Wzorce Projektowe, lab. 3

Sklep
- String nazwa
+ String getNazwa() + Sklep(String) + String toString() + rzuciliTowar(String)

1. Mamy głęboką komunę. Zaimplementuj klasę **Sklep** zgodnie ze schematem obok. **Sklep** ma pole **nazwa**, które reprezentuje jego nazwę, konstruktor parametrowy oraz **toString'a()**. Dodatkowo ma metodę **rzuciliTowar(String)**, która wyświetla w konsoli komunikat diagnostyczny typu: „*Rzucono X do sklepu Y! Tłumy szaleją.*", gdzie *X* stanowi nazwę towaru dostarczoną w parametrze, a *Y* nazwę sklepu.
2. Zaimplementuj klasę **PaniKolejkowa**, zgodnie ze schematem obok. **PaniKolejkowa** ma metodę **kupujIleSięDa(Sklep)**, która wyświetla w konsoli komunikat diagnostyczny typu: „*Pani X wykupuje towar z Y. Siaty pełne.*", gdzie *X* to **imie** i **nazwisko** dzielnej pani, a *Y* to **nazwa** sklepu.

PaniKolejkowa
- String imie - String nazwisko
+ PaniKolejkowa (String, String) + toString(): String + kupujIleSięDa(Sklep)

3. Zaimplementuj mechanizm obserwacji zgodnie ze standardem Javy ze slajdów od 2 do 8 pliku **observer.pdf**. Obiekty klasy **Sklep** są obserwowane przez obiekty klasy **PaniKolejkowa.** Klasa **Sklep** (na schemacie odpowiada **ConcreteObservable**) dziedziczy z dostępnej w Javie klasy **Observable**. Klasa **PaniKolejkowa** (na schemacie odpowiada **ConcreteObserver**) implementuje interfejs **Observer**, który ma zadeklarowaną metodę **update(Observable, Object).** W celu podłączenia obserwatora, należy wywołać metodę **addObserver(Observer)**, podając jako argument referencję na obserwatora (do tego wystarczy, że klasa implementuje interfejs **Observer** - jest to *rzutowanie w górę*). Odłączenie obserwatora to analogiczne wywołanie **deleteObserver(Observer)**. W celu powiadomienia wszystkich obserwatorów danego obiektu wywołuje się metodę **notifyObservers()** (model pull, slajd 9) lub **notifyObservers(Object)** z argumentem zawierającym jakieś dane (model push, slajd 9). My, póki co, nie chcemy przekazywać żadnych danych, więc używamy tej pierwszej. Przed wywołaniem **notifyObservers()** należy jeszcze ustawić stan zmiennej **changed** odziedziczonej z klasy **Observable** na **true** za pomocą metody **setChanged()**. W obserwatorach (klasa **PaniKolejkowa**) implementujemy obowiązkową (z interfejsu) metodę **update(…)** wywołując tam metodę **kupujIleSięDa(Sklep)**.
4. Tworzymy kilka sklepów oraz kilka „pań kolejkowych", np. po 3. Następnie generujemy relacje wiele do wielu: jedna pani obserwuje kilka sklepów, jeden sklep jest obserwowany przez kilka pań. Wywołujemy dla sklepów metody **rzuciliTowar(String)**. Powinny one generować swoje własne komunikaty, a od razu po nich komunikaty odpowiednich obserwatorów.
5. Zmieniamy wywołanie metody **notifyObservers()** na **notifyObservers(Object)** tak, aby przekazywała w argumencie nazwę towaru, jaki do sklepu rzucono (model push). Zmieniamy sygnaturę lub przeciążamy metodę **kupujIleSięDa** na **(Sklep, String)**, aby mogła ona przyjąć nazwę towaru i wyświetlić komunikat diagnostyczny typu: „*Pani X wykupuje Y z Z. Siaty pełne.*", gdzie *X* to **imie** i **nazwisko** pani kolejkowej, Y to nazwa towaru, a *Z* to **nazwa** sklepu. Nazwy towaru nie przechowujemy w żadnym obiekcie.

UWAGI:

- Wszystkie 5 podzadań realizujemy w ramach jednego projektu Javy.
- Nie tworzymy własnej klasy **Observable** ani interfejsu **Observer**, korzystamy z gotowych!!!
- Implementacja interfejsu to rodzaj dziedziczenia, więc obiekty klasy **PaniKolejkowa** mogą być wskazywane przez referencje typu **Observer**.
- Od Javy 9 **Observable** i **Observer** są traktowane jako przestarzałe, o czym może ostrzegać kompilator. Należy to po prostu zignorować – działanie pozostaje niezmienione, a naszym celem jest przede wszystkim zrozumienie wzorca projektowego. Sam wzorzec nie został „wycofany" – po prostu jego funkcjonalność można teraz osiągnąć poprzez odpowiednie użycie **Listenerów** w swojej aplikacji. Idea jest analogiczna, jak tutaj, więc warto to zrozumieć.