SPRING CORE

Materiały szkoleniowe

Autor	Marcin Górski
Email	marcin.gorski@basepoint.pl
Telefon	+48 507-543-982

S-01 Wymagania wstępne

Środowisko

Skonfigurowane środowisko pracy (Java, Maven). Poniższe komendy powinny wykonać się poprawnie.

```
java –version
javac -version
mvn – version
echo %JAVA_HOME%
echo %M2_HOME%
```

Przykład poprawnie wykonanych komend:

```
C:\>java -version
java version "1.8.0_05"
Java(TM) SE Runtime Environment (build 1.8.0_05-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.5-b02, mixed mode)

C:\>javac -version
javac 1.8.0_05

C:\>mvn -version
Apache Maven 3.2.2 (45f7c06d68e745d05611f7fd14efb6594181933e; 2014-
06-17T15:51:42+02:00)
Maven home: c:\tools\apache-maven-3.2.2
Java version: 1.8.0_05, vendor: Oracle Corporation
Java home: C:\tools\jdk1.8.0_05\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 8.1", version: "6.3", arch: "amd64", family: "dos"

C:\>echo %JAVA_HOME%
C:\tools\jdk1.8.0_05

C:\>echo %M2_HOME%
c:\tools\apache-maven-3.2.2\
```

Globalna konfiguracja Maven

Na potrzeby szkolenia niektóre ustawienia Maven trzymane są poza projektem, np.wersje używanych bibliotek. Umożliwia to łatwy upgrade wszystkich przykładów prezentowanych na szkoleniu.

Zadanie

Pobierz plik **settings.xml** od trenera i umieść go w odpowiednim katalogu. Zarówno dla systemów Windows jak i Unix plik ten powinien znajdować się w katalogu **.m2** znajdującym się w folderze domowym użytkownika. Folder ten jest domyślnie ukryty.

Uwaga: Jeśli posiadasz już własny, niestandardowy plik settings.xml, skopiuj nową zawartość nie tracąc swoich ustawień!

S01 Prosta aplikacja Java

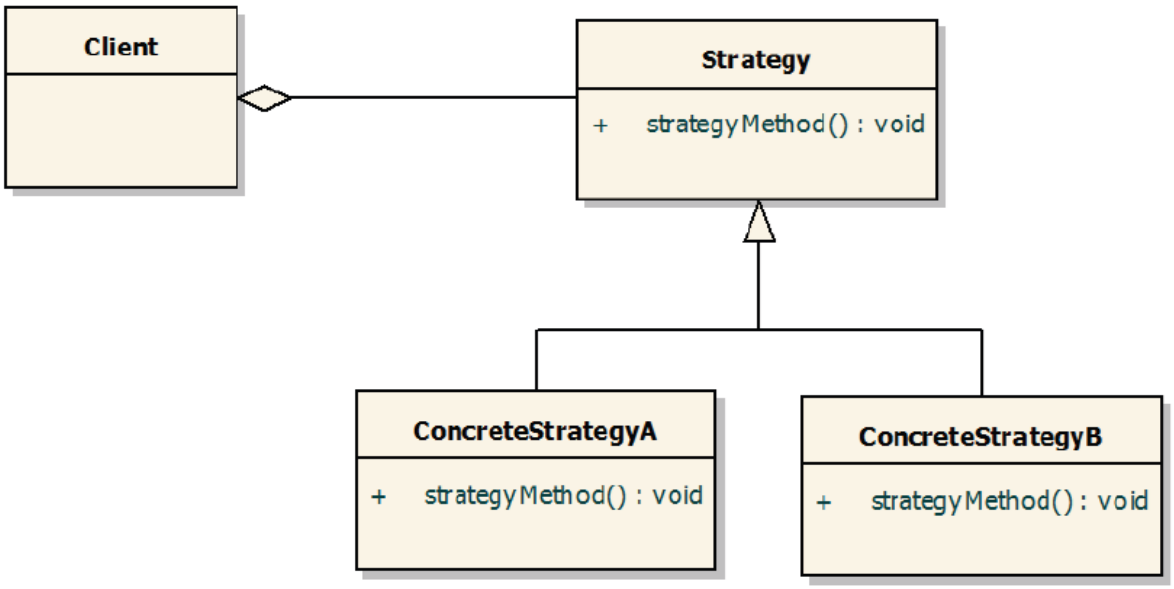
Uruchomienie aplikacji Java budowanej o Maven.

Zadanie

Pobierz przykładową aplikację. Skompiluj ją z linii poleceń wykonując **mvn clean install.** Zaimportuj aplikację do IDE (Eclipse, IntelliJ, Netbeans) oraz uruchom obserwując wynik.

S02 Wzorzec Strategia (Strategy Pattern)

Istnieje potrzeba stosowania różnych wersji jednego algorytmu, a jego wyborem sterują używające go obiekty lub dane, na których operuje. Chcemy w elastyczny sposób sterować wyborem wykorzystywanego algorytmu, chcemy w łatwy sposób dodawaćc nowe algorytmy.



Po zastosowaniu wzorca Strategy:

- każda wersja algorytmu lub reguły biznesowej jest implementowana w osobnej klasie
- klient przechowuje referencję do nadklasy lub interfejsu Strategy
- klient realizuje swoją˛ funkcjonalność´ wykorzystując metody udostępniane przez typ Strategy
- w czasie działania klient posługuje się konkretnym podtypem typu Strategy

Zadanie

Implementujesz różne algorytmy, które operują na dowolnej ilości argumentów typu integer (argumentem jest tablica *int[]*) a zwracają wynik w postaci liczby typu *long*. Powinieneś przewidzieć możliwość dodawania różnych wariantów w przysłości, oraz umożliwic łatwe podmienianie rodzaju algorytmu.

Utwórz interfejs *Startegy* z metodą execute.

```
public interface Startegy {
     public long execute(int ...  a);
}
```

- Utwórz poniższe klasy implementujące interfejs *Strategy*.
  - ConcreteStartegyAdd
  - ConcreteStrategyMultiply
  - ConcreteStrategySubstract
- Klasy powinny wykonywać operacjie odpowiednio dodawania, mnożenia oraz odejmowania na zadanych argumentach.
- Utwórz klasę Context o poniższej zawartości

```
public class Context {
      public Context(Strategy strategy){
          this.strategy = strategy;
}

     public int executeStrategy (int a, int b){
          this.startegy.execute(a,b);
}
}
```

- Utwórz klasę testującą strategię dodawania.

```
Context cxt; = new Context(new ConcreteStartegyAdd());
long result = ctx.executeStrategy( 7, 12, 0, 22, -1, 18 );
System.out.println("Result: "+result);
```

- Dodaj kod testujący analogicznie odejmowanie oraz mnożenie.
- Czy możliwe jest zaimplementowanie operacji modulo oraz dzielenia przy użyciu tego samego infterfejsu?

S03 Spring context

Użycie Spring'a w prostej aplikacji

Zadanie

Zmodyfikuj aplikację z poprzedniego przykładu tak aby używała Spring'a.

1. Context zastąp kontekstem Spring
2. Stwórz konfigurację w pliku **spring.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
      xmlns:c="http://www.springframework.org/schema/c"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.2.xsd">

   <!-- tutaj zdefiniuj beany projektu -->
    <!--<bean id="" class="" />-->

</beans>
```

3. Zainicjalizuj context i uruchom aplikację

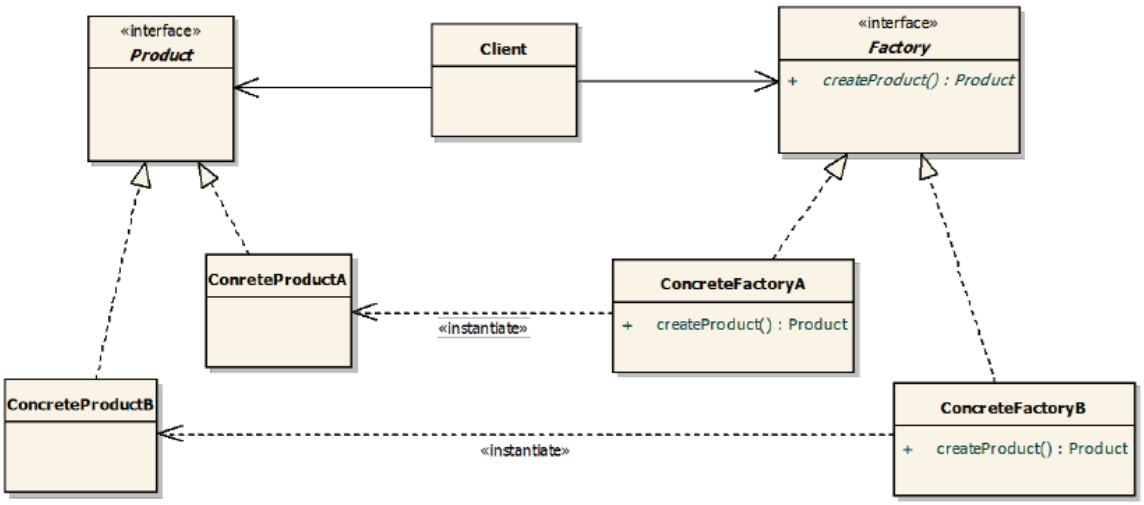
S03 Dynamiczne tworzenie instancji obiektu - wzorzec Factory Method

Problem:

- Chcemy w elastyczny sposób decydować o wyborze podklasy, której obiekty będą tworzone i wykorzystywane w aplikacji
- Wybór konkretnej klasy powinien być uzależniany od parametrów konfiguracyjnych lub otoczenia systemowego aplikacji

Po zastosowaniu wzorca Factory Method:

- klienci przechowują referencję typu Factory do obiektów klasy ConcreteFactory, odpowiedzialnej za tworzenie pożądanych obiektów
- jeżeli istnieje potrzeba wymiany klasy obiektów wykorzystywanych przez klienta, należy przekazać do niego obiekt innej klasy ConcreteFactory



Legenda

- Client - klasa wymagająca elastycznego sposobu wybierania współpracujących klas
- Factory - interfejs specyfikujący metody odpowiedzialne za tworzenie obiektów klas implementujących interfejs Product
- Product - specyfikacja metod, w oparciu o które działają obiekty klasy Client
- ConcreteFactoryA, ConcreteFactoryB - implementacje interfejsu Factory odpowiedzialne za tworzenie obiektów klas implementujących interfejs Product dla potrzeb obiektów klasy Client
- ConcreteProductA, ConcreteProductB 4 - klasy implementuja¸ce interfejs Product, dostarczaje˛ce funkcjonalności wymaganej przez obiekty klasy Client

Przykład

1. **Implementujesz prototyp apliklacji, która pozwoli przedstawić poziom naładowania odnalezionych w systemie baterii.** Każda odnaleziona w systemie bateria powinna być reprezentowana poprzez interfejs umożliwiający pobranie podstawowych informacji, tj. nazwa baterii, poziomu jej naładowania w procentach oraz informacji czy w danym momencie jest ładowana czy nie.
   1.1. Stwórz interfejs Battery reprezentujący baterię.

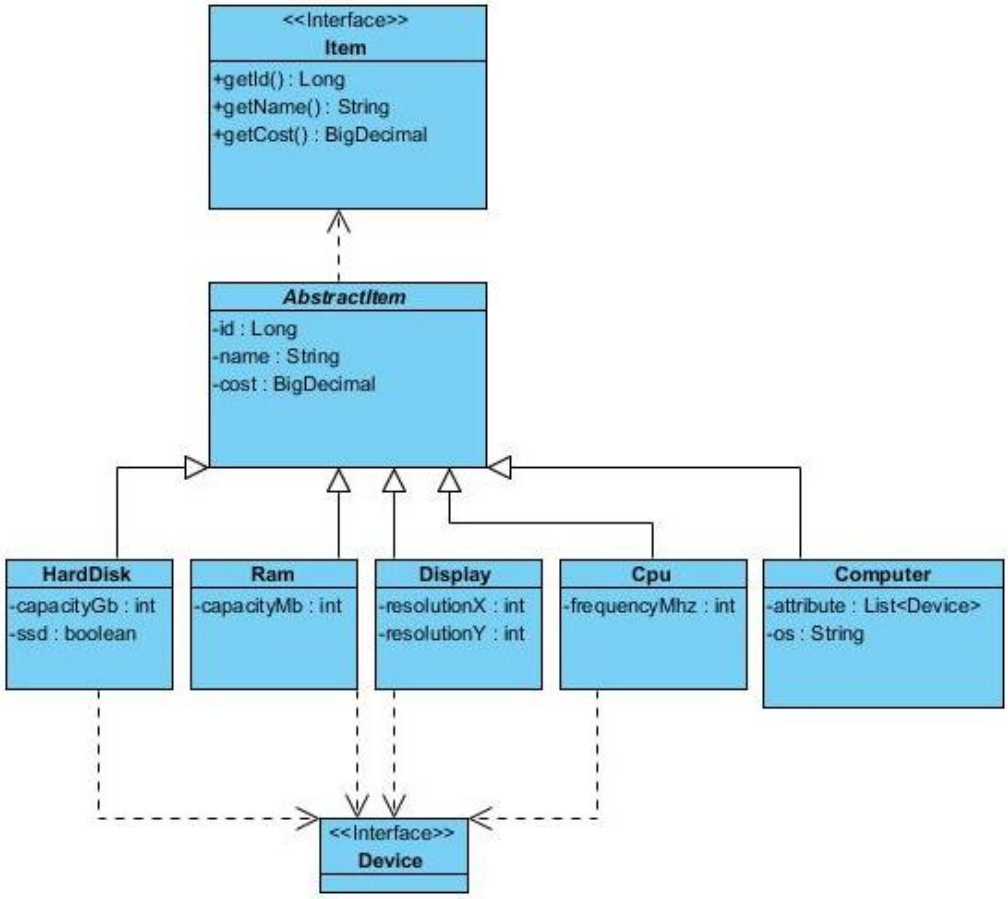
```
public interface Battery {
      public String getName();
      public boolean isCharging();
      public double chargeLevel();
}
```

2. Stwórz dwie przykładowe implementacje
   - LaptopBattery
   - MouseBattery
   Użyj "zahardcodowanych" danych - staraj się zasymulować działanie prawdziwego systemu.
3. Stwórz interfejs fabryki *BatteryFactory*. Stwórz publiczną metodę *createBattery* zwracającą obiekt rypu *Battery*.

4. Zaimplementuj dwie przykładowe fabryki:
   - LaptopBatteryFactory
   - MouseBatteryFactory
   Metody *createBattery* powinna zwracać odpowiednie obiekty.
5. Zaimplementuj z metodach fabryk ustawianie wartości *isCharging* oraz *chargeLevel* dla wspomnianych implementacji interfejsu *Battery*.
6. Utwórz klasę, która pozwoli wylistować systemowe baterie oraz wyświetlić ich stan.

S04 Spring – konfiguracja oparta o adnotacje

Będziesz tworzył wirtualny sklep. Stwórz strukturę klas jak na schemacie poniżej. Możesz wprowadzić własne modfikacje/ulepszenia.



1. Skonfiguruj obiekty za pomocą XML w Spring.

Dodaj bibliotekę *commons-lang* i przy pomocy jednej z jej klas - *ReflectionToStringBuilder* zaimplementuj (nadpisz) metody *toString* w powyższych klasach.

2. Utwórz kontekst Spring'a oraz przetestuj pobieranie bean'ów rozwiązanie.