Pyłek

Radosław Ćwikowski i Grzegorz Łach

20 września 2020

Plan prezentacji

- Nazwa i Przeznaczenie
- Motywacja
- Struktura
- Uczestnicy
- Tworzenie klasy
- Wywołanie pętli
- Wynik aplikacji

Nazwa i Przeznaczenie

Nazwa wzorca oraz klasyfikacja:
Pyłek (ang. flyweight)

Przeznaczenie:
Jest to wzorzec strukturalny, który przydaje się w aplikacjach, które korzystają z większej liczby identycznych obiektów.

Motywacja

Motywacja:
Wzorzec Pyłek stosuje się tam, gdzie jedna klasa ma wiele egzemplarzy, a każdy z tych egzemplarzy może być sterowany w ten sam sposób. Przykładowo tenże wzorzec można zastosować w programie wspomagającym modelowanie przestrzenne terenu. Jednym z wielu elementów, które muszą się w nim znaleźć są obiekt reprezentujący drzewo. Zakładamy, że obiekt taki posiada informacje o wyglądzie drzewa oraz jakieś inne jego cechy, przy czym także jego wysokości oraz jego współrzędne położenia. Podczas modelowania wielkich kompleksów zieleni złożonych z wielu egzemplarzy drzewa program może zacząć działać niezadowalająco wolno. Aby uporać się z takim problemem można zastosować wzorzec Pyłek. Po zastosowaniu tego wzorca projektowego zamiast tworzyć indywidualny egzemplarz klasy (obiekt) dla każdego drzewa możliwe jest stworzenie kompleksowego obiektu, który będzie w sobie zawierał informacje o wszystkich drzewach renderowanych na modelowanym terenie.

Struktura

Struktura:
W momencie tworzenia nowego obiektu, sprawdzane jest czy egzemplarz o takich samych parametrach istnieje już w pamięci. Jeśli tak, to pobieramy go, a jeśli nie, tworzymy nowy. Obiekty te są immutable (niezmienne), czyli przygotowujemy klasę tak aby nie dało się jej zmodyfikować. Natomiast gdy zajdzie potrzeba zmiany danych wtedy tworzona jest nową instancja. Dzięki wykorzystaniu pyłku zmniejszamy ilość wykorzystywanej pamięci.

Uczestnicy

Uczestnicy:

Implementacja flyweight. Tworzenie klasy Relation.

```
final public class Relation {
    final private String description;

    public Relation(String description) {
        this.description = description;
    }

    public String getDescription() {
        return description;
    }

}
```

Tworzenie klasy

Tworzenie klasy FlyweightFactory, w której będzie cała logika omawianego wzorca. W metodzie createRelation sprawdzamy, czy obiekt o podanym opisie już istnieje. Jeśli nie, to tworzymy ten obiekt i wyświetlamy komunikat użytkownikowi. W przeciwnym wypadku pobieramy obiekt z tym parametrem.

```
public class FlyweightFactory {
    private Map<String, Relation> relations = Collections.synchronizedMap(new HashMap());

    public synchronized Relation createRelation(String description){
        Relation relation = (Relation)relations.get(description);
        if(relation == null) {
            relation = new Relation(description);
            relations.put(description, relation);
            System.out.println("Creating new relation: " + relation.getDescription());
        }
        return relation;
    }
}
```

Wywołanie pętli

Wywołanie w pętli tworzenia obiektów w fabryce pyłków.

```
public class Test {
    public static void main(String[] args) throws InterruptedException {
        FlyweightFactory flyweightFactory = new FlyweightFactory();
        for(int i = 0; i < 100; i++) {
            flyweightFactory.createRelation("Test");
        }
    }
}
```

Wynik aplikacji

Wynik aplikacji:

```
Creating new relation: Test
Process finished with exit code 0
```

Konsekwencje:
Pomimo 100 wywołań tworzenia obiektu, obiekt tworzy się tylko raz.
W pozostałych 99 przypadkach pobierany jest utworzony już obiekt.