Podstawy Inżynierii Oprogramowania

Laboratorium

Apache Maven

przygotował: dr inż. Jacek Wiślicki

1. Historia zmian

Data	*Wersja*	*Autor*	*Opis zmian*
2010	1.0	Jacek Wiślicki	Pierwsza wersja dokumentu
06.01.2013	1.1	Radosław Adamus	Poprawki związane ze zmianami w Maven

1. Wprowadzenie

Apache Maven jest kolejnym narzędziem do wykonywania *buildów* aplikacji platformy Java. Projekt jest znacznie młodszy niż Apache Ant (Ant 1.1 – jako samodzielny projekt od roku 2000, Maven 1 – wydzielony z projektu Turbine i wydany jako samodzielny w roku 2004, Maven 2 – 2005, Maven 3 – rok 2010), obecnie trudno jednoznacznie określić, które z tych narzędzi jest bardziej popularne. Duża część starszych programistów i *build managerów* wciąż pozostaje przy Ant (przyzwyczajenie, przenoszenie istniejącej konfiguracji ze starszych projektów), z drugiej strony wiele nowych projektów rozpoczynanych jest w oparciu o Maven. Bez względu na popularność, Maven oferując podobne funkcjonalności do Anta, różni się od niego znacznie. Podstawową różnicą jest zorientowanie na projekt i jego dowolną złożoność – Ant wprawdzie pozwala na wykonywanie *buildów* dowolnie skomplikowanych projektów i wykonywanie wielu skryptów *buildfile* znajdujących się w różnych lokalizacjach, jednak w przypadku Mavena ta funkcjonalność stała się jednym z podstawowych determinantów architektury. Drugą zauważalną od razu różnicą jest zarządzanie zależnościami projektu – Ant wymagał od projektanta/programisty samodzielnego „ręcznego" zapewnienia dostępności odpowiednich bibliotek (np. plików JAR) i skonfigurowania odpowiednich *classpath*, Maven opiera się na publicznych repozytoriach bibliotek, których zasoby mogą zostać wykorzystane z dowolnego komputera podłączonego do Internetu (można także tworzyć i wykorzystywać prywatne i wewnątrzkorporacyjne repozytoria, niedostępne dla świata, jednak podczas laboratorium nie będziemy z nich korzystać). Pewną różnicę stanowi także fakt jawnego oparcia Mavena o wtyczki (*plugins*), które stanowią podstawę działania narzędzia. Wtyczki są odpowiednikami zadań (*tasks*) pojawiających się w Ant i także tutaj możemy korzystać ze zbioru publicznie dostępnych wtyczek, jak również pisać własne (istnieje także wtyczka pozwalająca wykonywać zadania Anta).

2. Instalacja i uruchomienie

Pliki źródłowe i skomplikowane binaria Apache Maven przeznaczone dla różnych platform systemowych można pobrać z http://maven.apache.org/download.html. Aplikacja nie wymaga instalacji, wystarczy rozpakować archiwum zawierające pliki wykonywalne (podczas laboratorium będziemy używać dobrze znanego i zdokumentowanego Maven 2, pomimo że dostępna jest już wersja 3) do dowolnego katalogu (np. C:/maven dla systemów MS Windows), wygodnie jest także upewnić się, że skrypt *mvn* (*mvn.bat*) w katalogu *bin* jest dostępny na ścieżce systemowej (zmienna środowiskowa PATH), abyśmy nie musieli wywoływać go podając pełną ścieżkę.

Najprostsze uruchomienie aplikacji wymaga wpisania w konsoli (wierszu poleceń) komendy *mvn* w katalogu projektu, z podaniem nazwy wtyczki i nazwy celu (*goal*), który wtyczka może wykonać (wtyczka może oferować dowolną liczbę celi):

```
mvn [nazwa-wtyczki]:[nazwa-celu]
```

np.:

```
mvn jar:jar
```

3. Projekt i POM

Pliki POM (*Project Object Model*) są dokumentami XML i stanowią podstawowy deskryptor projektu wykorzystywany przez Mavena (domyślna nazwa to *pom.xml* w głównym katalogu projektu). Struktura pliku POM wraz z opisem elementów pokazana jest poniżej:

```
<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>
```

```
  <!-- Podstawowe informacje o projekcie wymagane przez Mavena -->
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <version>...</version>
  <packaging>...</packaging>
  <dependencies>...</dependencies>
  <parent>...</parent>
  <dependencyManagement>...</dependencyManagement>
  <modules>...</modules>
  <properties>...</properties>

  <!-- Ustawienia buildu -->
  <build>...</build>

  <!-- Ustawienia generacji raportów i dokumentacji -->
  <reporting>...</reporting>

  <!-- Pozostałe (ogólne) informacje o projekcie -->
  <name>...</name>
  <description>...</description>
  <url>...</url>
  <inceptionYear>...</inceptionYear>
  <licenses>...</licenses>
  <organization>...</organization>
  <developers>...</developers>
  <contributors>...</contributors>

  <!-- Ustawienia dotyczące środowiska -->
  <issueManagement>...</issueManagement>
  <ciManagement>...</ciManagement>
  <mailingLists>...</mailingLists>
  <scm>...</scm>
  <prerequisites>...</prerequisites>
  <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
  <distributionManagement>...</distributionManagement>
  <profiles>...</profiles>
</project>
```

Poniżej znajdują się jedynie informacje podstawowe i istotne z punktu widzenia laboratorium, natomiast szczegółowy opis struktury i elementów POM można znaleźć pod adresem http://maven.apache.org/pom.html.

3.1. Podstawy

Wersja modelu (`modelVersion`) definiuje XSD (strukturę) dokumentu, przy czym 4.0.0 jest jedyną obsługiwaną przez Maven 2. Podstawę konfiguracji projektu stanowią elementy `groupId`, `artifactId` i `version`. Ich określenie stanowi minimum konieczne dla wykonania *buildu*. Tak więc minimalna wersja pliku POM ma postać:

```
<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>org.codehaus.mojo</groupId>
  <artifactId>my-project</artifactId>
  <version>1.0</version>
</project>
```

Przy czym zarówno `groupId` jak i `version` mogą być dziedziczone z nadrzędnego POM (rodzica) – dziedziczenie w plikach POM opisane jest poniżej. Struktura `groupId:artifactId:version` stanowi unikatowy adres generowanego artefaktu w repozytoriach Mavena (także opisane poniżej). Znaczenie poszczególnych elementów jest następujące:

- `groupId` – unikatowy identyfikator w ramach organizacji lub projektu. Przykładowo, wszystkie podstawowe artefakty Mavena definiowane są przez grupę „org.apache.maven” – zastosowano tu wygodny styl nazewnictwa znaną z pakietów Java, jakkolwiek nazwa grupy może być zupełnie dowolna. Jednak zastosowanie kropek pozwala łatwo budować strukturę

katalogową repozytorium (zupełnie jak z nazwami pakietów) i jest zalecaną konwencją nazewnictwa grup. W powyższym przykładzie tworzony jest katalog *$M2_REPO/org/codehaus/mojo*.

- `artifactId` – najczęściej jest to nazwa, pod którą rozpoznawany (znany) jest projekt. Wraz z `groupId` tworzy unikatowy klucz oddzielający projekt od innych projektów (`artifactId` musi być unikatowy w ramach grupy, ale może się powtarzać między grupami). Opierając się na przykładzie, otrzymujemy *$M2_REPO/org/codehaus/mojo/my-project*.
- `version` – najbardziej szczegółowe określenie generowanych artefaktów projektu. Wersja powinna jednoznacznie identyfikować „z czym" mamy do czynienia, może więc być wersją projektu dla oficjalnego wydania (jak w przykładzie), może także być określona jako *timestamp* dla wydań pomniejszych. W strukturze repozytorium wersja pozwala oddzielić artefakty pochodzące z poszczególnych *buildów* (możemy jednoznacznie określić, z której wersji projektu chcemy skorzystać), nasz przykładowy POM daje *$M2_REPO/org/codehaus/mojo/my-project/1.0*.

Kolejnym elementem dotyczącym generacji artefaktów projektu jest `packaging`. Definiuje jaki tym artefaktu jest generowany przez projekt. Jeżeli nie zdefiniujemy tego elementu, domyślną wartością jest `jar`, możliwe są także `pom`, `maven-plugin`, `ejb`, `war`, `ear`, `rar`, `par`. Współrzędne w strukturze repozytorium wykorzystujące `packaging` mają następującą postać `groupId:artifactId:packaging:version`.

Czasami można spotkać także element `classifier`. Pozwala odróżnić artefakty generowane z tej samej wersji projektu (tego samego POM), jednak różniące się zawartością (np. wynikające z kompilacji dla różnych docelowych wersji Java, czy też dla odróżnienia binarów od plików źródłowych i dokumentacji, o ile te pozostałe także są publikowane jako artefakty projektu). Pozycja tej wartości we współrzędnych jest następująca: `groupId:artifactId:packaging:classifier:version`.

3.2. Powiązania POM

Jedną z podstawowych zalet Mavena jest zarządzanie powiązaniami, czyli zależnościami (także przechodnimi – *transitive dependencies*), dziedziczeniem i agregacją (projekty wielomodułowe).

3.2.1. Zależności

Lista zależności projektu stanowi jeden z elementów plików POM:

```
<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">
  ...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.0</version>
      <type>jar</type>
      <scope>test</scope>
      <optional>true</optional>
    </dependency>
    ...
  </dependencies>
  ...
</project>
```

`groupId`, `artifactId` i `version` (a także opcjonalny `classifier`) opisane są powyżej i nie wymagają tu osobnego miejsca. Pozostałe elementy definicji zależności to:

- `type` – odpowiada wartości `packaging` zdefiniowanej dla artefaktu, domyślną wartością jest oczywiście `jar`.
- `scope` – odnosi się do zakresu obowiązywania *classpath* określonego zadania (kompilacji, wykonania, testowania, itp.) oraz ograniczenia przechodniości zależności. Dostępne jest 5 zakresów:
  - `compile` (domyślny) – zależności czasu kompilacji dostępne są we wszystkich *classpath*, ponadto te zależności są propagowane do projektów zależnych,
  - `provided` – podobny do `compile`, jednak wskazuje, że oczekujemy iż JDK lub kontener dostarczy zależności w czasie wykonania (nie poprzez repozytorium Mavena), zależność dostępn jest tylko dla *classpath* kompilacji i testów, nie jest przechodni,
  - `runtime` – zakres wskazuje, że zależność nie jest wymagana do kompilacji, ale do wykonania, zależność odnosi się do *classpath* czasu wykonania i testów, nie kompilacji,
  - `test` – wskazuje, że zależność nie jest wymagana do normalnego użycia aplikacji, zależność dostępna dla kompilacji i wykonania testów,
  - `system` – zakres podobny do `provided`, jednak należy dostarczyć zależność w sposób jawny, zakłada się, że artefakt dostępny jest zawsze i nie jest wyszukiwany w repozytorium.
- `systemPath` – element używany (i wymagany) tylko w przypadku zależności o zakresie `system`.
- `optional` – określa zależność jako opcjonalną kiedy projekt sam stanowi zależność (Maven nie musi pobierać tej zależności dla zbudowania projektu zależnego od lokalnego projektu, ponieważ została już wykorzystana i nie jest wymagana).

Przy definiowaniu zależności można posłużyć się także wykluczeniami, aby „pozbyć” się zbędnych lub niewłaściwych artefaktów (które zostają na przykład podmienione na inne wersje):

```
<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">
  ...
  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-embedder</artifactId>
      <version>2.0</version>
      <exclusions>
        <exclusion>
          <groupId>org.apache.maven</groupId>
          <artifactId>maven-core</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    ...
  </dependencies>
  ...
</project>
```

3.2.2. Dziedziczenie

Jawne dziedziczenie pomiędzy POMami pozwala na ponowne użycie konfiguracji, jedynym wyznacznikiem umożliwiającym wykorzystanie dziedziczenia jest określenie wartości `packaging` jako `pom`:

```
<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>org.codehaus.mojo</groupId>
  <artifactId>my-parent</artifactId>
  <version>2.0</version>
  <packaging>pom</packaging>
</project>
```

Dziedziczenie obejmuje:

- zależności,
- deweloperów i kontrybutorów,
- listę wtyczek,
- listę raportów,
- wykonania i identyfikatory wtyczek,
- konfigurację wtyczek.

POM wykorzystujący dziedziczenie wygląda następująco:

```
<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>

  <parent>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>my-parent</artifactId>
    <version>2.0</version>
    <relativePath>../my-parent</relativePath>
  </parent>

  <artifactId>my-project</artifactId>
</project>
```

Szczególnym przypadkiem dziedziczenia jest tzw. *super POM*, zawierający podstawową konfigurację dla wszystkich POMów, które tworzymy. Zawiera podstawową konfigurację repozytoriów, wtyczek i repozytoriów wtyczek.

3.2.3. Agregacja

Agregacja pozwala tworzyć projekty wielomodułowe (agregacyjne). Moduły stanowią projekty wymienione w POMie, które wykonywane są jako grupa. Moduły (pod-projekty) są względnymi katalogami w strukturze projektu nadrzędnego:

```
<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>org.codehaus.mojo</groupId>
  <artifactId>my-parent</artifactId>
  <version>2.0</version>
  <packaging>pom</packaging>

  <modules>
    <module>my-project</module>
    <module>another-project</module>
  </modules>
</project>
```

Jeżeli nie chcemy definiować zależności pomiędzy modułami, kolejność ich wymieniania jest istotna, ponieważ zależności muszą zostać zbudowane przed projektami zależnymi od nich.

3.3. Właściwości

Podobnie jak w plikach *buildfile* Anta, POMy pozwalają na definiowanie właściwości (*properties*).

Także sposób dostępu do nich jest podobny: *${nazwa.właściwości}*. Dostępne są następujące typy właściwości:

- `env.x` – prefiks „env” odnosi się do zmiennej środowiskowej, np. *${env.PATH}* oznacza wartość ścieżki systemowej,
- `project.x` – odnosi się do wartości elementu projektu, np. *${project.version}* zwróci wartość w `<project><version>1.0</version></project>`,
- `settings.x` – zwraca wartość w odpowiednim elemencie pliku *settings.xml* (dodatkowy plik konfiguracyjny projektu, zewnętrzny względem POMu), np. *${settings.offline}* odnosi się do `<settings><offline>false</offline></settings>`,
- właściwości systemowe Java, do których w kodzie mamy dostęp przez `java.lang.System.getProperties()`, uzyskiwane są przez identyczne nazwy, np. `${java.home}`, `${user.dir}`,
- `x` – zwraca wartość zdefiniowaną w elemencie `<properties />` pliku POM.

3.4. Ustawienia buildu

Konfiguracja *buildu* jest szczegółowo opisana na stronie http://maven.apache.org/pom.html#Build_Settings i zostanie pokazana na przykładach w dalszej części laboratorium.

4. Zadania

4.1. Automatyczna generacja POMu i projektu

Pliki POM wymagają dostosowania do potrzeb projektu, jednak Maven pozwala na wygenerowanie podstawowego pliku dla dowolnego projektu. Plik ten wymaga ręcznej edycji, jednak o ile struktura projektu jest zgodna z konwencją, wystarcza on do wykonania podstawowych operacji.

Polecenie służące do generacji POMu wygląda następująco:

```
mvn archetype:create -DarchetypeGroupId=org.apache.maven.archetypes
                                -DgroupId=com.mycompany.app -DartifactId=my-app
```

Używamy tu wtyczki (*plugin*) `archetype` z celem (*goal*) `create`, podając właściwości dotyczące wartości elementów (identycznie jak w Apache Ant, czyli poprzedzając je `-D`). *Output* polecenia powinien wyglądać następująco (zależnie od tego, czy uruchamiamy Mavena po raz pierwszy na danej maszynie, może zostać pominięte pobieranie domyślnych wtyczek i podstawowych artefaktów, także skrócone poniżej):

```
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'archetype'.
[INFO] org.apache.maven.plugins: checking for updates from central
[INFO] org.codehaus.mojo: checking for updates from central
[INFO] artifact org.apache.maven.plugins:maven-archetype-plugin: checking for updates from central
Downloading: http://repo1.maven.org/maven2/org/apache/maven/plugins/maven-archetype-
plugin/2.0/maven-archetype-plugin-2.0.pom
Downloading: http://repo1.maven.org/maven2/org/apache/maven/archetype/maven-archetype/2.0/maven-
archetype-2.0.pom
Downloading: http://repo1.maven.org/maven2/org/apache/maven/maven-parent/16/maven-parent-16.pom
Downloading: http://repo1.maven.org/maven2/org/apache/apache/7/apache-7.pom
Downloading: http://repo1.maven.org/maven2/org/apache/maven/plugins/maven-archetype-
plugin/2.0/maven-archetype-plugin-2.0.jar
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Default Project
```

```
[INFO]    task-segment: [archetype:create] (aggregator-style)
[INFO] ------------------------------------------------------------------------
Downloading: http://repo1.maven.org/maven2/org/apache/maven/archetype/archetype-
common/2.0/archetype-common-2.0.pom
Downloading: http://repo1.maven.org/maven2/net/sourceforge/jchardet/jchardet/1.0/jchardet-1.0.pom
Downloading: http://repo1.maven.org/maven2/dom4j/dom4j/1.6.1/dom4j-1.6.1.pom
Downloading: http://repo1.maven.org/maven2/xml-apis/xml-apis/1.0.b2/xml-apis-1.0.b2.pom
Downloading: http://repo1.maven.org/maven2/jdom/jdom/1.0/jdom-1.0.pom
Downloading: http://repo1.maven.org/maven2/org/apache/maven/maven-model/2.0.8/maven-model-2.0.8.pom
Downloading: http://repo1.maven.org/maven2/org/apache/maven/maven/2.0.8/maven-2.0.8.pom
.
.
.
Downloading: http://repo1.maven.org/maven2/commons-io/commons-io/1.4/commons-io-1.4.jar
Downloading: http://repo1.maven.org/maven2/org/codehaus/plexus/plexus-velocity/1.1.8/plexus-
velocity-1.1.8.jar
[INFO] [archetype:create {execution: default-cli}]
[WARNING] This goal is deprecated. Please use mvn archetype:generate instead
[INFO] Defaulting package to group ID: com.mycompany.app
[INFO] artifact org.apache.maven.archetypes:maven-archetype-quickstart: checking for updates from
central
Downloading: http://repo1.maven.org/maven2/org/apache/maven/archetypes/maven-archetype-
quickstart/1.1/maven-archetype-quickstart-1.1.jar
Downloading: http://repo1.maven.org/maven2/org/apache/maven/archetypes/maven-archetype-
quickstart/1.1/maven-archetype-quickstart-1.1.pom

[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-
quickstart:RELEASE
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: packageName, Value: com.mycompany.app
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: my-app
[INFO] Parameter: basedir, Value: /Volumes/Data/Users/jacenty/Desktop/KIS/zaje?cia/PIO/maven lab
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] ********************* End of debug info from resources from generated POM ******************
[INFO] project created from Old (1.x) Archetype in dir:
/Volumes/Data/Users/jacenty/Desktop/KIS/zaje?cia/PIO/maven lab/my-app
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 59 seconds
[INFO] Finished at: Mon Jan 03 02:53:29 CET 2011
[INFO] Final Memory: 14M/81M
[INFO] ------------------------------------------------------------------------
```

W efekcie otrzymujemy gotową strukturę projektu (nazwy katalogów zgodne z podanymi przez nas `groupId` i `artifactId`):

```
my-app
  |-pom.xml
  |-src
      |-main
      |   |-java
      |       |-com
      |          |-mycompany
      |              |-app
      |                  |-App.java
      |-test
          |-java
              |-com
                 |-mycompany
                     |-app
                        |-AppTest.java
```

A wygenerowany plik POM zawiera:

```
<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.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
```

```
  <name>my-app</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

4.2. Zdalne repozytorium

Przeglądanie domyślnego repozytorium Mavena możliwe jest za pośrednictwem strony http://search.maven.org/. Należy zapoznać się z jego strukturą i rozpoznać grupy, artefakty i wersje, a także typy artefaktów.

4.3. Lokalne repozytorium

Maven pobiera artefakty i wtyczki do lokalnego repozytorium zlokalizowanego w katalogu domowym użytkownika w podkatalogu *.m2* (*~/.m2*). Należy zapoznać się z jego strukturą i porównać ze strukturą repozytorium zdalnego. Powinny być identyczne, jednak repozytorium lokalne zawiera jedynie artefakty potrzebne do wykonania *buildów* (czyli tylko takie, które chociaż raz zostały użyte). Dzięki temu nie istnienie potrzeba pobierania ich za każdym razem, wykorzystywany jet lokalny *cache*.

4.4. Prosty projekt

Używając automatycznie wygenerowanego projektu z polecenia 4.1, należy:

1. Wykonać kompilację kodu (`mvn compile`) i sprawdzić, gdzie znajdują się utworzone pliki *.class*.
2. Wykonać pakietowanie (`mvn package`) i sprawdzić, gdzie znajdują się wygenerowane artefakty. Proszę zwrócić uwagę na automatycznie wykonane test i ich raporty.
3. Wyczyścić projekt (`mvn clean`) i sprawdzić, co zostało usunięte.
4. Wygenerować artefakty bez pakietowania (`mvn jar:jar`) i porównać efekt z pełnym pakietowaniem.
5. Zainstalować artefakty w lokalnym repozytorium (`mvn install`), aby stały się dostępne dla Mavena. Należy sprawdzić, czy artefakty przykładowego projektu znajdują się w odpowiednim miejscu lokalnego repozytorium.
6. Uruchomić przykładową aplikację w oparciu o zainstalowane artefakty (`mvn exec:java -Dexec.mainClass=com.mycompany.app.App`).

4.5. Projekt wielomodułowy

Należ pobrać przykładowy projekt wielomodułowy znajdujący się po adresem http://jacenty.kis.p.lodz.pl/pio/maven-multi.zip i rozpakować archiwum na lokalnym dysku twardym. Następnie:

1. Zapoznać się z zawartością pliku POM, rozpoznać zależności i moduły projektu, a także fizyczną organizację plików projektu.
2. Wygenerować artefakty projektu głównego. Proszę zwrócić uwagę na automatyczną kompilację i generację artefaktów poszczególnych modułów.
3. Zainstalować artefakty projekty głównego i sprawdzić, czy zostały umieszczone w lokalnym

repozytorium.
4. Uruchomić aplikację z projektu głównego.