Faculty of Electronics and Information Technology
WARSAW UNIVERSITY OF TECHNOLOGY

Programowanie Aplikacyjne (PAP)

Maven

Julian Myrcha
Institute of Computer Science
26.02.2025

WUT

Maven

- Maven - narzędzie automatyzujące budowę oprogramowania na platformę Java.

Maven

- Maven - narzędzie automatyzujące budowę oprogramowania na platformę Java.
- Poszczególne funkcje Mavena realizowane są poprzez wtyczki

Maven

- Maven - narzędzie automatyzujące budowę oprogramowania na platformę Java.
- Poszczególne funkcje Mavena realizowane są poprzez wtyczki
- Wtyczki są automatycznie pobierane przy ich pierwszym wykorzystaniu.

Maven

- Maven - narzędzie automatyzujące budowę oprogramowania na platformę Java.
- Poszczególne funkcje Mavena realizowane są poprzez wtyczki
- Wtyczki są automatycznie pobierane przy ich pierwszym wykorzystaniu.
- Plik określający sposób budowy aplikacji nosi nazwę POM-u (ang. Project Object Model)

Maven

- Maven - narzędzie automatyzujące budowę oprogramowania na platformę Java.
- Poszczególne funkcje Mavena realizowane są poprzez wtyczki
- Wtyczki są automatycznie pobierane przy ich pierwszym wykorzystaniu.
- Plik określający sposób budowy aplikacji nosi nazwę POM-u (ang. Project Object Model)
- W pliku mamy różne cele, które możemy budować

Maven

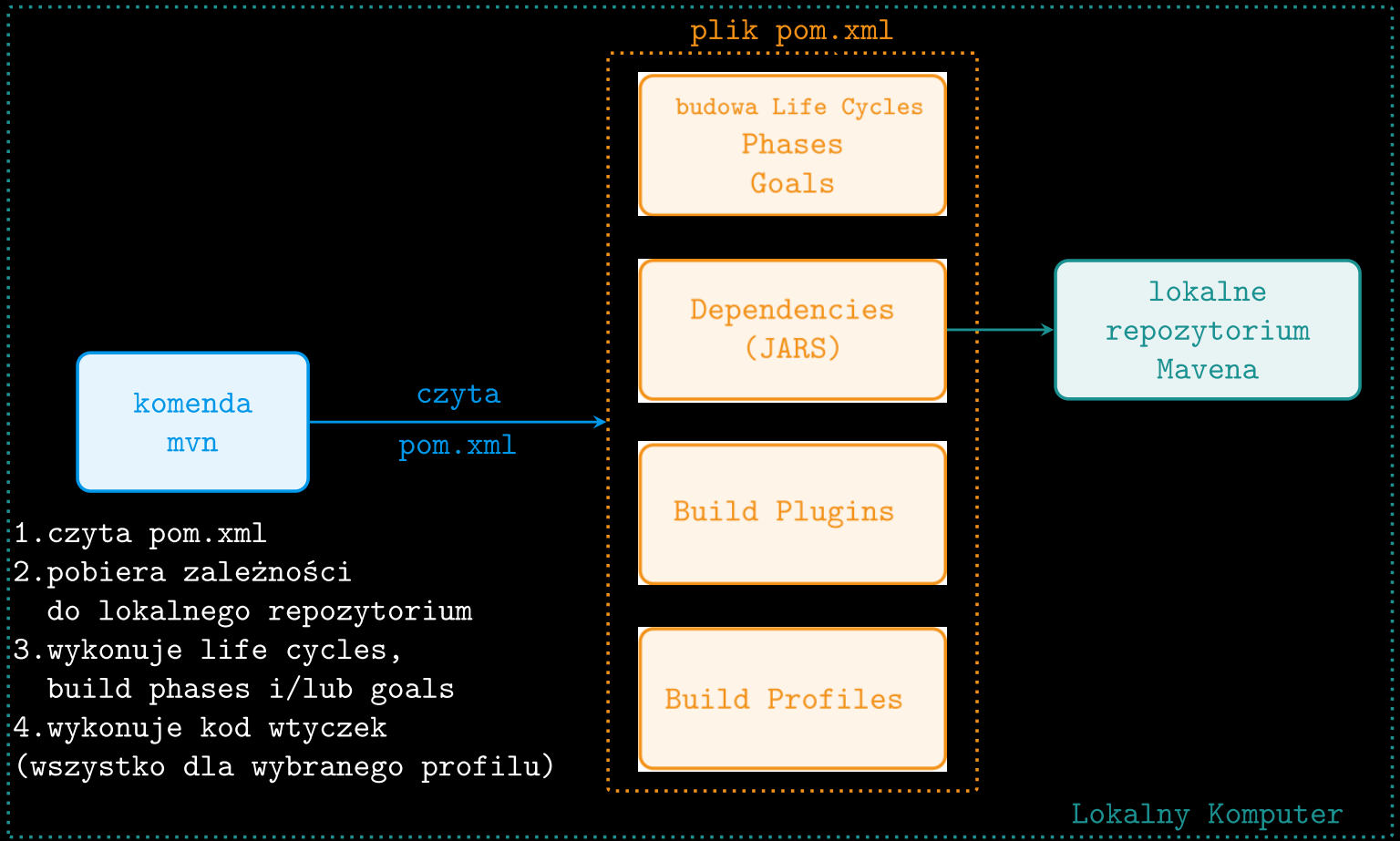
- Maven - narzędzie automatyzujące budowę oprogramowania na platformę Java.
- Poszczególne funkcje Mavena realizowane są poprzez wtyczki
- Wtyczki są automatycznie pobierane przy ich pierwszym wykorzystaniu.
- Plik określający sposób budowy aplikacji nosi nazwę POM-u (ang. Project Object Model)
- W pliku mamy różne cele, które możemy budować
- Budowa celu może uruchomić budowę innego celu

Maven

- Maven - narzędzie automatyzujące budowę oprogramowania na platformę Java.
- Poszczególne funkcje Mavena realizowane są poprzez wtyczki
- Wtyczki są automatycznie pobierane przy ich pierwszym wykorzystaniu.
- Plik określający sposób budowy aplikacji nosi nazwę POM-u (ang. Project Object Model)
- W pliku mamy różne cele, które możemy budować
- Budowa celu może uruchomić budowę innego celu
- Maven dociąga z internetu brakujące biblioteki w zadanych wersjach

Maven

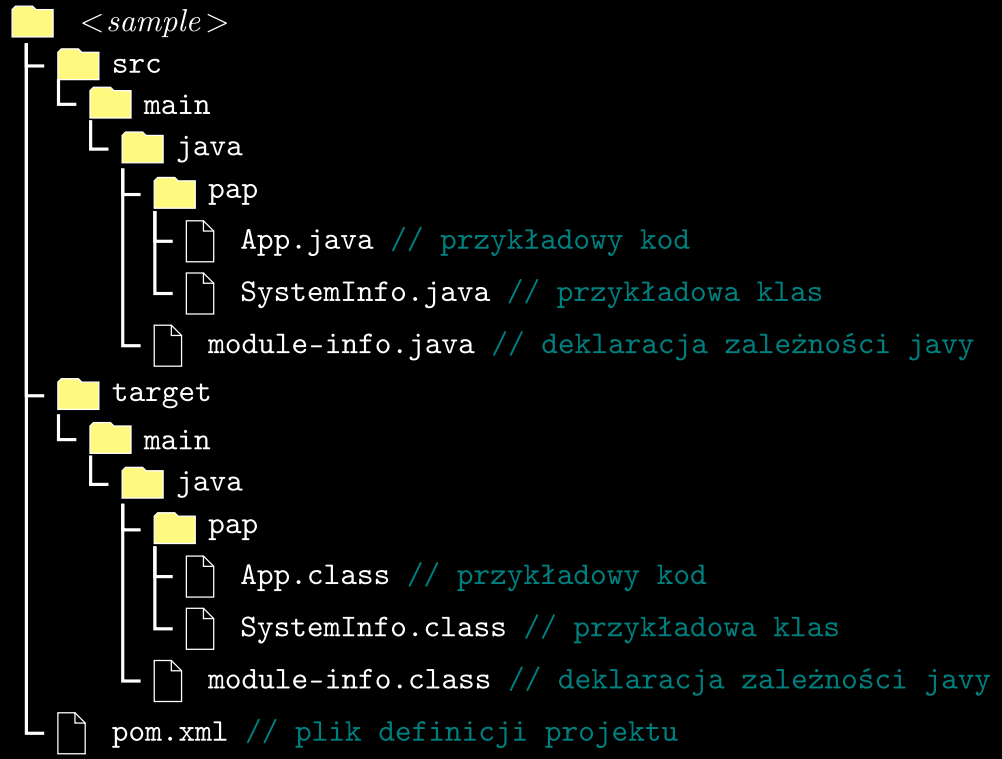
- Maven - narzędzie automatyzujące budowę oprogramowania na platformę Java.
- Poszczególne funkcje Mavena realizowane są poprzez wtyczki
- Wtyczki są automatycznie pobierane przy ich pierwszym wykorzystaniu.
- Plik określający sposób budowy aplikacji nosi nazwę POM-u (ang. Project Object Model)
- W pliku mamy różne cele, które możemy budować
- Budowa celu może uruchomić budowę innego celu
- Maven dociąga z internetu brakujące biblioteki w zadanych wersjach
- W katalogu `/.m2` tworzony jest cache ze ściągniętymi bibliotekami - jak mamy wiele projektów to automatycznie reużywamy i oszczędzamy zarówno łącze jak i miejsce na dysku


plik pom.xml
budowa Life Cycles
Phases
Goals
Dependencies
(JARS)
Build Plugins
Build Profiles
lokalne
repozytorium
Mavena
komenda
mvn
czyta
pom.xml
1.czyta pom.xml
2.pobiera zależności
do lokalnego repozytorium
3.wykonuje life cycles,
build phases i/lub goals
4.wykonuje kod wtyczek
(wszystko dla wybranego profilu)
Lokalny Komputer

- w katalogu /.m2/repository są pakiety sciągane przez mavena

```
jmy@ryzen20:$ ls -1 .m2/repository
antlr
asm
backport-util-concurrent
classworlds
com
commons-cli
commons-codec
commons-collections
commons-io
commons-lang
jdom
junit
net
org
xmlpull
xpp3
```

- otworzenie projektu w VS Code komiluje go - powstaje folder target



- mamy do dyspozycji komendy mavena

validate - sprawdzenie, czy projekt jest poprawny i czy wszystkie niezbędne informacje zostały określone

- mamy do dyspozycji komendy mavena

validate -
compile - kod źródłowy jest kompilowany

- mamy do dyspozycji komendy mavena

validate -
compile -
test - przeprowadzane są testy jednostkowe

- mamy do dyspozycji komendy mavena

validate -
compile -
test -
package - budowana jest paczka dystrybucyjna

- mamy do dyspozycji komendy mavena

validate -
compile -
test -
package -
integration -test zbudowany projekt umieszczany jest w środowisku testowym, gdzie przeprowadzane są testy integracyjne

- mamy do dyspozycji komendy mavena

validate -
compile -
test -
package -
integration -test
verify - sprawdzenie, czy paczka jest poprawna

- mamy do dyspozycji komendy mavena

validate -
compile -
test -
package -
integration -test
verify -
install - paczka umieszczana jest w repozytorium lokalnym - może być używana przez inne projekty jako zależność

- mamy do dyspozycji komendy mavena

validate -
compile -
test -
package -
integration -test
verify -
install -
deploy - paczka umieszczana jest w repozytorium zdalnym (opublikowana)

- repozytorium nam spuchło zanim zaczęliśmy kompilować

```
ls .m2/repository
antlr                       commons-cli          doxia      org
aopalliance                 commons-codec        javax      oro
asm                         commons-collections  jdom       plexus
avalon-framework            commons-digester     joda-time  sslext
backport-util-concurrent    commons-httpclient   jtidy      xerces
ch                          commons-io           junit      xml-apis
classworlds                 commons-lang         log4j      xmlpull
com                         commons-logging      logkit     xpp3
commons-beanutils           commons-validator    nekohtml
commons-chain               dom4j                net
```

- W katalogu głównym wydajemu komendę:

```
# to jedna komenda wiec jedna  linia
mvn archetype:generate -DgroupId=atj.tutorial -DartifactId=tutorial-app
  ↪ -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

- Powstał katalog `tutorial-app`

```
tutorial-app/pom.xml
tutorial-app/src/test/java/atj/tutorial/AppTest.java
tutorial-app/src/main/java/atj/tutorial/App.java
```

- opisuje budowaną aplikację
- zawiera zależności

```xml
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>atj.tutorial</groupId>
  <artifactId>tutorial-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>tutorial-app</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Maven z linii komend (3)

- W katalogu, w którym jest plik pom.xml wydajemy komendę:

```
mvn package
```

- Wynik kompilacji ląduje w katalogu target
- Uruchomienie wyniku:

```
java -cp target/tutorial-app-1.0-SNAPSHOT.jar atj.tutorial.App
```

- Można podać kilka faz i celów po kolei, zamiast:

```
mvn clean
mvn dependency:copy-dependencies
mvn package
```

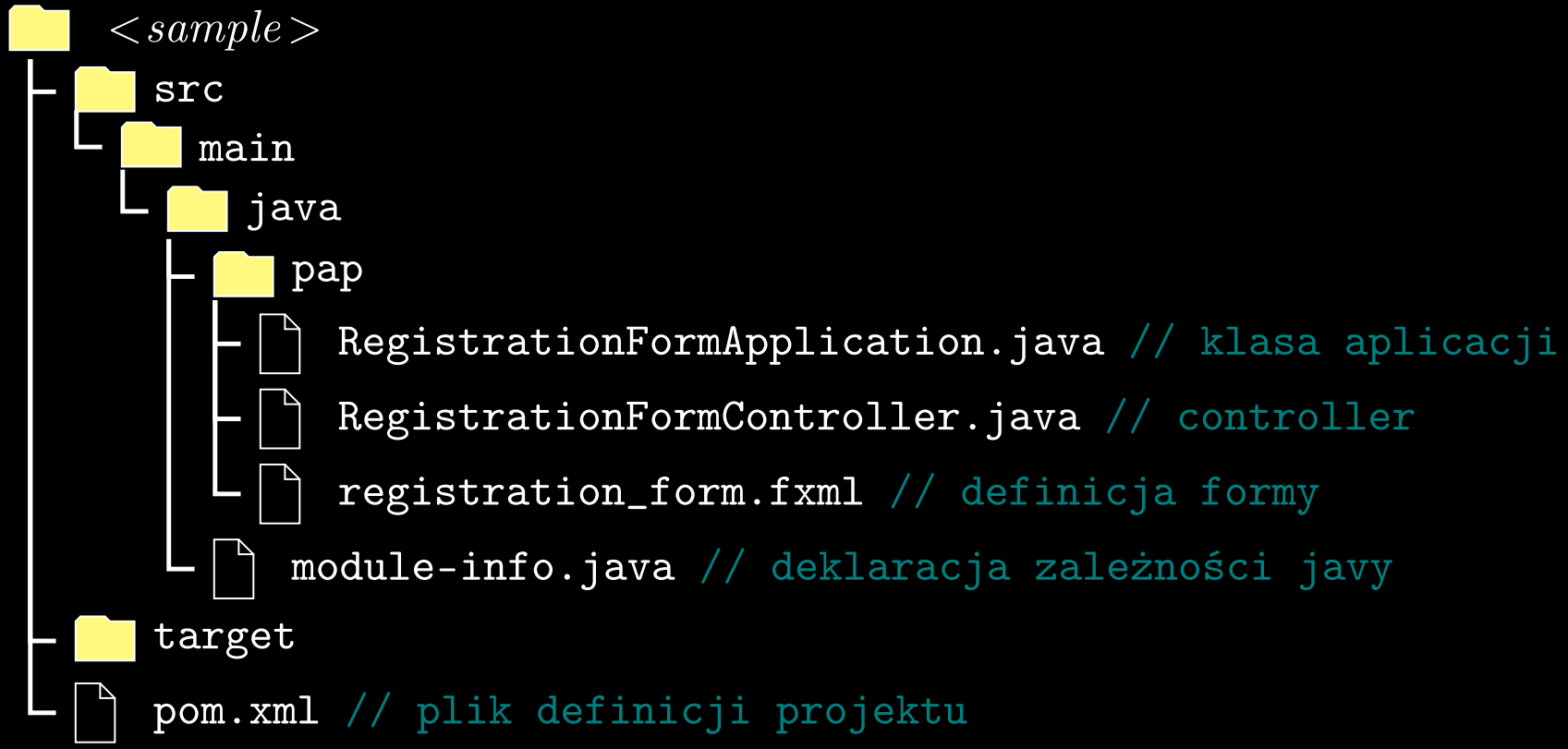
- możemy wywołać:

```
mvn clean dependency:copy-dependencies package
```

- Dokumentacja projektu (do katalogu target/site):

```
mvn site
```

- https://gluonhq.com/products/scene-builder/



plik ./src/main/java/pap/RegistrationFormController.java

```java
package pap;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.stage.Window;

public class RegistrationFormController {
    @FXML
    private TextField nameField;

    @FXML
    private TextField emailField;

    @FXML
    private PasswordField passwordField;

    @FXML
    private Button submitButton;

    @FXML
    protected void handleSubmitButtonAction(ActionEvent event) {
        Window owner = submitButton.getScene().getWindow();
        if (nameField.getText().isEmpty()) {
            AlertHelper.showAlert(Alert.AlertType.ERROR, owner, "Form Error!", "Please enter your name");
            return;
        }
        if (emailField.getText().isEmpty()) {
            AlertHelper.showAlert(Alert.AlertType.ERROR, owner, "Form Error!", "Please enter your email id");
            return;
        }
        if (passwordField.getText().isEmpty()) {
            AlertHelper.showAlert(Alert.AlertType.ERROR, owner, "Form Error!", "Please enter a password");
            return;
        }
```

```
39
40          AlertHelper.showAlert(Alert.AlertType.CONFIRMATION, owner, "Registration Successful!",
41                  "Welcome " + nameField.getText());
42      }
43  }
```

plik ./src/main/java/module-info.java

```
module pap {
    requires javafx.controls;
    requires javafx.fxml;
    requires javafx.graphics;
    opens pap;
}
```

plik ./src/main/java/pap/registration_form.fxml

```
<?import javafx.scene.layout.GridPane?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.control.Button?>
<GridPane fx:controller="pap.RegistrationFormController"
          xmlns:fx="http://javafx.com/fxml" alignment="center"
          hgap="10" vgap="10">
    <padding><Insets top="40" right="40" bottom="40" left="40"/></padding>
    <columnConstraints>
        <ColumnConstraints minWidth="100" prefWidth="100"
                           maxWidth="Infinity" halignment="RIGHT">
        </ColumnConstraints>
        <ColumnConstraints minWidth="200" prefWidth="200"
                           maxWidth="Infinity" hgrow="ALWAYS">
        </ColumnConstraints>
    </columnConstraints>

    <!-- Add Header Label -->
    <Label text="Registration Form (FXML)" GridPane.columnIndex="0"
           GridPane.rowIndex="0" GridPane.columnSpan="2"
           GridPane.rowSpan="1" GridPane.halignment="CENTER" >
        <font>
            <Font name="Arial" size="24" ></Font>
        </font>
        <GridPane.margin>
            <Insets top="20" right="0" bottom="20" left="0"></Insets>
        </GridPane.margin>
    </Label>

    <!-- Add Name Label -->
    <Label text="Full Name : " GridPane.columnIndex="0"
           GridPane.rowIndex="1" >
    </Label>
```

```
    <!-- Add Name Text Field -->
    <TextField fx:id="nameField" prefHeight="40"
               GridPane.columnIndex="1" GridPane.rowIndex="1"/>

    <!-- Add Email Label -->
    <Label text="Email ID : " GridPane.columnIndex="0"
           GridPane.rowIndex="2" >
    </Label>
    <!-- Add Email Text Field -->
    <TextField fx:id="emailField" prefHeight="40"
               GridPane.columnIndex="1" GridPane.rowIndex="2"/>

    <!-- Add Password Label -->
    <Label text="Password : " GridPane.columnIndex="0"
           GridPane.rowIndex="3" >
    </Label>
    <!-- Add Password Field -->
    <PasswordField fx:id="passwordField" prefHeight="40"
                   GridPane.columnIndex="1" GridPane.rowIndex="3"/>

    <!-- Add Submit Button -->
    <Button fx:id="submitButton" text="Submit"
            prefWidth="100" prefHeight="40" defaultButton="true"
            GridPane.columnIndex="0" GridPane.rowIndex="4"
            GridPane.columnSpan="2" GridPane.rowSpan="1"
            GridPane.halignment="CENTER"
            onAction="#handleSubmitButtonAction">
        <GridPane.margin>
            <Insets top="20" right="0" bottom="20" left="0"></Insets>
        </GridPane.margin>
    </Button>
</GridPane>
```

- pobieramy plik apache-maven-3.6.3-bin.zip ze strony https://maven.apache.org/download.cgi

- pobieramy plik apache-maven-3.6.3-bin.zip ze strony https://maven.apache.org/download.cgi
- rozpakowujemy go do katalogu c:/ *** /java/apache-maven-3.6.3

- pobieramy plik apache-maven-3.6.3-bin.zip ze strony https://maven.apache.org/download.cgi
- rozpakowujemy go do katalogu c:/ *** /java/apache-maven-3.6.3
- ustawiamy zmienne środowiskowe

```
export M2_HOME=/usr/local/apache-maven/apache-maven-3.3.9
export M2=$M2_HOME/bin
export MAVEN_OPTS=-Xms256m -Xmx512m
export PATH=$M2:$PATH                    # to przyklady linuksowe!!!
```

- pobieramy plik apache-maven-3.6.3-bin.zip ze strony https://maven.apache.org/download.cgi
- rozpakowujemy go do katalogu c:/ *** /java/apache-maven-3.6.3
- ustawiamy zmienne środowiskowe
- w nowym terminalu sprawdzamy wersję zainstalowanego nasrzędzia:

```
1 mvn -version
```

- Ctrl+Shift+P Java:Create Java project

- Ctrl+Shift+P Java:Create Java project
- Maven-create from archetype

- Ctrl+Shift+P Java:Create Java project
- Maven-create from archetype
- more ...

- Ctrl+Shift+P Java:Create Java project
- Maven-create from archetype
- more ...
- javafx-archetype-simple org.openjfx (version 0.0.0.5)

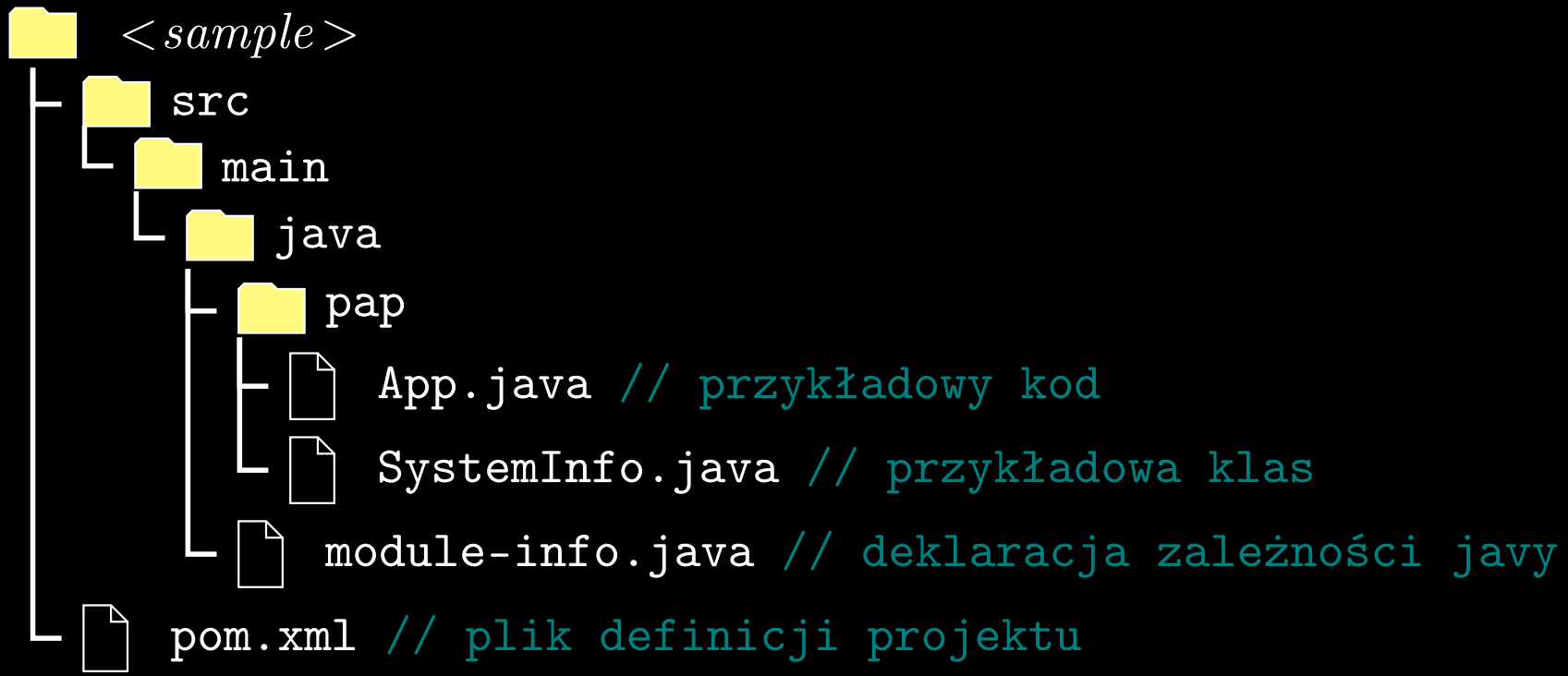
- Ctrl+Shift+P Java:Create Java project
- Maven-create from archetype
- more ...
- javafx-archetype-simple org.openjfx (version 0.0.0.5)
- select destination folder

- Ctrl+Shift+P Java:Create Java project
- Maven-create from archetype
- more ...
- javafx-archetype-simple org.openjfx (version 0.0.0.5)
- select destination folder
- define value for property groupID: edu.iipw.pap
  - groupId jednoznacznie identyfikuje Twój projekt we wszystkich projektach
  - Identyfikator grupy powinien być zgodny z regułami nazw pakietów Javy.
  - Oznacza to, że zaczyna się od odwróconej nazwy domeny, którą kontrolujesz.
  - Na przykład org.apache.maven, org.apache.commons

- Ctrl+Shift+P Java:Create Java project
- Maven-create from archetype
- more ...
- javafx-archetype-simple org.openjfx (version 0.0.0.5)
- select destination folder
- define value for property groupID: edu.iipw.pap
- define value for property artifactID: sample

- Ctrl+Shift+P Java:Create Java project
- Maven-create from archetype
- more ...
- javafx-archetype-simple org.openjfx (version 0.0.0.5)
- select destination folder
- define value for property groupID: edu.iipw.pap
- define value for property artifactID: sample
- define value for property version: 1.0-SNAPSHOT

- Ctrl+Shift+P Java:Create Java project
- Maven-create from archetype
- more ...
- javafx-archetype-simple org.openjfx (version 0.0.0.5)
- select destination folder
- define value for property groupID: edu.iipw.pap
- define value for property artifactID: sample
- define value for property version: 1.0-SNAPSHOT
- define value for property package: pap


<sample>
src
main
java
pap
App.java // przykładowy kod
SystemInfo.java // przykładowa klas
module-info.java // deklaracja zależności javy
pom.xml // plik definicji projektu

- plik pom.xml

```xml
<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/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>edu.pw.pap</groupId>
    <artifactId>sample</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>13</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.0</version>
                <configuration>
                    <release>11</release>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.openjfx</groupId>
                <artifactId>javafx-maven-plugin</artifactId>
                <version>0.0.4</version>
                <configuration>
                    <mainClass>pap.App</mainClass>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

- plik pom.xml

```xml
<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/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>edu.pw.pap</groupId>
    <artifactId>sample</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>
    <dependencies>
      ...
    </dependencies>
    <build>
        <plugins>
          ...
        </plugins>
    </build>
</project>
```

- plik pom.xml - zależności od innych pakietów zdefiniowanych za pomocą plików pom.xml w trakcie wykonania programu
- odwołujemy się do repozytorium mavena https://mvnrepository.com/repos/central gdzie jest WSZYSTKO
- pakiety używają tej samej konwencji nazewnictwa

  groupId - organizacja odpowiedzialna za pakiet

  artifactId - nazwa pakietu

  version - jego wersja

```
...
<dependencies>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>13</version>
    </dependency>
</dependencies>
...
```

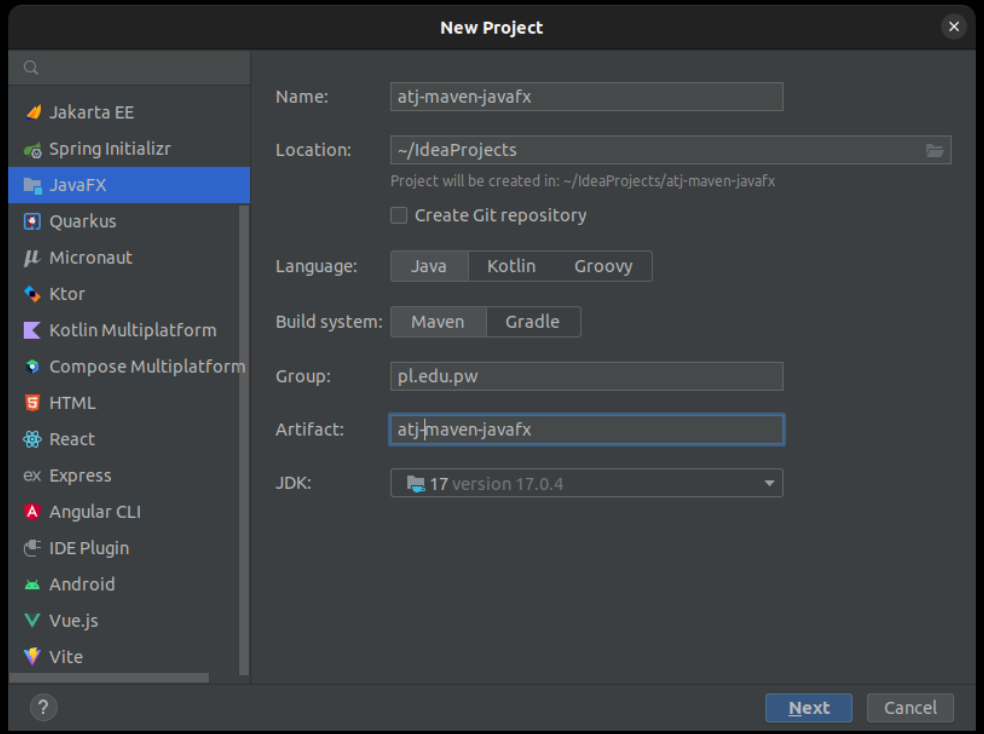
- plik pom.xml - zależności od innych pakietów zdefiniowanych za pomocą plików pom.xml w trakcie wykonania programu
- dopisując zależności ściągamy potrzebne biblioteki

```
    ...
    <dependencies>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>13</version>
        </dependency>
        <dependency>
             <groupId>org.openjfx</groupId>
             <artifactId>javafx-fxml</artifactId>
             <version>13</version>
        </dependency>
    </dependencies>
    ...
```

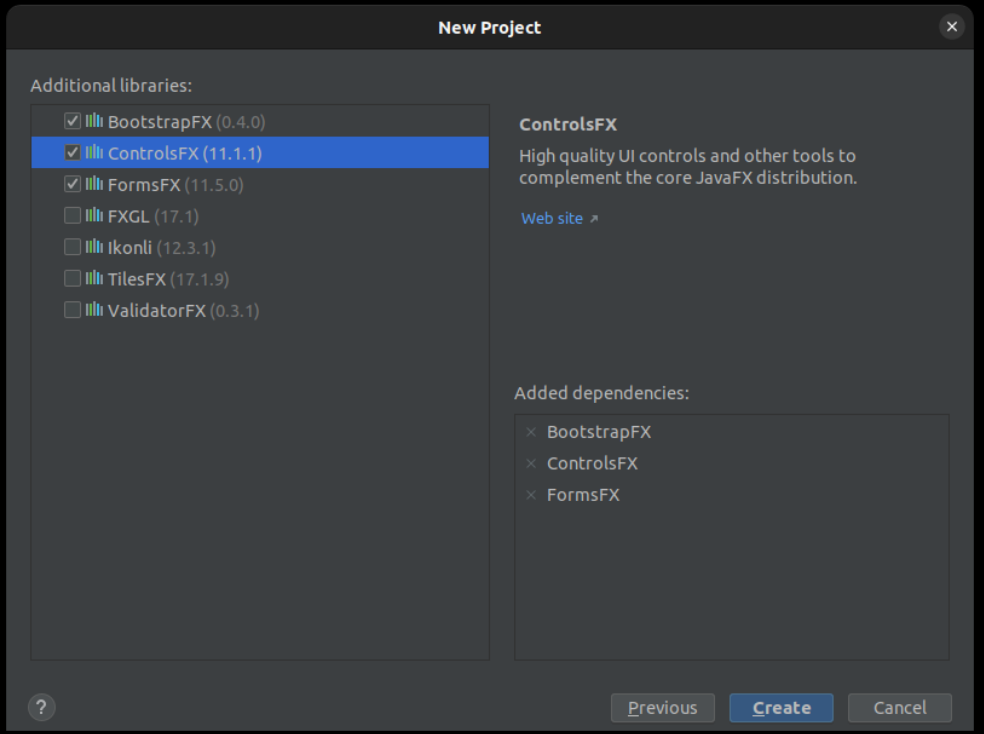
- plik pom.xml - zależności od innych pakietów zdefiniowanych za pomocą plików pom.xml w trakcie budowy programu

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.0</version>
            <configuration>
                <release>11</release>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-maven-plugin</artifactId>
            <version>0.0.4</version>
            <configuration>
                <mainClass>pap.App</mainClass>
            </configuration>
        </plugin>
    </plugins>
</build>
```

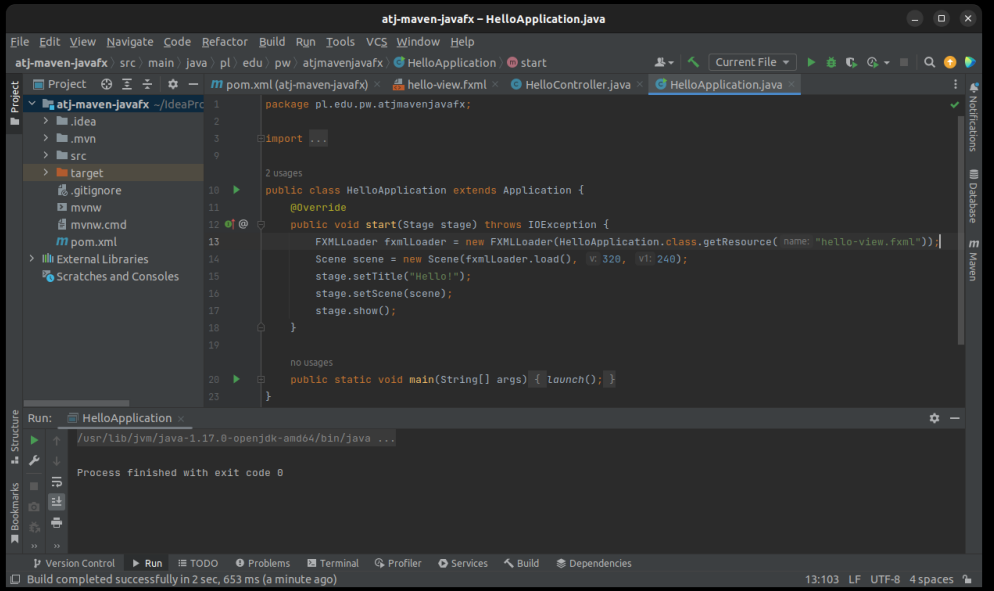
- utworzenie projektu wykorzystującego Mavena
- dodanie zależności
- utworzony projekt
- utworzony kontroler
- maven



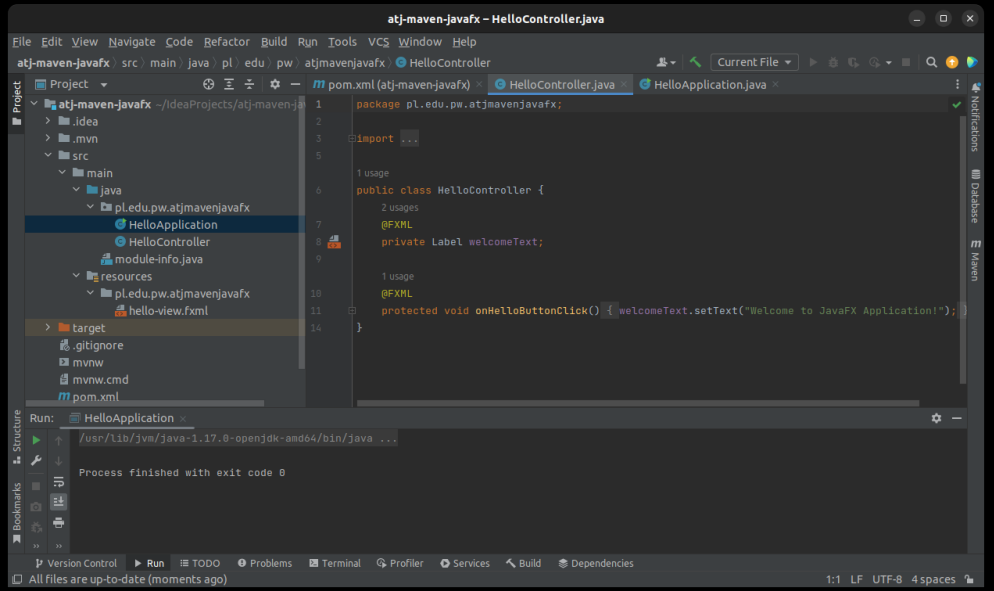
- utworzenie projektu wykorzystującego Mavena
- dodanie zależności
- utworzony projekt
- utworzony kontroler
- maven



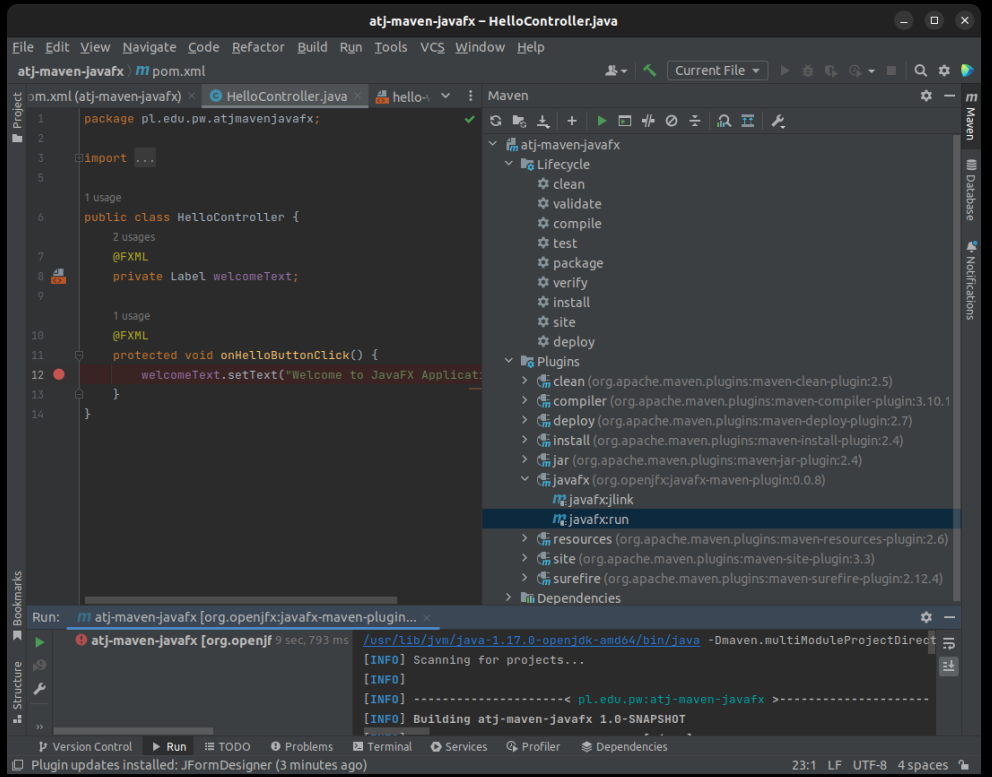
- utworzenie projektu wykorzystującego Mavena
- dodanie zależności
- utworzony projekt
- utworzony kontroler
- maven



- utworzenie projektu wykorzystującego Mavena
- dodanie zależności
- utworzony projekt
- utworzony kontroler
- maven



- utworzenie projektu wykorzystującego Mavena
- dodanie zależności
- utworzony projekt
- utworzony kontroler
- maven



- Apache Ant (”Another Neat Tool”)
  - Pliki konfiguracji Ant są napisane w XML i zgodnie z konwencją nazywają się **build.xml**.
  - uruchamiamy `cele` (targets) na przykład: `ant compile`
  - Główną zaletą Anta jest jego elastyczność
  - Zarządzanie zależnościami dodane później za pomocą Apache Ivy

```xml
<project>
  <target name="clean">
      <delete dir="classes" />
  </target>
  <target name="compile" depends="clean">
        <mkdir dir="classes" />
        <javac srcdir="src" destdir="classes" />
  </target>
  <target name="jar" depends="compile">
    <mkdir dir="jar" />
    <jar destfile="jar/HelloWorld.jar" basedir="classes">
        <manifest>
          <attribute name="Main-Class"
              value="antExample.HelloWorld" />
        </manifest>
    </jar>
  </target>
  <target name="run" depends="jar">
    <java jar="jar/HelloWorld.jar" fork="true" />
  </target>
</project>
```

- Apache Ant (”Another Neat Tool”)
- maven
  - Maven opiera się na konwencjach i zapewnia predefiniowane polecenia (cele)
  - Maven można uznać za framework do wykonywania wtyczek

```xml
<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>pap</groupId>
    <artifactId>mavenExample</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <description>Maven example</description>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

- Apache Ant (”Another Neat Tool”)
- maven
  - Maven opiera się na konwencjach i zapewnia predefiniowane polecenia (cele)
  - Maven można uznać za framework do wykonywania wtyczek

```
+---src
|   +---main
|   |   +---java
|   |   |   \---com
|   |   |       \---pap
|   |   |           \---maven
|   |   |                   HelloWorld.java
|   |   |
|   |   \---resources
|   \---test
|       +---java
|       \---resources
```

- Apache Ant (”Another Neat Tool”)
- maven
- gradle
  - Gradle polega na tym, że nie używa plików XML - ma DSL oparte na Groovy lub Kotlin
  - Wtyczka Java pozwala nam kompilować kod Java i inne cenne funkcje.

```
apply plugin: 'java'
repositories {
  mavenCentral()
}
jar {
  baseName = 'gradleExample'
  version = '0.0.1-SNAPSHOT'
}
dependencies {
  testImplementation 'junit:junit:4.12'
}
```

- Apache Ant (”Another Neat Tool”)
- maven
- gradle
  - Gradle polega na tym, że nie używa plików XML - ma DSL oparte na Groovy lub Kotlin
  - Wtyczka Java pozwala nam kompilować kod Java i inne cenne funkcje.

własne kroki budowania

```
task copyDependencies(type: Copy) {
   from configurations.compile
   into 'dependencies'
}
```

i teraz:

```
gradle copyDependencies
```

instalacja (ubuntu)

- pobieramy ze strony:

```
sudo mkdir /opt/gradle
sudo mv gradle-7.2 /opt/gradle
sudo ln /opt/gradle/gradle-7.2 /opt/gradle/latest
sudo nano /etc/profile.d/gradle.sh
>export GRADLE_HOME=/opt/gradle/latest
>export PATH=${GRADLE_HOME}/bin:${PATH}
sudo chmod +x /etc/profile.d/gradle.sh
source /etc/profile.d/gradle.sh
```

```
jmy@xps /media/jmy/a2000/Projects/gradle-tutorial $ gradle init
Starting a Gradle Daemon (subsequent builds will be faster)

Select type of project to generate:
  1: basic
  2: application
  3: library
  4: Gradle plugin
Enter selection (default: basic) [1..4] 2

Select implementation language:
  1: C++
  2: Groovy
  3: Java
  4: Kotlin
  5: Scala
  6: Swift
Enter selection (default: Java) [1..6] 3

Split functionality across multiple subprojects?:
  1: no - only one application project
  2: yes - application and library projects
Enter selection (default: no - only one application project)
↪  [1..2] 1

Select build script DSL:
  1: Groovy
  2: Kotlin
Enter selection (default: Groovy) [1..2] 1

Select test framework:
  1: JUnit 4
  2: TestNG
  3: Spock
  4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 4

Project name (default: gradle-tutorial): hello
Source package (default: hello):

> Task :init
Get more help with your project:
https://docs.gradle.org/7.2/samples/sample_building_java_applications.html

BUILD SUCCESSFUL in 47s
```

```
package hello;

public class HelloWorld {
  public static void main(String[] args) {
  Greeter greeter = new Greeter();
  System.out.println(greeter.sayHello());
  }
}
```

```
mkdir -p src/main/java/hello
|- src
    |-main
    |   |- java
    |       |- hello
    |           |- HelloWorld.java
    |           |- Greeter.java
    |- main
        |- test
            |- hello
                |- AppTest.java
```

```java
package hello;

public class Greeter {
  public String sayHello() {
  return "Hello world!";
  }
}
```

```
mkdir -p src/main/java/hello
|- src
    |-main
    |   |- java
    |       |- hello
    |           |- HelloWorld.java
    |           |- Greeter.java
    |- main
        |- test
            |- hello
                |- AppTest.java
```

```java
/*
 * This Java source file was generated by the Gradle 'init' task.
 */
package hello;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class AppTest {
    @Test void appHasAGreeting() {
        Greeter classUnderTest = new Greeter();
        assertNotNull(classUnderTest.sayHello(), "app should have
        ↪  a greeting");
    }
}
```

```
mkdir -p src/main/java/hello
|- src
    |-main
    |    |- java
    |        |- hello
    |            |- HelloWorld.java
    |            |- Greeter.java
    |- main
        |- test
            |- hello
                |- AppTest.java
```

- gradle jest systemem budowy programów (nie tylko java)
  - modyfikacja pliku app/build.gradle
- podobna funcjonalność do mavena
- wykorzystuje repozytoria mavena
  - budowa projektu
  - wykonanie projektu

```
plugins {
    // Apply the application plugin to add
    // support for building a CLI application in Java.
    id 'application'
}

repositories {
    // Use Maven Central for resolving dependencies.
    mavenCentral()
}

dependencies {
    // Use JUnit Jupiter for testing.
    testImplementation 'org.junit.jupiter:junit-jupiter:5.7.2'

    // This dependency is used by the application.
    implementation 'com.google.guava:guava:30.1.1-jre'
}

application {
    // Define the main class for the application.
    mainClass = 'hello.HelloWorld'
}

tasks.named('test') {
    useJUnitPlatform()
}

jar {
    manifest {
        attributes('Main-Class': 'hello.HelloWorld')
    }
}
```

- gradle jest systemem budowy programów (nie tylko java)
  - modyfikacja pliku app/build.gradle
- podobna funcjonalność do mavena
- wykorzystuje repozytoria mavena
  - budowa projektu
  - wykonanie projektu

```
jmy@xps ~/Projects $ gradle

> Task :help

Welcome to Gradle 7.2.

To run a build, run gradle <task> ...
To see a list of available tasks, run gradle tasks
To see a list of command-line options, run gradle --help
To see more detail about a task, run gradle help --task <task>
For troubleshooting, visit https://help.gradle.org

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed
jmy@xps ~/Projects $
```

- gradle jest systemem budowy programów (nie tylko java)
  - modyfikacja pliku app/build.gradle
- podobna funcjonalność do mavena
- wykorzystuje repozytoria mavena
  - budowa projektu
  - wykonanie projektu

```
jmy@xps ~/Projects $ gradle build

BUILD SUCCESSFUL in 2s
7 actionable tasks: 2 executed, 5 up-to-date
jmy@xps ~/Projects $
```

- gradle jest systemem budowy programów (nie tylko java)
  - modyfikacja pliku app/build.gradle
- podobna funcjonalność do mavena
- wykorzystuje repozytoria mavena
  - budowa projektu
  - wykonanie projektu

```
jmy@xps ~/Projects $ java -jar app/build/libs/app.jar
Hello world!
jmy@xps ~/Projects $
```

- gradlew to instalacja gradle dołączona do naszego projektu
  - po skopiowaniu projektu nie musimy mieć instalacji gradle na maszynie docelowej

```
1  package hello;
2  import org.joda.time.LocalTime;
3
4  public class HelloWorld {
5    public static void main(String[] args) {
6      LocalTime currentTime = new LocalTime();
7      System.out.println("The current time is: " + currentTime);
8
9      Greeter greeter = new Greeter();
10     System.out.println(greeter.sayHello());
11   }
12 }
```

- gradlew to instalacja gradle dołączona do naszego projektu
  - po skopiowaniu projektu nie musimy mieć instalacji gradle na maszynie docelowej
- dodanie zależności:

```
...
dependencies {
    // Use JUnit Jupiter for testing.
    testImplementation 'org.junit.jupiter:junit-jupiter:5.7.2'

    // This dependency is used by the application.
    implementation 'com.google.guava:guava:30.1.1-jre'

    implementation "joda-time:joda-time:2.2"
}
...
```

- gradlew to instalacja gradle dołączona do naszego projektu
  - po skopiowaniu projektu nie musimy mieć instalacji gradle na maszynie docelowej
- dodanie zależności:
- uruchomienie aplikacji:

```
jmy@xps ~/Projects $ ./gradlew run

> Task :app:run
The current local time is: 20:05:13.183
Hello world!

BUILD SUCCESSFUL in 628ms
2 actionable tasks: 1 executed, 1 up-to-date
jmy@xps ~/Projects
```