Hello, World!

ERNEST PERŻYŁO
KRYSTIAN ŻEŃCZAK
PAWEŁ ZEGARSKI

- Program "Hello World" jest używany, aby upewnić się, że kompilator i środowisko uruchomieniowe działają. Ilustruje również minimum kodu potrzebnego, aby uruchomić program generujący wyjście.

- Pozwala poznać podstawową składnię języka oraz dowiedzieć się jak projektować, zbudować i uruchomić aplikację.

Suffering-Oriented Programming

- Suffering-Oriented Programming jest stylem developmentu stworzonym przez Nathan'a Marz. Koncept ten brzmi: "First make it possible. Then make it beautiful. Then make it fast."
- Ten sposób myślenia zakłada, że na początku nasz kod powinien po prostu działać.
- Kiedy już mamy działający kod, który rozwiązuje nasz problem prawdopodobnie rozumiemy ten problem lepiej niż na samym początku, dlatego możemy uprościć rozwiązanie, żeby było bardziej przejrzyste. Jeżeli okaże się, że "piękniejsza" wersja kodu nie działa poprawnie to zawsze możemy wrócić do bazowej wersji kodu i spróbować ponownie.
- Ostatnim krokiem jest sprawienie, żeby kod był szybki. Trzeba skupić się na fragmentach kodu, które mogą być napisane nieoptymalnie, takie jak na przykład iteracje po strukturach danych.

Gradle

- Gradle jest obok Maven'a jednym z dwóch popularnych narzędzi pozwalających na budowanie aplikacji w Javie (ale także tych napisanych w Kotlinie, Scali, C czy JavaScript). Pozwala na prostą konfigurację i automatyzację procesu budowania z wykorzystaniem języka skryptowego zbliżonego do Javy.

- Gradle wykorzystuje strukturę plików zaczerpniętą z Maven'a, gdzie katalog src/main zawiera pliki aplikacji, a src/test zawiera pliki do testowania.

Przykładowy nadrzędny plik konfiguracyjny build.gradle.

Nasz plik jest podzielony na cztery sekcje:

```
apply plugin: 'java'
sourceCompatibility = 1.11
targetCompatibility = 1.11
ext {
    springFrameworkVersion = "5.1.5.RELEASE"
    jacksonVersion = "2.9.9"
    testNgVersion = "6.14.3"
}

allprojects {
    apply plugin: 'java'
    repositories {
        jcenter()
        mavenCentral()
    }
    dependencies {
        testImplementation "org.testng:testng:$testNgVersion"
    }
    test {
        useTestNG()
    } }
```

- Sekcja "plugin", gdzie wskazujemy, że projekt jest napisany w Javie.
- "sourceCompatibility" i "targetCompatibility", gdzie wskazujemy wersję Javy (w tym przypadku Java 11)
- Sekcja "ext", która ustawia zmienne dla każdego podmodułu, mamy tutaj na przykład zdefiniowaną wersję Springa.
- Sekcja "allprojects", gdzie znajduje się zestaw dyrektyw dotyczących budowania projektu, które odnoszą się do każdego podmodułu w ramach tego projektu.

W modułach możemy umieścić dodatkowe pliki konfiguracyjne, które będą zawierać zależności potrzebne w danym module.

```
dependencies {
    compile "org.springframework:spring-core:$springFrameworkVersion"
    compile "org.springframework:spring-context:$springFrameworkVersion"
    compile "org.springframework:spring-test:$springFrameworkVersion"
}
```

- Przykładowe zależności w pliku build.gradle w podmodule, w tym przypadku mamy trzy zależności Springa (spring-core, spring-context, spring-test). Wszystkie inne atrybuty takie jak wersja Javy będą pochodzić z nadrzędnego pliku build.gradle

Testy

- Wbrew powszechnym praktykom wstawienie wszędzie w kodzie "System.out.println" nie jest najlepszym sposobem na sprawdzenie czy program działa poprawnie.
- Testy pozwalają nam napisać założenia na temat kodu i ich walidację. Pomagają też w refaktoryzacji i mogą być podstawą do dokumentacji kodu.
- W tym przypadku do testów wykorzystana została biblioteka TestNG

Stworzyliśmy klasę HelloWorldGreeter implementującą interfejs Greeter, który ma dwie metody: setPrintStream i greet.

```java
package com.bsg5.chapter2;
import java.io.PrintStream;

public interface Greeter {
    void setPrintStream(PrintStream printStream);
    void greet();
}
```

```java
package com.bsg5.chapter2;

import java.io.PrintStream;

public class HelloWorldGreeter implements Greeter{
    private PrintStream printStream;

    @Override
    public void setPrintStream(PrintStream printStream) {
        this.printStream = printStream;
    }

    @Override
    public void greet() {
        printStream.print("Hello, World!");
        printStream.close();
    }
}
```

Przykładowy test dla metody greet() klasy HelloWorldGreeter.

```java
package com.bsg5.chapter2;
import org.testng.annotations.Test;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import java.io.UnsupportedEncodingException;
import java.nio.charset.StandardCharsets;
import static org.testng.Assert.assertEquals;

public class HelloWorldTest {
    @Test
    public void testHelloWorld() {
        Greeter greeter = new HelloWorldGreeter();
        final ByteArrayOutputStream baos = new ByteArrayOutputStream();
        try (PrintStream ps = new PrintStream(baos, autoFlush: true, encoding: "UTF-8")) {
            greeter.setPrintStream(ps);
            greeter.greet();
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        String data = baos.toString(StandardCharsets.UTF_8);
        assertEquals(data, expected: "Hello, World!");
    }
}
```

Dziękujemy za uwagę!!