WYŻSZA SZKOŁA BANKOWA
Warszawa

Programowanie aplikacji w Java

Maciej Gowin

Zjazd 8 - dzień 1

Linki

Opis

https://maciejgowin.github.io/wsb-java/

Kod źródłowy przykładów oraz zadań

https://github.com/MaciejGowin/wsb-programowanie-aplikacji-java

Wzorce projektowe

Wzorce projektowe (ang. design pattern) opisują rozwiązania często pojawiających się problemów w kontekście danego zagadnienia z dziedziny projektowania oprogramowania.

Nie definiują one dokładnego opisu odwzorowanego kodem źródłowym. Są one jedynie opisem lub też szablonem, który może zostać użyty w danym przypadku.

Wzorce projektowe są formalnym opisem dobrych praktyk programistycznych.

Wzorce projektowe: singleton

Wzorzec singleton ogranicza możliwość stworzenia instancji danej klasy tylko do pojedynczej instancji.

> Singleton często nazywany jest antypaternem (ang. anti-pattern) w związku z przechowywaniem globalnego stanu.

Wzorce projektowe: obserwator

Wzorzec obserwator (ang. observer) używany jest do utrzymywania listy obserwatorów powiązanych z danym tematem oraz wysyłaniu automatycznych notyfikacji do wszystkich obserwatorów w przypadku nowych zdarzeń dla danego tematu.

Wzorce projektowe: dekorator

Wzorzec dekorator (ang. decorator) pozwala na dynamiczne dodawanie funkcjonalności obiektom bez zmiany zachowania innych obiektów danej klasy.

> Dekorator jest często wykorzystywany w celu pominięcia skomplikowanej struktury dziedziczenia.

Programowanie: przykład 69

Przykłady wzorców projektowych w Java.

Project Lombok

W wielu przypadkach zmuszeni jesteśmy do pisania powtarzalnego kodu, którego przykładem mogą być: settery, gettery czy metody equals oraz hashCode.

Project Lombok jest biblioteką, która wspomaga programistów w dynamicznym generowaniu kodu dla najbardziej powtarzalnych operacji.

Biblioteka opera się na szeregu adnotacji, które są przetwarzane, a następnie na ich podstawie tworzony jest kod źródłowy.

> Project Lombok używa specjalnego mechanizmu pozwalającego na przetworzenie adnotacji podczas kompilacji. Aby zmiany te były widoczne w IDE, konieczne jest dodanie pluginów wspierających Project Lombok.

Project Lombok

Aby użyć biblioteki Project Lombok w praktyce, wystarczy jedynie dostarczyć ją jako zależność Maven.

```
<dependencies>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.22</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Szczegóły biblioteki dostępne pod adresem: https://projectlombok.org/

Programowanie: przykład 70

Użycie Project Lombok w praktyce.

Logowanie

Do tej pory do przeglądu zachowań i testowania aplikacji używaliśmy wypisywania informacji przy pomocy standardowego wyjścia (`System.out`).

W oprogramowaniu logowanie to process zapisywania informacji o zdarzeniach, które wystąpiły w systemie. Informacje te mogą później posłużyć do analizy działania oprogramowania i wspierają w zrozumieniu problemów, które wystąpiły podczas działania.

Dane czy też pliki, które powstają podczas logowania (ang. logging), nazywane są logami (ang. logs).

Java w praktyce: Java logging

JSE dostarcza wbudowaną obsługę logowana poprzez pakiet `java.util.logging`, który został dodany wraz z Java 1.4.

Główną klasą służącą do zapisywania logów jest `java.util.logging.Logger`, której to instancje tworzone są przy pomocy statycznej metody.

```
Logger logger = Logger.getLogger("logger");
logger.log(Level.INFO, "Application: start");
logger.info("Application: end");
```

Java w praktyce: Java logging

Logowanie opiera się na poziomach logowania, do których zaliczamy

SEVERE (poziom najwyższy), WARNING, INFO (poziom domyślny), CONFIG, FINE, FINER, FINEST (poziom najniższy)

Podczas konfiguracji mechanizmu logowania możemy zdecydować, od którego poziomu logowania informacje będą wypisywane.

Załóżmy, że nasza aplikacja używa każdego z poziomów logowania oraz system został skonfigurowany do używania poziomu INFO. W efekcie wypisane zostaną tylko informacje o logach z poziomu INFO oraz wyższych (INFO, WARNING oraz SEVERE).

Java w praktyce: konfiguracja Java logging

Konfiguracja logowania odbywa się poprzez specjalnie przygotowany plik `logging.properties`.

Plik konfiguracyjny może być załadowany przy użyciu kilku mechanizmów:

Automatyczny

Od Java 9 konfiguracja jest automatycznie ładowana z katalogu `$JAVA_HOME/conf`.

Podczas uruchomienia programu

Nadpisanie pliku konfiguracyjnego podczas uruchomienia programu przy pomocy zmiennej `java.util.logging.config.file`.

```
java -jar -Djava.util.logging.config.file=/path/to/logging.properties application.jar
```

Java w praktyce: konfiguracja Java logging

Z poziomu programu przez zmienną

Zdefiniowanie lokalizacji pliku oraz nadpisanie zmiennej systemowej `java.util.logging.config.file`.

```
System.setProperty("java.util.logging.config.file", "/path/to/logging.properties");
```

Z poziomu programu przez LogManager

Załadowanie pliku explicite oraz dodanie to klasy konfiguracyjnej.

```
InputStream is = new FileInputStream("/path/to/logging.properties");
LogManager.getLogManager().readConfiguration(is);
```

Java w praktyce: konfiguracja Java logging

Konfiguracja systemu logowania pozwala nam na zdefiniowanie:

- klas obsługujących logowanie (np. wypisywanie informacji na konsolę, zapisywanie danych do pliku)
- poziomów logowania
- formatu wiadomości

Programowanie: przykład 71

Użycie Java Logging.

Java w praktyce: Java logging

Logowanie przy pomocy wbudowanej biblioteki `java.util.logging` jest dość ograniczone. Większość projektów korporacyjnych czy też bibliotek OpenSource używa innych rozwiązań takich jak: `log4j` czy `logback`.

W wielu przypadkach integracja ta odbywa się przy pomocy interfejsu `SLF4J`.

Java w praktyce: SLF4J

`SLF4J` (ang. Simple Logging Facade for Java) jest fasadą dla wielu bibliotek logowania (np. `java.util.logging`, `logback`, `Log4J`). Dostarcza szereg interfejsów, które tworzą abstrakcję dla konkretnej implementacji użytej w projekcie.

Konkretne implementacje mogą być dołączone podczas uruchomienia.

Biblioteka `logback` jest natywną implementacją `SLF4J` dzięki czemu wystarczy bezpośredniej użycie biblioteki.

W przypadku `Log4J` koniecznej jest dodanie biblioteki umożliwiającej dostosowanie `Log4J` do `SLF4J`.

Koncepcyjnie użycie logowanie przy pomocy `SLF4J` jest analogiczne do tego znanego z `java.util.logging`.

Programowanie: przykład 72

Użycie Logback.

Java w praktyce: Jackson

Jackson jest biblioteką służącą do automatycznej konwersji (serializacji oraz deserializacji) obiektów do formatu JSON.

Centralnym elementem biblioteki jest klasa `ObjectMapper` realizująca konwersję. Biblioteka ta w dużej mierze opiera się na konfiguracji konwersji przy użyciu adnotacji.

Jackson wspiera konwersję typów wbudowanych w język Java oraz pozwala na definiowanie własnych, alternatywnym metod serializacji i deserializacji typów.

Programowanie: przykład 73

Użycie Jackson w praktyce.

Spring Framework

Spring Framework jest szkieletem do tworzenia aplikacji przy pomocy języka Java.

Powstał on jako alternatywa dla rozwiązania EJB będącego częścią J2EE (Java Enterprise Edition), które narzucało zbyt sztywne ramy ograniczające swobodne tworzenie oprogramowania.

Co więcej, dla prostych projektów narzut związany z używaniem EJB był zbyt duży.

Spring Framework

Spring Framework składa się z wielu modułów rozwiązujących konkretne problemy, przy czym nie narzuca on specyficznego modelu programowania. Mogą one być dołączane w razie potrzeby.

Podstawowe moduły mogą być użyte przez dowolną aplikację w Java, aczkolwiek dostępne rozszerzenia pozwalają na budowanie aplikacje webowe na podstawie J2EE (Java Enterprise Edition).

W programowaniu korporacyjnym Spring Framework jest wykorzystywany głównie do tworzenia aplikacji Web.

Spring: kontener

Centralnym elementem działania frameworka jest kontener, którego głównym zadaniem jest tworzenie i zarządzanie obiektami oraz ich pełnym cyklem życia.

Obiekty te są inicjalizowane przy użyciu konkretnych mechanizmów frameworka oraz zarządzane przez wewnętrzny kontener Springa, dzięki czemu mogą one być używane wielokrotnie.

Spring: kontener

Obiekty, będące pod zarządzaniem kontenera Spring, określane są mianem tzw. **ziaren (ang. bean)**.

Kontener Springa jest głównym oraz najistotniejszym elementem frameworka.

Cały proces ładowania oraz zarządzania komponentów jest związany z ideami: **Inversion of Control** i **Dependency Injection**.

Spring: Inversion of Control

Inversion of Control (IoC, pol. odwrócenie kontroli) jest podejściem, w którym przekazujemy zarządzanie obiektami do swego rodzaju kontenera czy też frameworka.

W porównaniu z tradycyjnym podejściem, w którym to nasz kod odwołuje się do klas i bibliotek, IoC przejmuje kontrolę nad przepływem kodu i zarządzaniem wywołaniami. Z naszej perspektywy dostarczamy rozszerzeń zachowań lub dodajemy nowe, które później są kontrolowane przez mechanizm IoC.

W frameworku Spring podejście IoC jest realizowane przez mechanizm Dependency Injection (DI, pol. wstrzykiwanie zależności).

Spring: Inversion of Control

Głównymi zaletami podejścia IoC są:

- modułowość rozwiązania,
- oddelegowanie zarządzania,
- oddzielenie implementacji od interfesju,
- łatwe przełączanie się pomiędzy implementacjami,
- ułatwione testowanie niezależnych implementacji.

Spring: Dependency Injection

Dependency Injection (DI, pol. wstrzykiwanie zależności) to mechanizm realizujący podejście IoC, w którym zarządzanie zależnościami pomiędzy obiektami zostało odwrócone.

Łączenie obiektów i ich zależności odbywa się poprzez ich "wstrzykiwanie" na poziomie elementu zarządzającego, a nie explicite z poziomu kodu źródłowego.

Spring: Dependency Injection

W klasycznym podejściu nasz kod mógłby wyglądać następująco:

```
class Worker {
    private Job job;

    public Worker() {
        this.job = new HardJob();
    }
}
```

W podejściu DI zapis uległby zmianie:

```
class Worker {
    private Job job;

    public Worker(Job job) {
        this.job = job;
    }
}
```

Spring: Dependency Injection

W podejściu DI nie definiujemy konkretnej zależności. Jest ona automatycznie wstrzykiwana, a jej implementacja jest definiowana na poziomie konfiguracji kontekstu mechanizmu DI.

Mechanizm ten jest później odpowiedzialny za poprawne powiązanie zależności oraz dostarczenie ich podczas użycia.

Spring: IoC Container

Spring implementuje IoC oraz DI za pomocą kontenera IoC (ang. IoC container), który jest reprezentowany poprzez klasę `ApplicationContext`.

Kontekst tworzony jest na podstawie konfiguracji, która opisuje dostępne komponenty zwane ziarnami (ang. bean).

Na późniejszy etapie to kontener jest odpowiedzialny za tworzenie, konfigurację oraz składanie obiektów (ziaren), jak i zarządzanie ich cyklem życia.

Spring Context

Mechanizm IoC oraz DI jest obsługiwany przez moduł Spring pod nazwą `spring-context`.

Moduł ten jest centralnym modułem frameworka pozwalającym zainicjalizowanie kontenera IoC.

Istnieje wiele możliwości zainicjalizowania kontekstu poprzez konfigurację opartą o:

- XML,
- kod Java,
- adnotacje.

Podejścia te mogą być łączone.

Spring Context: konfiguracja oparta o XML

Tworzenie kontekstu może odbyć się na podstawie definicji zawartej z pliku konfiguracyjnym w formacie XML. Plik ten może zostać załadowany ze ścieżki `classpath` przy użyciu klasy `ClassPathXmlApplicationContext`.

```
ApplicationContext context = new ClassPathXmlApplicationContext("classpath:context.xml");
```

Plik konfiguracyjny `context.xml` zawiera definicję ziaren, które zostaną załadowane i skonfigurowane na poziomie kontenera.

```
<bean id="routeReader" class="pl.wsb.programowaniejava.maciejgowin.service.RouteReader"/>
<bean id="airportReader" class="pl.wsb.programowaniejava.maciejgowin.service.AirportReader"/>

<bean id="routeService" class="pl.wsb.programowaniejava.maciejgowin.service.RouteService">
    <constructor-arg name="routeReader" ref="routeReader" />
    <constructor-arg name="airportReader" ref="airportReader" />
</bean>
```

Programowanie: przykład 74

Inicjalizacja Spring Context na podstawie konfiguracji plikiem XML.

Spring Context: konfiguracja oparta o XML oraz adnotacje

Spring 2.5 wprowadził opcję konfiguracji pozwalającą na dokonywanie wstrzykiwań na podstawie adnotacji. Główną adnotacją pozwalającą na dokonanie wstrzyknięć zależności jest `@Autowired`. Może ona zostać użyta na poziomie pól klasy.

```
(...)
public class RouteService {

    @Autowired
    private RouteReader routeReader;
(...)
```

Kontener automatycznie wyszuka odpowiednich ziaren oraz dokona wstrzyknięcia zależności. Funkcjonalność ta nie jest domyślnie włączona. Aby ją aktywować, należy wzbogacić konfigurację XML o znacznik:

```
<context:annotation-config>
```

Programowanie: przykład 75

Inicjalizacja Spring Context na podstawie konfiguracji plikiem XML oraz konfiguracji adnotacjami.

Spring Context: konfiguracja oparta o XML oraz adnotacje

Do tej pory definiowaliśmy ziarna na poziomie pliku konfiguracyjnego XML. Możliwe jest też oznaczenie klasy jako klasy definiującej ziarno. Dzięki temu zostanie ono automatycznie utworzone bez konieczności jego deklaracji w pliku konfiguracyjnym.

Do podstawowych adnotacji oznaczających klasę jako klasę dostarczającą ziarna są: `@Component`, `@Repository`, `@Service`.

Aby aktywować tę funkcjonalność, musimy poinformować kontekst, aby dokonał wyszukiwania adnotacji odpowiedzialnych za tworzenie ziaren w danej lokalizacji. Naszą konfigurację XML należy wzbogacić o znacznik:

```
<context:component-scan base-package="pl.wsb.maciejgowin" />
```

Programowanie: przykład 76

Inicjalizacja Spring Context na podstawie konfiguracji plikiem XML oraz skanowania komponentów.

Spring Context: konfiguracja oparta o kod Java

Konfigurację poprzez pliki XML możemy w pełni zastąpić poprzez klasy konfiguracyjne oznaczone adnotacją `@Configuration` oraz dostarczające ziarna poprzez metody oznaczone adnotacją `@Bean`. Opcja ta została wprowadzona w Spring 3.0.

Aby zainicjalizować kontekst przy pomocy klasy konfiguracyjnej użyjemy `AnnotationConfigApplicationContext` oraz parametrem będącym klasą konfiguracyjną:

```
ApplicationContext context = new AnnotationConfigApplicationContext(MainConfiguration.class);
```

Adnotacja `@Configuration` informuje kontener, że klasa zawiera definicję ziaren oraz dodatkowe opcje konfiguracyjne.

Programowanie: przykład 77

Inicjalizacja Spring Context na podstawie konfiguracji kodem Java.

Spring Context: konfiguracja oparta o kod Java oraz adnotacje

Podobnie jak w przypadku konfiguracji XML, podczas konfiguracji kontekstu z poziomu kodu Java możemy użyć adnotacji `@Autowired` w celu wstrzyknięcia zależności.

W przypadku kontekstu `AnnotationConfigApplicationContext` opcja ta jest automatycznie włączona.

Programowanie: przykład 78

Inicjalizacja Spring Context na podstawie konfiguracji kodem Java oraz konfiguracji adnotacjami.

Spring Context: konfiguracja oparta o kod Java oraz adnotacje

Aby włączyć opcję wyszukiwania klas dostarczających ziaren z poziomu kodu Java, użyjemy adnotacji `@ComponentScan`, zdefiniowanej na poziomie klasy konfiguracyjnej użytej podczas tworzenia kontekstu.

```
@Configuration
@ComponentScan("pl.wsb.programowaniejava.maciejgowin")
public class MainConfiguration {
}
```

Programowanie: przykład 79

Inicjalizacja Spring Context na podstawie konfiguracji kodem Java oraz skanowania komponentów.

Spring: wstrzykiwanie zależności

Wstrzykiwanie zależności może odbywać się za pomocą:

- konstruktora (ang. constructor based): kontener wywołuje konstruktor wraz z wartościami zależności, które powinny zostać ustawione,
- metody setter (ang. setter based): kontener wywołuje setter dla danej zależności, której wartość powinna zostać ustawiona,
- pól klasy (ang. field based): kontener wstrzykuje zależności refleksji do pól oznaczony adnotacją `@Autowired` przy pomocy mechanizmu refleksji.

Spring: @Autowired

Automatyczne wiązanie (ang. autowiring) zależność poprzez adnotację @Autowired wymaga poprawnego dobrania instancji ziaren, które odpowiadają danej zależności.

Istnieją 4 modele automatycznego dostosowywania danych ziaren do zależności:

- **brak:** automatyczne wiązanie jest wyłączone oraz zależności muszą być ustawione explicite,
- **po nazwie (ang. by name):** automatyczne wiązanie odbywa się po nazwie pola, kontener wyszukuje ziaren o nazwie pola, które ma zostać ustawione,
- **po type (ang. by type):** automatyczne wiązanie odbywa się po typie pola, kontener wyszukuje ziaren o typie pola, które ma zostać ustawione, przy dopasowaniu więcej niż jednego ziarna zwrócony zostanie błąd,
- **po konstruktorze (ang. constructor):** automatyczne wiązanie oparte o argumenty konstruktora, kontener wyszukuje ziaren o typach odpowiednich dla argumentów konstruktora.

Spring: jakie obiekty powinny być ziarnami?

Oczywiście nie wszystkie obiekty powinny być definiowane jako ziarna.

Obiekty domenowe (takie jak encje Hibernate) powinny być tworzone i zarządzane przez logikę biznesową.

Z definicji ziarnami będą wszystkie serwisy oraz usługi definiujące logikę biznesową, obsługę dostępu do bazy danych, obsługę warstwy prezentacji, obsługę I/O itp.