Wzorce projektowe II

Wstęp

Na drugich zajęciach poświęconych wzorcom będziemy pisać fragmenty kodu UKI – Uniwersalnego Komunikatora Internetowego.

UKI jest systemem, który realizuje podstawowe operacje dostarczane przez tego typu programy:

- łączenia i rozłączania się z siecią,
- wyświetlania znajomych, dodawania nowych i usuwania istniejących,
- łączenia znajomych w wielopoziomowe grupy,
- przeprowadzania rozmowy z jednym lub wieloma znajomymi,
- szyfrowania komunikacji,
- zmieniania wyglądu („skórki”).

Cała reszta funkcjonalności zapewniana jest przez dodatki (plug-iny). Wtyczki odpowiadają m.in. za realizację specyficznych protokołów komunikacyjnych (np. Gadu Gadu, Jabber, IRC).

Waszym zadaniem jest zaproponowanie rozwiązania konkretnych wyzwań projektowych korzystając z wzorca, który pasuje do aktualnej sytuacji.

Pierwszym krokiem jest dokonanie odpowiedniego przydziału wzorców do zadań. Kierując się wiedzą z wykładu, treścią kodu, wsparciem prowadzących oraz intuicją proszę określić bijekcję (podpowiedź z matematyki: funkcję wzajemnie jednoznaczną) pomiędzy wyzwaniami a wzorcami projektowymi:

Abstract Factory (Fabryka abstrakcyjna)	Reagowanie na przychodzącą wiadomość zależnie od statusu
Composite (Kompozyt)	Reagowanie wtyczek na zmiany statusu
Decorator (Dekorator)	Szyfrowanie komunikacji
Observer (Obserwator)	Wielopoziomowe grupy znajomych
Singleton	Zarządzanie ustawieniami aplikacji
State (Stan)	Zmiana wyglądu aplikacji („skórki”)

Następnie dla każdego wzorca (zgodnie z kolejnością poniżej) należy napisać fragment kodu, który implementuje podstawowe aspekty tego wzorca dla powiązanego z nim wyzwania:

- Singleton
- Composite
- AbstractFactory
- State
- Listener
- Decorator

Sugerowane tempo pracy to trzy wzorce na jedne zajęcia.

Podczas pracy można posiłkować się materiałami z Internetu (np. anglojęzyczną Wikipedią), ale proszę konsultować nabywaną wiedzę z prowadzącym.

Opis kodu

Fragment kodu oznaczony

```
// ASSUME_PROVIDED
```

oznacza, że można założyć dany fragment funkcjonalności za zaimplementowany, nie trzeba go

dopisywać. Komentarz

```
// TASK
```

oznacza miejsce, od którego warto zacząć implementację danego wyzwania projektowego, a komentarz

```
// TODO
```

oznacza konieczność napisania w tym miejscu konkretnego fragmentu kodu.

Opis wyzwań

Przy każdym wyzwaniu podano tzw. punkty zainteresowania: miejsca w projekcie, od których warto zacząć czytanie kodu i implementowanie wzorców.

Reagowanie na przychodzącą wiadomość zależnie od statusu

W zależności od aktualnego statusu (widoczny, zajęty, ukryty) program powinien różnie reagować na przychodzącą od znajomego wiadomość.

- Przy statusie „widoczny” nadejście wiadomości powoduje otwarcie okna rozmowy i wpisanie do niego treści wiadomości, która nadeszła.
- Przy statusie „zajęty” nadejście wiadomości powoduje odesłanie do nadawcy informacji, że odbiorca jest nieobecny i prosi o kontakt, kiedy wróci.
- Przy statusie „ukryty” nadejście wiadomości nie powoduje żadnej akcji; wiadomość jest ignorowana.

Same operacje wspomniane powyżej nie są istotne (można użyć `ASSUME_PROVIDED` w odpowiednim miejscu w kodzie), ważna jest implementacja odpowiedniego wzorca.

Punkty zainteresowania:

`io.patterns.UKI.buddies.User`, `io.patterns.UKI.buddies.Status`

Reagowanie wtyczek na zmiany statusu

Wtyczki powinny reagować na pewne zdarzenia „globalne” w aplikacji, np. gdy użytkownik zmieni w interfejsie aplikacji swój status, wtyczka powinna zostać o tym poinformowana, żeby odpowiednio zgłosić zmianę statusu w swojej natywnej sieci.

Zakładamy tutaj uproszczone wymaganie informowania **tylko** o zmianie statusu; inne zdarzenia, o których powinny wiedzieć wtyczki nas w tym momencie nie interesują.

Punkty zainteresowań:

`io.patterns.UKI.comm.Protocol`, `io.patterns.UKI.comm.ExemplaryPlugin`

Szyfrowanie komunikacji

Aplikacja umożliwia szyfrowanie treści wysyłanych wiadomości i odszyfrowywanie treści otrzymanych wiadomości (jeśli są one zaszyfrowane).

Szyfrowanie **nie** powinno odbywać się na poziomie protokołu sieciowego, ponieważ wiele sieci albo nie wspiera szyfrowania, albo jakość szyfrowania, jakie proponują jest poniżej krytyki.

Zamiast tego szyfrowana jest sama treść wpisana przez użytkownika i jako „binarne śmieci” jest przekazana dalej do wysłania.

Szyfrowanie odbywa się przy użyciu klasy `ManyUtils`. Każdy kontakt na liście może posiadać swoje własne hasło, które jest używane jako klucz szyfrowania. Szyfrowanie może być selektywnie włączane i wyłączane podczas pracy programu, dla każdego użytkownika z osobna.

Dodatkowo, aby oszczędzić czas, można selektywnie kompresować wiadomości. Podobnie jak dla szyfrowania, kompresja powinna odbywać się na poziomie treści wiadomości, nie protokołu sieciowego.

Kompresję można przeprowadzić z pomocą klasy `ManyUtils`.

Punkty zainteresowań:

`io.patterns.UKI.comm.Protocol`, `io.patterns.UKI.messages`

Wielopoziomowe grupy znajomych

Dużym atutem UKI jest możliwość grupowania znajomych. Każda grupa może zawierać użytkowników albo inne grupy.

Nie tylko do pojedynczej osoby, ale też do grupy można wysłać wiadomość. Trafia wtedy ona do każdej osoby będącej w tej grupie, lub w którejś z podgrup. Duży zysk dla pracowników działu marketingu!

Zakładamy, że w aplikacji jest już odpowiedni mechanizm uniemożliwiający tworzenie cykli.

Należy napisać kod obsługujący takie grupy i wysyłanie do nich wiadomości.

Punkty zainteresowań:

`io.patterns.UKI.buddies.Group`, `io.patterns.UKI.buddies`

Zarządzanie ustawieniami aplikacji

Aplikacja powinna umożliwiać konfigurowanie zachowania. Zachowania są globalne dla całej aplikacji.

Pojedyncze ustawienie jest parą klucz-wartość. Ograniczamy się do kluczy i wartości będących napisami.

Ustawienia mogą być zapisywane i wczytywane z pliku znajdującego się w katalogu domowym użytkownika. Jeżeli pliku nie da się odczytać, należy „po cichu” pominąć obsługę pliku i operować tylko na danych w obiekcie. Jeżeli plik był odczytany, to po zmianie ustawień należy go zapisać. (podpowiedź: wesprzeć się standardową klasą `java.util.Properties` oraz metodą `System.getProperty`)

Punkty zainteresowań:

`io.patterns.UKI.Settings`

Zmiana wyglądu aplikacji („skórki”)

Najważniejszą (*sic!*) cechą UKI jest możliwość zmiany „skórek”. Skórka obejmuje takie elementy interfejsu jak:

- lista znajomych,
- okno wiadomości,
- wygląd predefiniowanych ikon (`STATUS_ACTIVE`, `SMILEY`, itd.).

Zastosowanie wzorca polega na eleganckim tworzeniu interfejsu użytkownika poprzez pobieranie podczas tworzenia odpowiednich „uspójnionych” elementów.

Dla uproszczenia można założyć, że skórki będą dwie, „funny” oraz „simple”, a to, jaka skórka będzie użyta zależy od ustawienia aplikacji `uki.theme` (patrz wyzwanie „Zarządzanie ustawieniami aplikacji”).

Punkty zainteresowań:

`io.patterns.UKI.gui.UIManager`, `io.patterns.UKI.gui`