Przewodnik 8 – implementacja polimorfizmu dynamicznego w języku JAVA na przykładzie hierarchii klas Person, Employee, Secretary i Programmer

dr inż. Łukasz Sosnowski
Akademia WIT
pod auspicjami Polskiej Akademii Nauk

1 Dodanie metody w klasie bazowej

W klasie bazowej Person przekazanej hierarchii klas zmień definicję klasy na abstrakcyjną oraz dodaj publiczną metodę abstrakcyjną:

boolean search(arg)

mogącą generować wyjątki klasy *Exception* i przyjmującą 1 parametr:
Map<String,Pair<Object,Byte>> conditions
gdzie:
kluczem mapy jest unikalna nazwa zmiennej składowej
wartością jest obiektem Pair<Object,Byte> biblioteki `org.javatuples`

W celu wykorzystania tej biblioteki proszę dodać zależność w pliku pom.xml

```
<dependency>
    <groupId>org.javatuples</groupId>
    <artifactId>javatuples</artifactId>
    <version>1.2</version>
</dependency>
```

Znaczenie wartości obiektu Pair ma być następujące: Wartość zadana dla danej zmiennej, identyfikator operatora. Zakładamy, że wartość nie może być null.

2 Implementacja metody w klasie Employee

Wykonaj implementację metody **search** w klasie pochodnej (Employee) w taki sposób aby:

1. implementacja wykorzystywała metodę *matches* w tej klasie oraz klasach bazowych
2. uwzględnij odpowiednie operatory dla odpowiednich typów zmiennych składowych
3. wykonaj obsługę błędów dla przypadku użycia niezgodnego operatora

Metoda ma zwracać true jeśli koniunkcja warunkowa wyrażeń logicznych zbudowanych dla poszczególnych elementów mapy conditions będzie prawdziwa. Do implementacji wyboru

zmiennej składowej dla której ma być wykonana operacja użyj instrukcji switch (nie refleksji gdyż nie jest jeszcze znana ;)).

UWAGA: jeśli mapa conditions będzie pusta to metoda ma zwrócić true.

Metoda ta ma za zadanie określać spełnienie warunku dopasowania względem przekazanych warunków.
Wykonaj implementację z użyciem metody z klasy bazowej oraz dodatkowo uwzględnij w implementacji obsługę zmiennych składowych klasy Employee.

Kluczem mapy conditions jest nazwa zmiennej z postaci łańcucha znaków. Wartością jest obiekt Pair, który jako klucz posiada wartość kryterium a jako wartość operator zakodowany na typie Byte. Należy przeiterować się przez poszczególne elementy mapy i obsłużyć poszczególne warunki. Warunki zbudowane w ten sposób należy połączyć z użyciem koniunkcji warunkowej.

UWAGA: w zależności od przekazanego klucza dotyczącego konkretnego pola różne operatory mogą być dozwolone. Należy sprawdzać czy wartość podana jako Object koresponduje z podanym operatorem. Sprawdzenie to powinno być zrobione z użyciem „instanceof".

Podpowiedź: porponuję utworzyć metody prywatne sprawdzające czy dla danego typu pola dozwolony jest przekazany operator, np.

```
private boolean isStringOperator(Byte operator) {
		if(operator.equals(OperatorsConst.EQUALS)
				|| operator.equals(OperatorsConst.ILIKE)) {
			return true;
		}
		else {
			return false;
		}
	}
```

3 Przesłonięcie metody w klasie Secretary

Zdefiniuj metodę przesłoniętą **search.**

Wykonaj implementację z użyciem metody z klasy bazowej oraz dodatkowo uwzględnij w implementacji obsługę zmiennych składowych klasy Secretary.

4 Przesłonięcie metody w klasie Programmer

Zdefiniuj metodę przesłoniętą **search.**

Wykonaj implementację z użyciem metody z klasy bazowej oraz dodatkowo uwzględnij w implementacji obsługę zmiennych składowych klasy Programmer.

5 Dodanie metody w klasie DbPersons

Dodaj metodę

```
public List<AbstractPerson4> search(Map<String,Pair<Object,Byte>> conditions) throws
Exception
```

Obsługującą wyszukiwanie z użyciem wcześniej zaimplementowanych metod search.

6 Dodanie testu jednostkowego

Wykonaj test jednostkowy z klasy DBPersonsTest i sprawdź poprawność działania