Technologia Programowania 2017/2018
Lista 3 (ćwiczenia)

Jeśli w zadaniu nie ma innych instrukcji rozwiązania powinny być wyrażone za pomocą diagramów klas i diagramów sekwencji.

Wzorce Projektowe

Zadanie 1 — Wskaż zasady GRASP wykorzystane w kilku wzorcach GoF, np. Adapter, Factory, Proxy.

Zadanie 2 — Załóżmy, że implementujesz serwis do przeprowadzania rozgrywek pokera. Twoim zadaniem jest m.in. implementacja bota do gry. Chciałbyś uruchomić w ramach swojej aplikacji bota napisanego przez kolegę, jednak boty implementują inne interfejsy. Zaproponuj przykładowe interfejsy i rozwiązanie projektowe, które uchroni Cię przed koniecznością modyfikacji warstwy dziedzinowej. Wykorzystaj wzorzec **Adapter**.

Zadanie 3 — Tworzysz aplikacje, która pobiera wpisy z różnych portali społecznościowych i prezentuje je w jednym miejscu zgodnie ze Twoimi preferencjami. Oczywiście każdy portal wymaga innej implementacji funkcji `getMessage()` odpowiedzialnej za pobieranie wpisów. Ponadto chcesz mieć łatwy wpływ na treść wiadomości, np. obcinania wiadomości dłuższych niż 100 słów, cenzurowanie określonych słów. Wykorzystując wzorzec **Decorator** zaproponuj rozwiązanie, które umożliwi dynamiczne i elastyczne (tzn. bez tworzenia podklas) rozszerzenie funkcjonalności metody `getMessage()`. Podaj przykład sytuacji, w której kolejność „dekorowania" ma wpływ na końcową postać wiadomości.

Zadanie 4 — Piszesz program do automatycznego generowania raportów na temat klientów instytucji finansowej. Raporty mogą być potrzebne do różnych celów i w zależności od celu powinny zawierać jedynie wybrane informacje o klientach (np. nazwisko, adres, wiek, telefon, płeć, stan konta). Ponadto Twój program ma współpracować z wieloma systemami i potrzebnych będzie wiele różnych reprezentacji reportu, dla których trudno ustalić wspólny interfejs (np. HTML, JComponent). W oparciu o wzorzec **Builder** zaproponuj rozwiązanie.

Zadanie 5 — Rozwiązanie podobne do wzorca **Singleton** polega na stworzeniu klasy, której wszystkie metody są statyczne. Jakie są zalety i wady takiego rozwiązania.

Zadanie 6 — Na przykładzie wyjaśnij na czym polega double-checked locking. Jakie są korzyści zastosowania takiego rozwiązania w implementacji wzorca Singleton? W jaki sposób można zaimplementować wzorzec **Singleton** za pomocą typu wyliczeniowego *enum*?

Zadanie 7 — Istnieje wiele typów urządzeń, ale każde urządzenie może znajdować się w jednym z czterech stanów: ON, OFF, $STARTED$, $STOPPED$. Wszystkie urządzenia mają wspólny interfejs zawierający metody: $start()$, $stop()$, $turnOn()$, $turnOff()$. Rezultatem wywołania danej metod może być zmiana stanu. To, czy zmiana nastąpi i jaki nowy stan przyjmie urządzenie zależy od jego aktualnego stanu. Np. polecenie $start()$ w stanie OFF nie zmienia stanu urządzenia, polecenie $turnOff()$ w stanie $STOPPED$ zmienia stan na OFF.

a) Narysuj diagram stanów UML z możliwymi przejściami między stanami.

b) Wykorzystaj wzorzec **State** by uniknąć instrukcji warunkowych wewnątrz metod $start()$, $stop()$, $turnOn()$, $turnOff()$. Narysuj diagram klas UML ilustrujący Twoje rozwiązanie.

c) Czy klasy stanów mogłyby być singletonami?

Zadanie 8 — W systemie bankowym różni pracownicy banku powinni mieć różne prawa dostępu do informacji dotyczących kont klientów (np. o właścicielu, stanie konta, dokonywanych płatnościach). Wykorzystaj wzorce **Proxy** i **Factory** by zaproponować rozwiązanie, umożliwiające łatwą kontrolę dostępu i łatwe wprowadzanie nowych poziomów uprawnień. Rozwiązanie powinno być zgodne z *open-close principle*.

Zadanie 9 — W oparciu o wzorce **Singleton, Factory, Strategy, Composite** zaproponuj rozwiązanie umożliwiające dynamiczny wybór strategii licytacji w rozgrywce pokera. Strategie mogą być złożone, dla przykładu `MySuperCompositeStrategy` może być strategią opierającą się o dwie strategie proste: `BluffStrategy` oraz `AggressiveStrategy`. Wykorzystaj refleksje, by zaproponować rozwiązanie umożliwiające zmianę strategii w trakcie rozgrywki.

Zadanie 10 — Tworzysz narzędzie do automatycznego generowania raportów. Wszystkie raporty składają się z trzech części: *header*, *footer* oraz *body*. Trudno jest przewidzieć z góry jaka reprezentacja raportu będzie potrzebna (np. plain text, HTML, XML), ale możesz założyć, że klasa reprezentująca daną cześć raportu jest zgodna z góry ustalonym interfejsem (np. każda klasa nagłówka musi mieć metody ustalające i zwracające tytuł i datę raportu jako odpowiednio sformatowany *String*). Wykorzystaj wzorzec **Abstract Factory** by zaproponować rozwiązanie umożliwiające oddzielenie w kodzie programu reguł wyboru i tworzenia elementów raportu (rodziny powiązanych produktów) od pozostałej części logiki. Rozwiązanie powinno być zgodne z *open-close principle*.

Zadanie 11 — Tworzysz aplikację dla domu maklerskiego. Aplikacja ma wysyłać powiadomienia o istotnych zmianach cen akcji do wszystkich zainteresowanych klientów. Klienci mogą w dowolnej chwili zapisać się na listę zainteresowanych akcjami danej spółki i wypisać się z niej. Ponadto klienci mogą być indywidualni (jedna osoba) lub instytucjonalni (wiele osób). Zaproponuj rozwiązanie oparte na **Observer** oraz **Composite**.

Zadanie 12 — Piszesz program dla międzynarodowej firmy reklamowej. Program wybiera z bazy losowego klienta i losową reklamę. Reklama przed wysłaniem jest tłumaczona na preferowany przez klienta język. Ponadto w zależności od tego ile zapłacił reklamodawca, reklama może być wysyłana za pomocą listu elektronicznego, wiadomość sms lub wiadomości głosowej odczytanej przez syntezator mowy.

a) Zaproponuj rozwiązania oparte o **TemplateMethod**. Możesz założyć, że metoda szablonowa `deliver` zawiera wywołania metod `translate` oraz `send`. Ile klas musisz stworzyć, żeby uwzględnić wszystkie możliwe kombinacje implementacji tych dwóch metod?

b) Zaproponuj rozwiązanie oparte na **Strategy**. Które z rozwiązań będzie Twoim zdaniem lepsze?

Zadanie 13 — Tworzysz prosty edytor tekstowy, który powinien umożliwiać operacje wstawiania i usuwania znaków, a także cofania i ponawiania operacji (*undo* i *redo*). Przeanalizuj dokładnie i wyjaśnij rozwiązanie oparte na wzorcu **Command** zaproponowane w książce Holub on Patterns.

J.L.