

Temat : *Strukturalny wzorzec projektowy Adapter (adapter)*

Historia zmian

<i>Data</i>	<i>Wersja</i>	<i>Autor</i>	<i>Opis zmian</i>
22.02.2010	0.1	Tomasz Kowalski	Utworzenie dokumentu i edycja wprowadzenia
...
20.3.2012	2.2	Tomasz Kowalski	Aktualizacja dotycząca pracy z svn (problemy z <i>bin</i>)
20.3.2012	2.3	Tomasz Kowalski	Aktualizacja dotycząca pracy z svn (praca z <i>trunk</i>)
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

1. Cel laboratorium

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

Czas realizacji laboratoriów wynosi 2 godziny.

2. Zasoby

2.1. Wymagane oprogramowanie

Polecenia laboratorium będą dotyczyły programowania wzorców w języku Java. Potrzebne będzie środowisko dla programistów (JDK – Java Development Kit¹) oraz zintegrowana platforma programistyczna (np. Eclipse²).

2.1. Materiały pomocnicze

Materiały dostępne w Internecie:

<http://www.vincehuston.org/dp/>

[http://en.wikipedia.org/wiki/Design_pattern_\(computer_science\)](http://en.wikipedia.org/wiki/Design_pattern_(computer_science))

3. Laboratorium:

1. Na platformie github zrób **fork** projektu narzędziowego powp_plotter_project (*PlotterMagic*) 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 *PlotterMagic.jar*, *Drawer.jar* oraz *PlotSoftBase.jar*.
5. Zapoznaj się (pobieżnie) z dokumentacją w folderze *doc* dotyczącą obu systemów:
 - *PlotterMagic* – fragment biblioteki obsługującej ploter,
 - *Drawer* – prosta biblioteka do rysowania,
 - *PlotSoftBase* – podstawa do budowy aplikacji okienkowej dla plotera.
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>

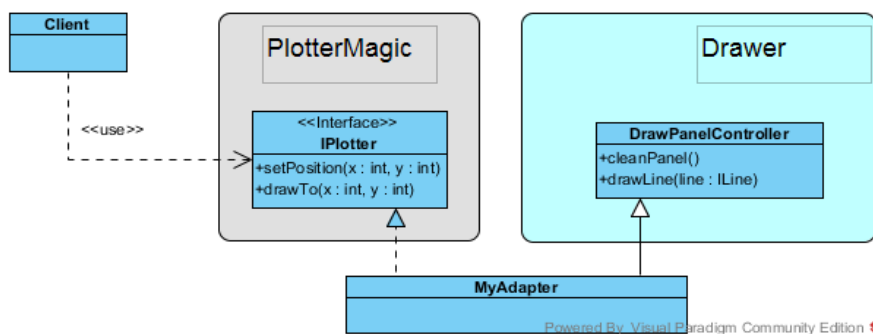
2 <http://www.eclipse.org/>

3.1. Wprowadzenie

Nasz klient Pat Tern, szef firmy **TęczaPrints**, zakupił nowy ploter wraz z napisanym w Javie oprogramowaniem **PlotterMagic**, które zawiera demonstracyjne wzory. Niestety jedynym sposobem na zobaczenie tych tajemniczych wzorów jest uruchomienie plotera (co jest oczywiście BARDZO drogie). Jesteś proszony o poprawienie programu umożliwiającego podgląd demonstracyjnych wzorów bez konieczności użycia plotera. Dla ułatwienia masz wykorzystać gotową i sprawdzoną bibliotekę **Drawer**. Pakiet *edu.iis.client.plotermagic.preset* zawiera wzory prezentujące możliwości plotera:

1. Przyjrzyj się działaniu klasy *DrawPanelController* (**Drawer**) w teście *TestDrawer.java*
2. Interfejs *IPlotter* reprezentuje podstawową funkcjonalność plotera i jest kluczowy w procesie uruchamiania wzorców demonstracyjnych. Przyjrzyj się działaniu testu *IPlotterTest.java*.
3. Komponenty aplikacji do plotera testowane są w *TestPlotSoftPatterns.java*. Dokumentacja dostępna jest folderze *doc* i komentarzach wybranych metod projektu. W szczególności należy zapoznać się z dokumentacją do następujących klas i metod biblioteki *PlotSoftBase*:
 - *edu.iis.powp.app.Application*
 - *addDriver(String name, IPlotter plotter)*
 - *addTest(String name, ActionListener listener)*
 - *getDriverManager()*
 - *edu.iis.powp.app.DriverManager*
 - *getCurrentPlotter()*
 - *setCurrentPlotter(IPlotter plotter)*
4. Na koniec zapoznaj się z rolą klasy *edu.iis.powp.features.DrawerFeature.java* w projekcie.


3.2. Dwa warianty adapterów (klasy i obiektu)



Ilustracja 1: Diagram UML adaptera między klasami *IPlotter* i *DrawPanelController*

Stażysta (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 *IPlotter* zgodnie z wszystkimi wcześniej wymienionymi założeniami. Dostęp do instancji *DrawPanelController* zintegrowanej z oknem aplikacji jest zdefiniowany w metodzie  *getDrawPanelController()* w klasie *ApplicationWithDrawer*.
2. Przetestuj na przykładowym wzorcu z klasy *FiguresJoe* (*figureScript1*).
3. Dla Pata Terna dodaj test (metoda *addTest*) aby można pokazać drugi przykładowy wzorec 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.

Oprócz sterownika opartego na *IPlotter*, która umożliwia operowanie na rzeczywistym ploterze, dysponujesz zaimplementowanym przez Ciebie adapterem do obiektu *DrawPanelController*. Twoim zadaniem (na razie) jest umożliwienie testowania oprogramowania korzystającego z implementacji *drivera* wykorzystującego system **Drawer**. Ploter może pracować w dwóch trybach (linia ciągła i linia przerywana), dlatego musisz napisać drugi sterownik wykorzystujący różne obiekty pochodzące z *LineFactory*. Na lunch-u kolega z pracy powiedział Ci, że **Drawer** udostępnia specjalny typ odcinka *SpecialLine* (oraz inne). Według kolegi Pat Tern byłby na pewno zadowolony, gdyby zobaczył wzory demonstracyjne narysowane tym typem linii.

1. Zaimplementuj własny *LinePlotterAdapter*, która będzie umożliwiała klientowi korzystanie z funkcjonalności biblioteki **Drawer** przy pomocy interfejsu *IPlotter* oraz wybranego rodzaju linii. Konstrukcja takiego sterownika jest analogiczna do **adaptera** opracowanego w ramach poprzedniego zadania.
2. Do aplikacji z GUI (*TestPlotSoftPatterns.java*) dodaj możliwość korzystania w testach z zaimplementowanego **adaptera** (należy wykorzystać metodę *addDriver*) tak aby było można przetestować symulacje rysowania linii używając różnych trybów plotera.
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 *IPlotter*

1. Zaprojektuj i zaimplementuj rozwiązanie oparte na klasie, która będzie dziedziczyła z klasy abstrakcyjnej *AbstractPlotter*, 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?