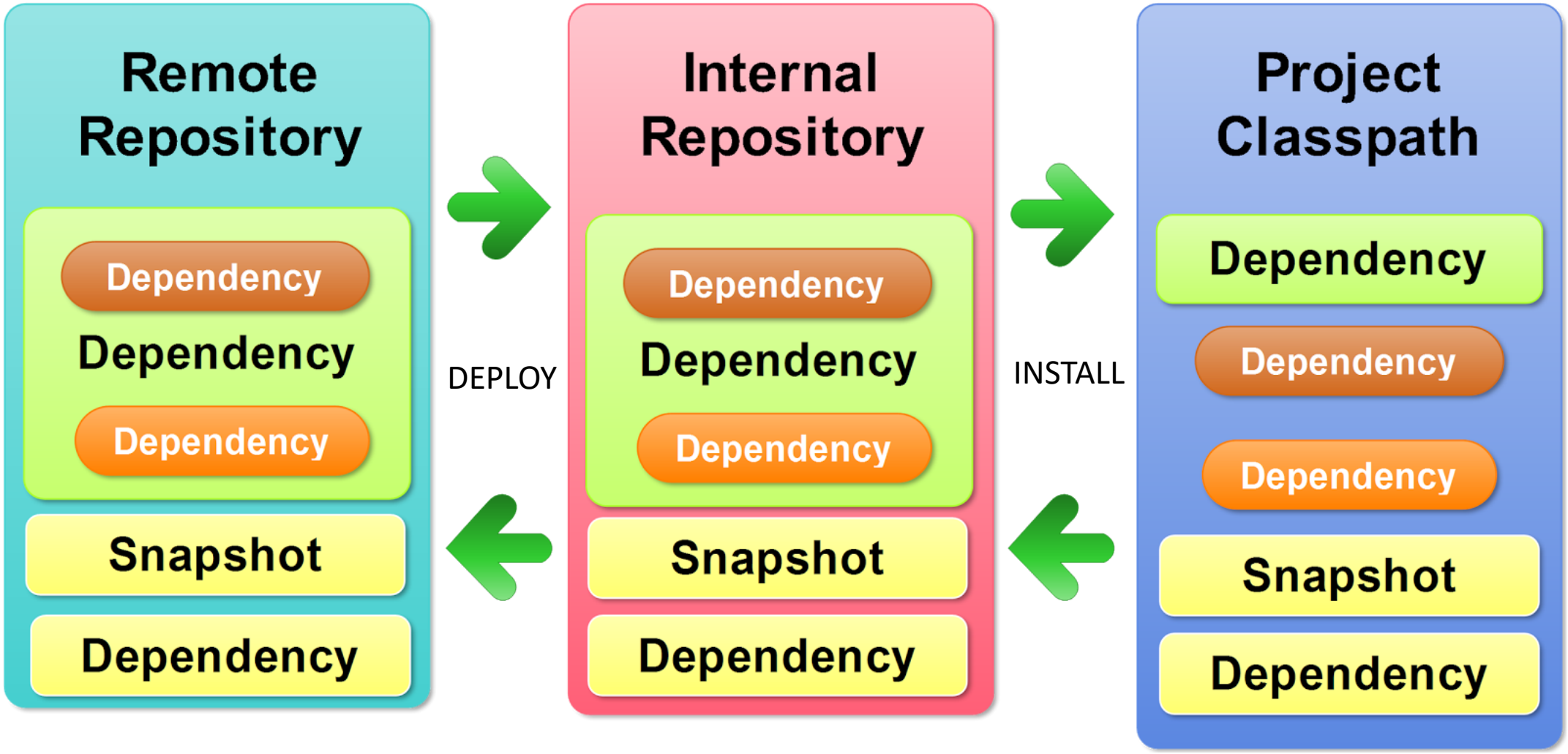
POM – wykluczanie zależności

```
<dependency>
        <groupId>sample.ProjectA</groupId>
        <artifactId>Project-A</artifactId>
        <version>1.0</version>
        <exclusions>
                <exclusion> <!-- declare the exclusion here -->
                <groupId>sample.ProjectB</groupId>
                 <artifactId>Project-B</artifactId>
                </exclusion>
        </exclusions>
</dependency>
```

Repozytoria – publiczne i lokalne



Repozytoria – wewnętrzne i zdalne



Ćwiczenie 8

- Dodaj do my-web zależność do:

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

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

Repozytoria – definiowanie repozytorium

- Repozytorium domyślne

```
<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 9

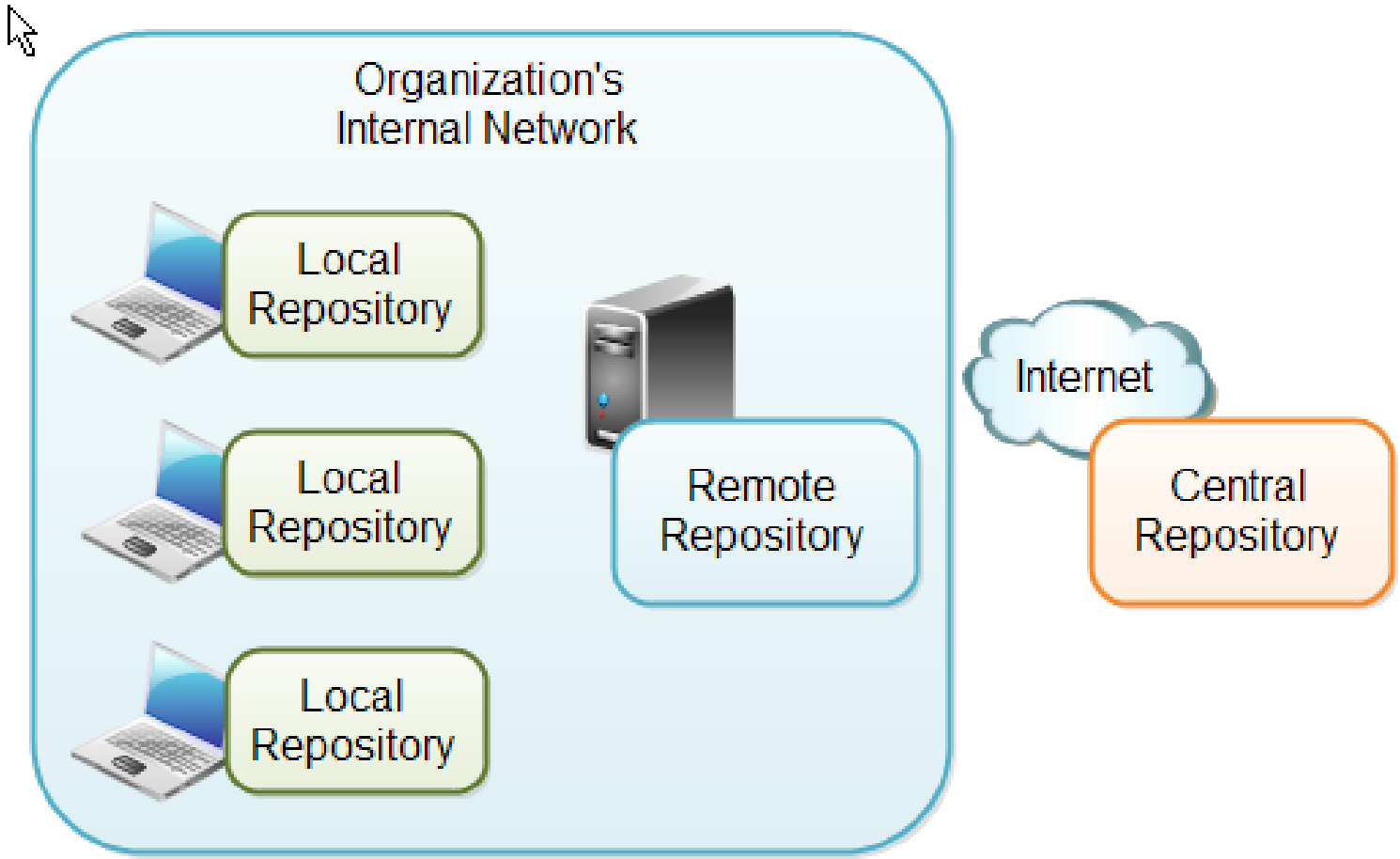
- Dodaj repozorium:

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

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

Repozytoria prywatne

- Nexus
- Archiva



Ćwiczenie 10

- 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 *surfire*
- Testy jednostkowe są uruchamiane przy każdym budowaniu projektu

  mvn test przed package

- Plugin uruchamia testy, które znajdują się w klasach o nazwie *Test.java

maven i testy integracyjne

- Do uruchamiania testów integracyjnych służy wtyczka *failsafe*
- Testy integracyjne są uruchamiane przy każdym instalowaniu projektu w repozytorium

mvn verify przed install

- Plugin uruchamia testy, które znajdują się w klasach o nazwie *IT.java, IT*.java, *ITCase.java

Ćwiczenie 11

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

```
<build>
  <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>
```

Ustawienie wersji JAVA

```
<build>
   <plugins>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-compiler-plugin</artifactId>
         <configuration>
            <source>1.5</source>
            <target>1.5</target>
         </configuration>
      </plugin>
   </plugins>
</build>
```

Zmienne

- Deklaracja:
  <**properties**>
  <**log4j.version**>1.1.3</**log4j.version**>
  </**properties**>
- Użycie

<**version**>${log4j.version}</**version**>

- Ćwiczenie 12:
  Zdefiniuj zmienne dla wszystkich wersji zależności. Zbuduj aplikację i sprawdź czy działa.

Dziękuję za uwagę