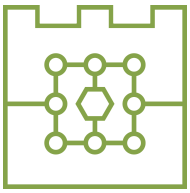


Programowanie w języku Java

Laboratorium nr 3

Politechnika Krakowska
Wydział Informatyki i Telekomunikacji
Katedra Informatyki

2024

Omówienie narzędzia Maven

Maven to narzędzie do zarządzania projektami i budowania aplikacji w Javie.

- Kompilację kodu.
- Zarządzanie zależnościami (bibliotekami używanymi w projekcie).
- Tworzenie plików JAR/WAR.
- Testowanie i uruchamianie aplikacji.

pom.xml - Podstawowy plik konfiguracyjny Maven, znajdujący się w katalogu głównym projektu.
Służy do:

- Definiowania projektu (np. groupId, artifactId, version).
- Deklaracji zależności.
- Konfiguracji wtyczek i zadań budowania.

Fazy budowania projektu:

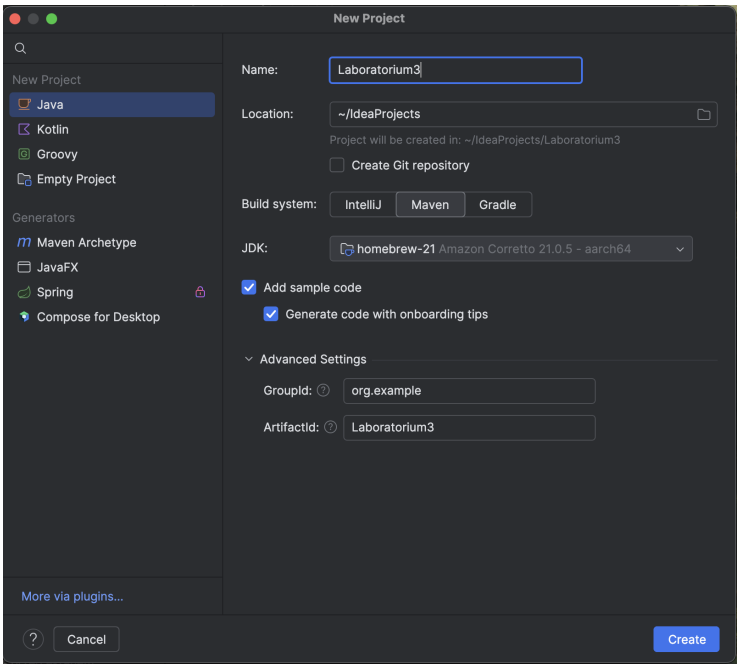
1. clean — usuwa wszystkie skompilowane pliki z katalogu docelowego (miejsca, w którym zapisywane są gotowe artefakty)
2. validate — sprawdza, czy dostępne są wszystkie informacje wymagane do zbudowania projektu
3. kompiluj — kompiluje pliki kodu źródłowego
4. package — pakuje skompilowane pliki (w archiwum JAR, WAR itp.)
5. verify - sprawdza, czy spakowany plik jest gotowy
6. install — umieszcza pakiet w lokalnym repozytorium. Teraz może być używany przez inne projekty jako biblioteka zewnętrzna
7. site — tworzy dokumentację projektową
8. deploy - kopiuje zbudowane archiwum do zdalnego repozytorium

Wszystkie fazy są wykonywane sekwencyjnie: na przykład czwarta faza nie może zostać rozpoczęta, dopóki fazy 1-3 nie zostaną zakończone.
Oficjalna dokumentacja Maven: https://maven.apache.org/guides/index.html

Instalacja

Tworzenie projektu Maven w **IntelliJ**:

1. Otwórz IntelliJ IDEA.
2. Kliknij: File -> New Project.



3. Wybierz opcję "Maven".
4. Wypełnij szczegóły projektu.

Tworzenie projektu Maven w **Visual Studio Code**:

1. Sprawdź, czy Maven jest zainstalowany:

```
mvn -v
```

2. Maven wymaga zainstalowanego JDK.

```
java -version
```

3. W VS Code przejdź do zakładki Extensions (Ctrl+Shift+X) i zainstaluj:
   - Extension Pack for Java (zawiera obsługę JDK, Maven i Debuggera).
   - Maven for Java.
4. Utwórz projekt Maven. W terminalu wbudowanym w VS Code:

```
mvn archetype:generate -DgroupId=com.example
    ↪ -DartifactId=ProjectName
    ↪ -DarchetypeArtifactId=maven-archetype-quickstart
    ↪ -DinteractiveMode=false
```

5. Otwórz projekt. Projekt powinien zawierać następną strukturę:

```
project-name/
├── pom.xml
├── src/
    ├── main/
    │   ├── java/
    │   │   └── ...
    │   ├── resources/
    └── test/
        ├── java/
        │   └── ...
        └── resources/
```

Zadania

Zadanie 1: Przygotowanie projektu Maven

1. Utwórz nowy projekt Maven.
2. Umieść wszystkie klasy i rekordy z Laboratorium 1 (`Shape`, `Rectangle`, `Triangle`, `ShapeDescriber`) oraz Laboratorium 2 (`Color`).
3. Skonfiguruj plik `pom.xml` tak, aby używał odpowiedniej wersji Javy.

Zadanie 2: Rozszerzenie klasy Shape o kolor

1. Dodaj pole `Color` do klasy `Shape`, aby każda figura mogła mieć przypisany kolor.
2. Zaktualizuj konstruktor klasy `Shape`, aby przyjmował obiekt typu `Color`.
3. Dodaj metodę `getColorDescription`, która zwraca tekstowy opis koloru figury (np. `"Red: 255, Green: 0, Blue: 0, Alpha: 255"`).
4. Zaktualizuj funkcję `main`, aby tworzyć obiekty figur z przypisanymi kolorami.

Zadanie 3: Logowanie informacji z wykorzystaniem SLF4J

1. **Dodaj zależność do biblioteki SLF4J w pliku `pom.xml`:**

```xml
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>2.0.7</version>
</dependency>
```

2. **Zaimplementuj logowanie w klasie `ShapeDescriber`:**

- Dodaj pole `Logger`, które będzie obsługiwać logowanie informacji.
- W metodzie `describe` loguj:
  - Opis figury (np. jej typ i kolor).
  - Pole figury (`getArea`).
  - Obwód figury (`getPerimeter`).

3. **Przetestuj działanie logowania w funkcji `main`:**

- Stwórz kilka figur (`Rectangle`, `Triangle`, itp.) z różnymi wymiarami i kolorami.
- Wywołaj metodę `describe` i upewnij się, że informacje zostały poprawnie zalogowane.

Zadanie 4: Polimorfizm i rozszerzenie funkcjonalności

1. Dodaj klasę `ShapeRenderer`, która rysuje figury w terminalu z uwzględnieniem ich koloru.
2. Klasa `ShapeRenderer` powinna działać z każdym obiektem typu `Shape` (np. `Rectangle`, `Triangle`, `Circle`), demonstrując zastosowanie polimorfizmu.
3. Utwórz listę obiektów `Shape` i przekaż ją do `ShapeRenderer`, aby wyświetlić wszystkie figury.

Zadanie 5: Dodaj testy jednostkowe

Dodaj zależność do biblioteki JUnit w pliku pom.xml
Upewnij się, że testy pokrywają co najmniej 80% kodu (można użyć narzędzia do analizy pokrycia, np. `JaCoCo`).

Important Information

Jako wynik zadania należy przesłać **plik PDF** zawierający cały kod oraz wynik uruchomienia metody main. (PDF może zawierać link do repozytorium GitHub, natomiast w pliku README w repozytorium należy umieścić zrzut ekranu z wynikiem działania programu.)