Inżynieria oprogramowania

LAB 7 Wzorce projektowe. Wzorzec Prototyp

WZORCE PROJEKTOWE

Wzorzec projektowy (*ang. design pattern*) – w inżynierii oprogramowania, uniwersalne, sprawdzone w praktyce rozwiązanie często pojawiających się, powtarzalnych problemów projektowych. Pokazuje powiązania i zależności pomiędzy klasami oraz obiektami i ułatwia tworzenie, modyfikację oraz pielęgnację kodu źródłowego. Wzorzec projektowy nie jest gotową implementacją rozwiązania, lecz jego opisem. Stosowane są w projektach wykorzystujących programowanie obiektowe.

KLASYFIKACJA PODSTAWOWYCH WZORCÓW

Wzorce kreacyjne

- Budowniczy (obiektowy),
- Fabryka abstrakcyjna (obiektowy),
- Metoda wytwórcza (klasowy),
- Prototyp (obiektowy),
- Singleton (obiektowy);

Wzorce strukturalne

- Adapter (klasowy oraz obiektowy),
- Dekorator (obiektowy),
- Fasada (obiektowy),
- Kompozyt (obiektowy),
- Most (obiektowy),
- Pełnomocnik (obiektowy),
- Pyłek (obiektowy);

Wzorce czynnościowe

- Interpreter (klasowy),
- Iterator (obiektowy),
- Łańcuch zobowiązań (obiektowy),
- Mediator (obiektowy),
- Metoda szablonowa (klasowy),
- Obserwator (obiektowy),
- Odwiedzający (obiektowy),
- Pamiątka (obiektowy),
- Polecenie (obiektowy),
- Stan (obiektowy),
- Strategia (obiektowy).
-

PROTOTYP

Prototyp - w inżynierii oprogramowania jeden z obiektowych, konstrukcyjnych (kreacyjnych) wzorców projektowych, którego celem jest umożliwienie tworzenia obiektów danej klasy bądź klas z wykorzystaniem już istniejącego obiektu, zwanego prototypem. Głównym celem tego wzorca jest uniezależnienie systemu od sposobu w jaki tworzone są w nim produkty.

Omawiany wzorzec stosujemy między innymi wtedy, gdy nie chcemy tworzyć w budowanej aplikacji podklas obiektu budującego. Wzorzec ten stosujemy podczas stosowania klas specyfikowanych podczas działania aplikacji.

Wzorzec Prototyp powinien być używany, gdy:

- system powinien być niezależny od tego, jak jego produkty są tworzone, składane i reprezentowane;
- klasy, których egzemplarze należy tworzyć są specyfikowane w czasie wykonywania programu, np. przez dynamiczne ładowanie
- istnieje potrzeba uniknięcia budowania hierarchii klas fabryk, która jest porównywalna z hierarchią klas produktów
- stan obiektów klasy może przyjmować tylko jedną z kilku różnych wartości; może być wówczas wygodniej zainstalować odpowiednią liczbę prototypów i klonować je niż ręcznie tworzyć egzemplarze klasy za każdym razem z odpowiednim stanem.

Wzorzec prototypu określa rodzaj obiektów do tworzenia za pomocą prototypowej instancji. Prototypy nowych produktów są często budowane przed pełną produkcją, ale w poniższym przykładzie, prototyp jest bierny i nie bierze udziału w kopiowaniu siebie samego. Przykładem aktywnego prototypu (czyli biorącego udział w kopiowaniu siebie samego) jest biologiczny podział jednej komórki w dwie identyczne. Wtedy mamy do czynienia z klonowaniem.

PRZYKŁAD:

```
1  package wzorceProjektowePrototyp;
2
3  public class Cookie implements Cloneable {
4
5      private String Name;
6
7      public String getName() {
8          return Name;
9      }
10
11     public void setName(String name) {
12         this.Name = name;
13     }
14
15     public Object clone() {
16         try {
17             Cookie copy = (Cookie) super.clone();
18             return copy;
19
20         } catch (CloneNotSupportedException e) {
21             e.printStackTrace();
22             return null;
23         }
24     }
25 }
```

```java
package wzorceProjektowePrototyp;

public class CoconutCookie extends Cookie{

}
```

```java
package wzorceProjektowePrototyp;

public class CookieMachine {

  private Cookie cookie;

   public CookieMachine(Cookie cookie) {
     this.cookie = cookie;
   }

   public Cookie makeCookie() {
     return (Cookie)cookie.clone();
   }
}
```

```java
package wzorceProjektowePrototyp;

import java.util.ArrayList;

public class Prototyp {

    public static void main(String[] args) {

        ArrayList<Cookie> cookies = new <Cookie>ArrayList();

        Cookie tempCookie = null;
        Cookie prototyp = new CoconutCookie();
        CookieMachine cm = new CookieMachine(prototyp);

        for (int i = 1; i <= 100; i++) {
            tempCookie = cm.makeCookie();
            tempCookie.setName("CoonutCookie " + i);
            cookies.add(tempCookie);
        }

        for (Cookie cookie : cookies) {
            System.out.println(cookie.getName());
        }
    }
}
```

ZAD.1.

Utwórz projekt w NetBeans, który zawierał będzie klasy i metody zawarte w powyższym przykładzie. Przetestuj działanie tego programu.

ZAD.2.

Wzorując się na przedstawionym przykładzie utwórz klasę **Kartka** implementującą interfejs *Clonable*:

- Utwórz wewnątrz tej klasy prywatną zmienną tekstową **Text**

- Dodaj metody pozwalające na nadanie wartości tej zmiennej (**setText()**) oraz pobranie wartości z tej zmiennej (**getText()**)
- Dodaj metodę **kseruj()**, zwracającą klon klasy prototypu

Utwórz klasę **ProdukcjaKartek**:

- Utwórz wewnątrz tej klasy prywatną zmienną typu **Kartka**
- Utwórz konstruktor tej klasy, który będzie pobierał obiekt klasy **Kartka** i przypisywał jego wartość do zadeklarowanej wcześniej zmiennej prywatnej
- Utwórz metodę **zrobKsero()** zwracającą klon utworzony na zmiennej prywatnej

Utwórz klasę **Prototyp** i przetestuj działanie utworzonych klas i metod analogicznie jak w powyższym przykładzie