Programowanie I

Zajęcia nr 13

Rafał Masełek

2. czerwca 2022r.

Wstęp

Nie warto wymyślać koła od nowa ani wyważać otwartych drzwi. Podobnie jest z programowaniem. Pewnego rodzaju problemy są uniwersalne i pojawiają się niezależnie od rodzaju programu. Np. jak napisać program wykorzystujący obiekty róznych klas tak, żeby łatwo dało się go rozwijać poprzez dodanie nowych typów? Ten problem może dotyczyć typów postaci w grze cRPG, kształtów dostępnych w programie rodzaju Microsoft Paint, czy analizie fizycznej, w której cząstki są reprezentowane przez klasy. Uniwersalne problemy powinny mieć uniwersalne rozwiązania i takimi są wzorce projektowe. Na dzisiejszych zajęciach poznacie trzy wzorce: Singleton, Fabryka i Proxy. Tak naprawdę jeden wzorzec, Dekorator, już poznaliście, ponieważ w Pythonie ma specjalną składnię (pamiętacie?). Wzorce projektowe zazwyczaj nie są uczone na podstawowych kursach programowania, ale ponieważ ten kurs ma R w nazwie, więc możemy się nimi zająć. Wzorzec projektowy to nie jest element języka, ale sposób pisania programu, który ma zapewnić łatwość utrzymania i rozwijania programu, dlatego znajomość wzorców projektowych jest ceniona przy aplikowaniu o pracę w IT. Na razie mogą wam się wydawać czymś nadmiarowym, niepotrzebną kompikacją, ale to dlatego że: a) programy, które piszecie są bardzo krótnie i proste b) nie musicie pracować z kodem pisanym przez innych.

Zadanie 1 – Singleton

Singleton to wzorzec projektowy stosowany wtedy, gdy chcemy mieć pewność, że powstanie co najwyżej jeden obiekt danej klasy w programie. Np. wyobraźmy sobie, że napisaliśmy grę MMO, w której gracze rywalizują ze sobą w turniejach. W każdej potyczce jeden z graczy jest wyróżniony, np. jako zwycięzca poprzedniej tury ma specjalny bonus. Mechanika gry zakłada, że tylko jeden z graczy będzie wyróżniony. Jednakże błąd programisty mógłby spowodować wyróżnienie większej ilości, dlatego przyda się kod napisany w taki sposób, żeby taka pomyłka była niemożliwa. Można tutaj użyć wzorca singleton.

W zadaniu skupimy się na implementacji samego wzorca w oderwaniu od konkretnego zastosowania. W tym celu:

- Stwórz klasę Singleton, która zawiera pole "__instance"
- Konstruktor klasy Singleton ma dwojakie działanie: jeżeli pole "__instance" jest puste, to nowy obiekt zostanie utworzony i referencja do niego będzie zapisana w polu "__instance"; jeżeli pole ma już jakąś wartość (istnieje obiekt typu Singleton) to konstruktor rzuci wyjątek informujący o niemożności utworzenia nowego obiektu.
- Dodatkowo, klasa Singleton posiada metodę statyczną "getInstance()", która zwraca referencję do obiektu Singleton. Jeżeli obiekt nie istnieje, to metoda go najpierw utworzy a potem zwróci.

W programie użyj konstruktora i metody getInstance kilka razy i za każdym razem wyprintuj uzyskane obiekty. Przekonaj się, że za każdym razem masz do czynienia z tym samym obiektem. Sprawdź, czy da się stworzyć drugi (inny) obiekt.

Zadanie 2 – Fabryka

Fabryka to jeden z najpopularniejszych wzorców projektowych pozwalający na przejrzyste i łatwe rozbudowywanie istniejącego programu tak, żeby obsługiwał nowe typy obiektów bez zmieniania interfejsu i sposobu interakcji z użytkownikiem. Najłatwiej zrozumieć na przykładzie, daltego spójrz na kod poniżej:

```
class FrenchLocalizer:
        """ it simply returns the french version """

        def __init__(self):

                self.translations = {"car": "voiture", "bike": "bicyclette",
                                                        "plane":"avion"}

        def localize(self, msg):

                """change the message using translations"""
                return self.translations.get(msg, msg)

class SpanishLocalizer:
        """it simply returns the spanish version"""

        def __init__(self):
                self.translations = {"car": "coche", "bike": "bicicleta",
                                                        "plane":"avion"}

        def localize(self, msg):

                """change the message using translations"""
                return self.translations.get(msg, msg)

class EnglishLocalizer:
        """Simply return the same message"""

        def localize(self, msg):
                return msg
```

Podany kod jest częścią programu, który pozwala na tłumaczenie słów z języka angielskiego na francuski bądź hiszpański. Dla przejrzystości, słownik zawiera jedynie 3 słowa: samochód, rower i samolot. Każdy z obiektów reprezentuje jeden ze słowników. Metoda "localize" zwraca tłumaczenie słowa, jeżeli jest ono w słowniku, lub podane słowo, jeżeli go nie rozpozna. Możemy sobie wyobrazić, że w głównej części programu (do napisania dla was) użytkownik jest proszony o podanie języka, na który chce tłumaczyć, wtedy tworzony jest odpowiedni obiekt, a następnie użytkownik może tłumaczyć słowa aż do zakończenia programu (np. przez podanie "q"). Co jeżeli zamiast 3 języków mamy 300? Co jeżeli w pewnym momencie chcemy dodać kolejne? Dobrze napisać program tak, żeby:

- sposób obsługi programu przez użytkownika nie uległ zmianie
- główna część programu, w której prosimy użytkownika o podanie słów i je wczytujemy pozostała czytelna
- obsługa wszystkich obiektów była identyczna

Można w tym celu użyć wzorca Fabryki, który polega na stworzeniu wspólnego interfejsu do obsługi wszystkich zdefiniowanych typów. W tym celu zdefiniuj funkcję[1] Fabryka, która przyjmuje jeden argument: nazwę języka. W funkcji następuje sprawdzenie, czy dany język jest rozpoznawany, jeżeli nie, to wyświetla komunikat, listę dostępnych języków i kończy program. Jeżeli język jest wspierany, to funkcja zwraca obiekt odpowiedniej klasy. Korzystając z tego wzorca mamy tylko jedno miejsce w kodzie, które zmieniamy po dodaniu nowego typu: funkcję Fabryka.

Zaimplementuj:

[1] Można użyć klasy do zaimplementowania wzorca, ale my użyjemy funkcji.

- funkcję Fabryka
- słownik dla języka polskiego
- główną część programu aka "main", w którym prosisz użytkownika o podanie słownika a następnie tłumaczysz dowolną ilość słów aż do podanie "q"

Zadbaj o to, żeby program rozpoznał podaną nazwę języka bez względu na wielkość użytych liter.

Dla chętnych: W zadaniu tłumaczymy z angielskiego na inne jezyki, ale dlaczego nie umożliwić tłumaczenia w drugą stronę? Rozbuduj program tak, żeby można było tłumaczyć z jednego języka na inny, gdzie użytkownik wybiera dwa języki. Rozbuduj program jeszcze bardziej, żeby umożliwić wykrywanie języka, w którym podane jest słowo, tak, żeby użytkownik musiał podać tylko język docelowy. Zachowaj wzorzec fabryki.

Zadanie 3 – Proxy

Spójrz na kod poniżej, który stanowi szkielet programu wczytującego obrazki.

```
class Image:
    """ class for image loading and displaying """
    def __init__( self, filename ):
        self._filename = filename

    def load_image_from_disk( self ):
        print("loading " + self._filename )
        # here comes the code that loads image

    def display_image( self ):
        print("display " + self._filename)
        # here comes the code that actually displays the image
```

Powyższy kod jest oczywiście schematyczny, ale można dostrzec pewien problem. Klasa zawiera dwie metody, jedną do wczytywania danych i jedną do ich wyświetlania. Generalnie to dobre rozwiązanie, jeżeli chcemy wyświetlić obrazek dwa razy to nie musimy go wczytywać dwa razy. Ale może się zdarzyć, że gdzieś w dużym programie pogubimy się i nie będziemy wiedzieć, czy obrazek został już wczytany. Np. użytkownik wczytuje sobie obrazki, wyświetla je, później znowu chce wczytać i wyświetlić. Są dwa ryzyka:

1. ponownie wczytamy ten sam obrazek tracąc czas i pamięć
2. spróbujemy wyświetlić obrazek, którego nie ma w pamięci i nastąpi błąd

Rozwiązaniem jest przechowywanie informacji o tym, jakie obrazki zostały wczytane, a jakie nie. Moglibyśmy zmodyfikować klasę Image, ale czasami nie mamy takiej możliwości, bo np. korzystamy z klasy zdefiniowanej w zewnętrznej bilbiotece. Moglibyśmy przepisać definicję klasy z zewnętrznej biblioteki i ją modyfikować, ale jak już pewnie rozumiecie, to słabe rozwiązanie. Znacznie lepiej skorzystać ze wzorca projektowego Proxy. W tym celu należy stworzyć dwie klasy: Proxy i ProxyImage.

Klasa Proxy jest bardzo ogólna i użyjemy jej do tworzenia klas pochodnych korzystających ze wzorca projektowego. Klasa ta ma dwa pola, jedno zawiera referencję do obiektu podwanego jako argument konstruktora, a drugie zawiera informację o "stanie" obiektu Proxy i może być np. liczbą całkowitą.

Druga klasa, ProxyImage, dziedziczy po klasie Proxy i zawiera tylko jedną metodę: "display_image" – taka sama nazwa jak w klasie Image. Wewnątrz tej metody sprawdź stan Proxy i w zależności od niego zrób jedną z dwóch rzeczy:

1. wczytaj obrazek z dysku (korzystając z odpowiedniej metody klasy Image na rzecz obiektu zapisanego w polu Proxy), zmień stan i wyświetl

2. wyświetl obrazek, który już został wczytany

W głównej części programu stwórz dwa obiekty typu Image, odpowiadające im dwa obiekty ProxyImage, i wywołaj kilkukrotnie "display_image". Dodaj odpowiednie instrukcje print i sprawdź, że wczytane już obrazki nie są wczytywane ponownie.

Dla chętnych: W naszym programie tak naprawdę nie czytamy żadnych obrazków. Zmień to korzystając np. z biblioteki matplotlib i zmierz czas wykonania programu z użyciem ProxyImage i bez.