Programowanie w języku JAVA
Zajęcia laboratoryjne 1

Wzorce projektowe

1. Zasady zaliczenia przedmiotu

Kontakt do prowadzącego: wojtek.skwierawski@gmail.com

- Zajęcia laboratoryjne są obowiązkowe. Nieobecność wpływa na ocenę końcową.
- Nieobecność należy usprawiedliwić i nadrobić tj. wykonać samodzielnie kod z laboratorium i przesłać go do prowadzącego.
- Jeżeli student nie skończy zadań na zajęciach, zobligowany jest do dokończenia ich samodzielnie w domu. Brak kodu oznacza obniżenie oceny końcowej
- Na początku zajęć prowadzący wybierą w sposób losowy kilka osób i zadaje im kilka pytań dotyczących podstaw programowania w języku JAVA.
- Studenci dostaną do wykonania prosty projekt w języku JAVA. Termin oddania pokrywa się z terminem ostatnich zajęć laboratoryjnych. Spóźnienie się z oddaniem oznacza obniżenie oceny.
- Ocena końcowa jest wystawiana na podstawie kodu studenta obniżona o ewentualne nieobecności i wyniki z odpytywania.
- Na początku zajęć jeden ze studentów będzie omawiał kod, który napisał podczas poprzednich zajęć.

2. Zadania do wykonania

Wzorzec projektowy *Singleton*

Zadanie 1. W wybranym przez siebie środowisku programistycznym utwórz nowy projekt.

Zadanie 2. Do projektu dodaj nową klasę o nazwie: *MyLogger*. Dodaj w tej klasie polę *path*, które potem przechowywać będzie ścieżkę do pliku z raportem z działania aplikacji.

Zadanie 3. Dodaj do klasy dwa konstruktory (metody tworzące obiekt). Pierwszy przyjmujący ścieżkę do pliku jako parametr i drugi bez parametrów.

Zadanie 4. W klasie *MyLogger* napisz metodę przyjmującą jako parametr treść raportu (*String*) i zapisującą go do pliku razem z aktualna datą i godziną . Jeżeli plik z raportami nie istnieje metoda powinna go stworzyć. (Wyszukaj w internecie sposób na pobranie w programie aktualnego czasu oraz metody na poprawne sformatowanie). Wypróbuj działanie swojej aplikacji podając wybraną ścieżkę na dysku komputera.

Przykładowy raport:

19:42:46 16/05/2019 | Treść raportu

Zadanie 5. W metodzie do raportowania dodaj sprawdzanie czy pole ścieżki zostało zdefiniowane. Jeżeli nie to wyrzuć nowy wyjątek typu *IOException* z komunikatem o braku ścieżki. W klasie Main spróbuj utworzyć nową instancję klasy za pomocą konstruktora bez parametru. Następnie wykonaj dla niej metodę do zapisu raportu do pliku. Zaobserwuj wyjątek zgłoszony przez aplikację.

Zadanie 6. Przekształć klasę *MyLogger* tak aby jej konstruktory były dostępne wyłącznie w klasie *MyLogger* - zmień poziom dostępu tych metod. Zamiast konstruktora z parametrem napisz metodę do ustawiania ścieżki w klasie *MyLogger* (metoda *setPath*) oraz metodę do zwracania pola *path*. (metoda *getPath*)

Zadanie 7. W klasie *MyLogger* dodaj prywatne statyczne pole typu *MyLogger* o nazwie instance przyjmujące początkowo wartość null.

Zadanie 8. W klasie *MyLogger* dodaj statyczną metodę *getInstance*, która zwracać będzie polę *instance*. Jeżeli wcześniej to pole nie zostało zainicjalizowane to ta metoda powinna stworzyć nowy obiekt przed zwróceniem go. (Jeżeli jesteś zainteresowany zapytaj prowadzącego o *lazy initialization*).

Zadanie 9. Zmień klasę *MyLogger* tak aby żadna klasa nie mogła dziedziczyć po niej. Wyszukaj w internecie odpowiednie słowo kluczowe, które do tego służy. Zastanów się dlaczego we wzorcu *Singleton* uniemożliwia się dziedziczenie

Zadanie 10. W klasie *Main* wykorzystaj metodę getInstance kilkukrotnie aby mieć kilka zmiennych typu *MyLogger* tak jak w przykładzie poniżej.

```
MyLogger logger1 = MyLogger.getInstance();

MyLogger logger2 = MyLogger.getInstance();

MyLogger logger3 = MyLogger.getInstance();
```

Następnie ustaw ścieżkę do pliku z raportami tylko i wyłącznie dla logger1 (metoda setPath) Potem wykorzystaj metodę do zapisywania raportu do pliku dla logger1, logger2, i logger3.

Tak jak w przykładzie poniżej:

```
logger1.AppendLog("Logger 1 zapisuje do pliku");

logger2.AppendLog("Logger 2 zapisuje do pliku");

logger2.AppendLog("Logger 3 zapisuje do pliku");
```

Otwórz plik z raportami i sprawdź czy wszystkie raporty poprawnie zostały zapisane do pliku.

Zadanie 11. Sprawdź i wypisz w konsoli czy wszystkie “loggery” wskazują na dokładnie ten sam obiekt. (Użyj do tego operatora ==, a następnie wypisz na ekran wartość typu boolean).

Podsumowanie

Singleton używany jest kiedy chcemy zapewnić globalny dostęp do obiektu w naszej aplikacji oraz mieć pewność , że zawsze odwołamy się do tej samej instancji tego obiektu. Przykładowo możemy zastosować singleton kiedy korzystamy w aplikacji z zewnętrznego hardware’u. Klasa zarządzająca połączeniem i komunikacją z tym urządzeniem powinna pilnować aby nikt nie mógł równolegle utworzyć kilku połączeń i tym samym zakłócić prawidłową komunikację.

Podstawowymi elementami singletona są:

1. Prywatne konstruktory - Na zewnątrz singletona nie ma możliwości utworzenia jego instancji.
2. Prywatne statyczne pole z instancją obiektu i publiczna statyczna metoda zwracająca instancję - Za każdym razem jak chcemy skorzystać z naszego obiektu musimy odpytać klasę singletona o to aby zwróciła nam instancję.

Wzorzec projektowy *Fabryka*

Zadanie 1. Dodaj w projekcie abstrakcyjną klasę *Animal* z polem *name* oraz konstruktorem ustawiającym to pole

Zadanie 2. Dodaj abstrakcyjną metodę *makeSound*. Metoda ta powinna wypisać w konsoli dźwięk wydawany przez to zwierzę.

Zadanie 3. Stwórz klasy: *Lion*, *Crow*, *Whale* tak aby dziedziczyły one po klasie *Animal*.

Zadanie 4. Dodaj nową klasę o nazwie *AnimalFactory*. Utwórz w niej metodę, która przyjmuje jako parametr nazwę zwierzęcia do stworzenia (String) i zwraca typ *Animal*. Wewnątrz tej metody wykorzystaj instrukcję *switch-case*. W zależności od wartości parametru zwróć inny rodzaj zwierzęcia.

Jeżeli użytkownik wpisze EXIT aplikacja powinna przestać odpytywać użytkownika i zakończyć działanie.

Zadanie 5. W klasie głównej napisz metodę która będzie odpytywać w pętli użytkownika o to jakie zwierze chce stworzyć a następnie wykorzysta metodę makesound.

Zadanie 6.* Stwórz nowy typ wyliczeniowy, który przypiszę wartość liczbową dla każdej z podklas klasy *Animal*.

Zadanie 7.* Zmień kod aplikacji tak aby przyjmowała ona od użytkownika wartość liczbową zamiast *stringa*, a następnie zamieniła go na typ wyliczeniowy i tworzyła instancję klasy *Animal*.

Podsumowanie

Wzorzec fabryka wykorzystujemy kiedy chcemy zarządzać tworzeniem obiektów, które posiadają na tyle wiele wspólnych cech, że dziedziczą po wspólnej klasie abstrakcyjnej albo implementują wspólny interfejs. Dzięki wykorzystaniu fabryki możemy dodawać kolejne typy obiektów do naszej aplikacji bez zmiany w kodzie, który korzysta ze stworzonych obiektów.

Wzorzec projektowy *Fasada*

Do wykorzystania wzorca fasada posłużymy się przykładem prostej aplikacji sklepu z meblami. Klient próbuje kupić w sklepie meble na raty. Sklep zanim sprzeda klientowi towar musi sprawdzić 3 warunki.:

- Klient musi mieć zgodę swojego banku na zakup.
- W magazynie muszą być dostępne meble, zamawiane przez klienta.
- Firma dostawcza powinna zgodzić się na dowóz do miejsca zamieszkania klienta,

Zadanie 1. Dodaj do projektu nową klasę *Order*. Powinna zawierać ona pola *address, clientID* oraz *furnitureID*. Natomiast konstruktor powinien przyjmować te trzy parametry.

Zadanie 2. Dodaj trzy nowe klasy: *Bank, Warehouse* i *Delivery*. Dodaj do nich odpowiednie metody weryfikujące odpowiednie warunki. Na potrzeby aplikacji wymuś aby weryfikacja, za każdym razem zachodziła prawidłowo. Wypisz w konsoli efekt weryfikacji.

Zadanie 3. W klasie Main dodaj kod programu tworzący nowe zamówienie (uzupełnij je losowymi danymi) oraz sprawdzający wszystkie warunki. Zauważ, że teraz aby sprawdzić zamówienie potrzebujesz stworzyć samodzielnie instancje trzech klas oraz trzykrotnie wywołać sprawdzenie. Z perspektywy klienta jest to niepotrzebnie skomplikowane zachowanie.

Zadanie 4. Dodaj nową klasę *ValidationFacade*, która jako pola będzie posiadała 3 klasy weryfikujące i samodzielnie tworzy ich instancje w swoim konstruktorze.

Zadanie 5. W nowej klasie napisz metodę, która przyjmować będzie klasę *Order* jako parametr i sprawdzi poprawność trzech warunków. Jeżeli wszystkie zostały spełnione powinna zwracać logiczną wartość true;

Zadanie 6. W klasie *Main* wykorzystaj przed chwilą napisany kod do weryfikacji nowego zamówienia. Zauważ, że z perspektywy klienta (albo programisty piszącego aplikacje dla sklepu) taka procedura jest o wiele prostsza i bardziej intuicyjna.