Laboratorium
Projektowanie Obiektowe – Wzorce Projektowe

Temat : *Strukturalny wzorzec projektowy Adapter – problem niedopasowania impedancji*

Historia zmian

Data	*Wersja*	*Autor*	*Opis zmian*
22.02.2010	0.1	Tomasz Kowalski	Utworzenie dokumentu i edycja wprowadzenia
…	...	…	...
7.5.2013	3.0	Tomasz Kowalski	Aktualizacja związana ze zmianą kolejności laboratoriów
5.4.2014	4.0	Tomasz Kowalski	Aktualizacja do Mavena + wyciągnięcie wprowadzenia
9.4.2015	5.0	Dominik Żurek	Aktualizacja aplikacji i zmiana nazw na język angielski
20.4.2016	5.1	Tomasz Kowalski	Poprawki i aktualizacja treści
9.5.2016	6.0	Tomasz Kowalski	Połączenie i modyfikacja zadań
6.10.2016	6.1	Tomasz Kowalski	Zmiana repozytorium z svn-a na github
29.3.2017	6.2	Tomasz Kowalski	Instrukcja i kody na organizacji na github
1.3.2018	6.3	Tomasz Kowalski	poprawka dot. lokalizacji projektu na github
3.4.2018	7.0	Tomasz Kowalski	Aktualizacja projektu i biblioteki PlotSoftBase
29.5.2018	7.1	Tomasz Kowalski	Wydzielenie pytań
28.10.2018	7.2	Tomasz Kowalski	Poprawka nazwy klasy i metody
1.4.2019	8.0	Tomasz Kowalski	Zmiany dotyczące dziedziny problemowej.
21.2.2020	8.1	Tomasz Kowalski	Drobne aktualizacje dot. czasu pracy i nazwy projektu

1. Cel laboratorium

Głównym celem laboratoriów jest zapoznanie się z wzorcem projektowym: *Adapter*. Należy on do grupy wzorców strukturalnych. Zajęcia powinny pomoc studentom rozpoznawać omawiane wzorce w projektach informatycznych, samodzielnie implementować wzorce oraz dokonywać odpowiednich modyfikacji wzorca w zależności od potrzeb projektu.
Czas realizacji laboratoriów wynosi ~3 godziny.

2. Zasoby

2.1. Wymagane oprogramowanie

Polecenia laboratorium będą dotyczyły programowania wzorców w języku Java. Potrzebne będzie środowisko dla programistów (JDK – Java Development Kit[1]) oraz zintegrowana platforma programistyczna (np. Eclipse[2]).

2.1. Materiały pomocnicze

Materiały dostępne w Internecie:
http://www.vincehuston.org/dp/
http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

3. Laboratorium:

1. Na platformie github zrób **fork** projektu narzędziowego powp_jobs2d (***Visual2Djobs***) z organizacji podanej przez prowadzącego.
2. Fork projektu należy pobrać lokalnie (np. clone) i zaimportować do Eclipse IDE wybierając *File → Import... → Maven → Existing Maven Projects*. Następnie należy wybrać katalog zawierający plik *pom.xml* jako *Root Directory* i kliknąć *Finish*.
3. Zapoznaj się ze strukturą projektu.
4. Sprawdź czy w „*Maven Dependencies*" są załadowane trzy biblioteki *Jobs2dMagic.jar*, *Drawer.jar* oraz *PowpAppBase.jar.*
5. Zapoznaj się (pobieżnie) z dokumentacją w folderze *doc* dotyczącą obu systemów:
   - Jobs2dMagic – fragment biblioteki obsługującej urządzenia 2D,
   - Drawer – prosta biblioteka do rysowania,
   - PowpAppBase – podstawa do budowy aplikacji okienkowej na potrzeby wizualizacji.
6. **UWAGA: pod koniec zajęć wyniki prac na laboratorium muszą być każdorazowo oznaczane w repozytorium jako osobny *release/tag.* Braki w tym zakresie są równoważne z brakiem obecności na zajęciach.**

   Diagramy i odpowiedzi na pytania *nie* powinny być dodawane do repozytorium tylko mają być przesłane w wiadomości mailowej do prowadzącego zajęcia.

1 http://java.sun.com/javase/downloads/index.jsp
2 http://www.eclipse.org/

3.1. Wprowadzenie

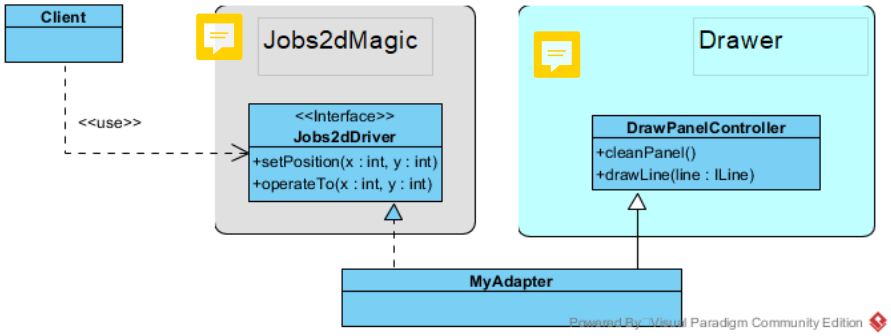
Pat Tern, szef firmy *POWP*, wpadł na pomysł generycznego oprogramowania, które można by było stosować do szerokiej gamy urządzeń wykonujących różnorodne prace za pomocą głowicy na płaszczyźnie 2D. Przykładowo mogą być to urządzenia do:

- cięcia wodą (ang. WaterJet) - https://pl.wikipedia.org/wiki/Waterjet
- cięcia laserowego - https://pl.wikipedia.org/wiki/Cięcie_laserowe
- płaskiego rysowania - https://pl.wikipedia.org/wiki/Ploter_płaski

Ich wspólną cechą jest możliwość wykonywania określonej operacji podczas przesuwania głowicy nad określoną powierzchnią. Można wyobrazić sobie jeszcze wiele zadań, które są realizowane według powyższego schematu. Rozpoczęto już pierwsze prace, ale niestety zespół za nie odpowiedzialny został zatrudniony (a wcześniej prawdopodobnie *podkupiony*) przez konkurencyjną korporację. Pozostało napisane w Javie oprogramowanie ***Jobs2dMagic***, które zawiera bazowe API oraz przykładowe *demonstracyjne wzory.* Niestety, jedynym sposobem na zobaczenie tych tajemniczych wzorów jest wykorzystanie jakiegoś rzeczywistego urządzenia (którego w obecnej fazie prac firma jeszcze nie posiada). Jesteś proszony o poprawienie programu umożliwiającego podgląd demonstracyjnych wzorów. Dla ułatwienia bilblioteka korzysta z gotowej i sprawdzonej biblioteki ***Drawer***. Pakiet *edu.kis.powp.jobs2d.magicpresets* zawiera wyżej wspomniane wzory.

1. Przyjrzyj się działaniu klasy *DrawPanelController* (***Drawer***) w teście *TestDrawer.java*
2. Interfejs *Job2dDriver* reprezentuje podstawową funkcjonalność urządzenia 2D i jest kluczowy w procesie uruchamiania wzorców demonstracyjnych. Przyjrzyj się działaniu testu *Job2dDriverTest.java*.
3. Aplikacja do wizualizacji wzorców znajduje się w klasie *TestJobs2dPatterns.java*. Zawiera ona kod odpowiedzialny za zestawienie ze sobą i użycie podstawowych komponentów znajdujących się w projekcie i bibliotekach.
4. Do bibliotek dostępna jest zewnętrzna dokumentacja a wybrane metody i klasy projektu posiadają odpowiednie komentarze.
   W szczególności należy zapoznać się z dokumentacją do następujących klas i metod:
   - *edu.kis.powp.appbase.Application //* **(w *PowpAppBase*)**
     - *addTest*(String *name*, ActionListener *listener*)
   - *edu.kis.powp.jobs2d.features.DriverFeature*
     - *addDriver*(String *name*, Job2dDriver *driver*)
     - *getDriverManager*()
   - *edu.kis.powp.jobs2d.drivers.DriverManager*
     - *getCurrentDriver*()
     - *setCurrentDriver*(Job2dDriver *driver*)
5. Na koniec zapoznaj się z rolą klasy *edu.kis.powp.jobs2d.features.DrawerFeature.java* w projekcie.

3.2. *Niedopasowanie impedancji*



Ilustracja 1: Diagram UML adaptera między klasami Job2dDriver i DrawPanelController

Stażysta (który już zakończył przygodę z naszą firmą) zaimplementował własną klasę *MyAdapter* (projekt przedstawiony na diagramie na ilustracji 1) rzekomo rozwiązującą w pewnym zakresie opisany wcześniej problem. W jej działaniu wykryto trzy istotne błędy:

- symulacja miała być wyświetlana w oknie aplikacji (a nie w dodatkowym oknie),
- wzorem z testu „Figure Joe 1" na pewno nie miała być parasolka,
- nieprecyzyjna nazwa klasy.

1. Twoim zadaniem jest tak naprawić adapter, aby umożliwiał klientowi korzystanie z funkcjonalności klasy *DrawPanelController* przy pomocy interfejsu *Job2dDriver* zgodnie z wszystkimi wcześniej wymienionymi założeniami. Dostęp do instancji *DrawPanelController* zintegrowanej z oknem aplikacji jest zdefiniowany w metodzie *getDrawerController*() w klasie *DrawerFeature*.
2. Przetestuj poprawki na przykładowym wzorcu z klasy *FiguresJoe (figureScript1)*.
3. Dla Pata Terna dodaj test (metoda *addTest* klasy Application) aby można pokazać drugi przykładowy wzorzec z klasy *FiguresJoe (figureScript2)*.
4. W języku UML naszkicuj diagram klas zaproponowanego przez Ciebie rozwiązania.
5. ***Pytanie*: Kiedy warto (lub trzeba) korzystać z **adaptera klasy** (wariant z projektu stażysty)?

3.3. Adaptery, c.d.

Część urządzeń 2D może pracować w dwóch (lub kilku) trybach, których działanie można porównać do rysowania linia ciągłej i linia przerywanej. Twoim kolejnym zadaniem jest napisanie drugiego sterownika wykorzystującego różne obiekty pochodzące z *LineFactory*. Na lanch-u kolega z pracy powiedział Ci, że ***Drawer*** między innymi udostępnia specjalny typ odcinka *SpecialLine*. Według kolegi Pat Tern byłby na pewno zadowolony, gdyby zobaczył wzory demonstracyjne narysowane tym typem linii.

1. Zaimplementuj własny *LineDrawerAdapter*, który będzie umożliwiał klientowi korzystanie z funkcjonalności biblioteki ***Drawer*** przy pomocy interfejsu *Job2dDriver* oraz wybranego rodzaju linii. Konstrukcja takiego sterownika jest analogiczna do **adaptera** opracowanego w ramach poprzedniego zadania.
2. Do aplikacji z GUI (*TestJobs2dPatterns.java*) dodaj możliwość korzystania w testach z zaimplementowanego **adaptera** (należy wykorzystać metodę *addDriver*) tak aby było można przetestować symulacje rysowania linii używając różnych „trybów".
3. *Wręcz narzuca się, żeby do aplikacji dodać jakiś sposób na dobór parametrów linii. Niestety, domyślne implementacje linii tego nie zapewniają. Możesz coś z tym zrobić?

3.4.* Adapter do adaptera?

Nie zapomnij, że Pat Tern chce zobaczyć jeszcze demonstracyjne wzory z klasy *FiguresJane*. Niestety skrypty generujące wzory nie wykorzystują bezpośrednio interfejsu *Job2dDriver*

1. Zaprojektuj i zaimplementuj rozwiązanie oparte na klasie, która będzie dziedziczyła z klasy abstrakcyjnej *AbstractDriver*, ale do symulacji będzie wykorzystywało bieżący sterownik.
2. Dodaj nowy test wyświetlający wzorce demonstracyjne z klasy *FiguryJane* i przetestuj korzystając z zaimplementowanego w poprzednim punkcie **adaptera**.
3. ***Pytanie*:** Jak powinien wyglądać diagram UML takiego **adaptera?** Czy jest on bliższy wariantowi **adaptera klasy** czy **adaptera obiektu**? Z jakim wzorcem projektowym naprawdę mamy do czynienia?