Splash!

Projekt: Rastrowy edytor graficzny

Wybrane z dokumentacji projektu

Spis treści:

1. Opis projektu

Splash! To wieloplatformowy, rastrowy edytor graficzny wzorowany na programach takich jak Photoshop, Gimp czy Microsoft Paint. Ideą przyświecającą aplikacji było wypełnienie luki pomiędzy prostym i intuicyjnym Paintem a zaawansowanym i topornym Gimpem. Aplikacja została napisana w języku Java przy użyciu biblioteki Swing. Głownym celem projektu była nauka i praktyczne wykorzystanie wielu wzorców projektowych.

Projektowane funkcjonalności:

a) Narzędzia rysowania:

- ołówek
- pędzel
- linia
- linia łamana
- wypełnienie
- zmiana koloru pierwszego planu
- zmiana koloru tła
- prostokąt
- trójkąt
- owal
- spray
- zaznaczenie i wycinanie (jak w paincie)
- gumka
- przesunięcie (warstw)
- pipeta
- tekst
- zoom
- zmiana rozmiaru rysowania

b) Warstwy:

- tworzenie
- przesuwanie pomiędzy warstwami
- scalanie w dół
- usuwanie
- zmiana widoczności
- zmiana nazwy

c) Filtry dla całego arkusza oraz dla pojedynczych warstw:

- inwersja kolorów
- obrót (o dowolny kąt)
- zmiana jasności (o dany procent)
- zmiana kontrastu (o dany procent)
- rozmazanie
- wyostrzenie
- bilans bieli

d) I/O:

- otwieranie oraz zapisywanie obrazów do plików o rozszerzeniach:
- jpg
- png
- bmp
- gif
- .rozszerzenie_aplikacji (zapisanie i odczytanie stanu obecnego arkusza wraz z warstwami).

2. Wykorzystane wzorce projektowe

1. MVC
2. Observer
3. Strategy
4. Memento
5. Chain of Responsibility
6. Template method
7. Factory (Simple Factory)
8. Prototype
9. Singleton
10. Composite
11. Adapter + Iterator

LEGENDA:
Wzorce kreacyjne
Wzorce strukturalne
Wzorce czynnościowe
Wzorce architekturalne

3. Opis wykorzystania oraz diagramy UML

1. MVC (Model View Controller)

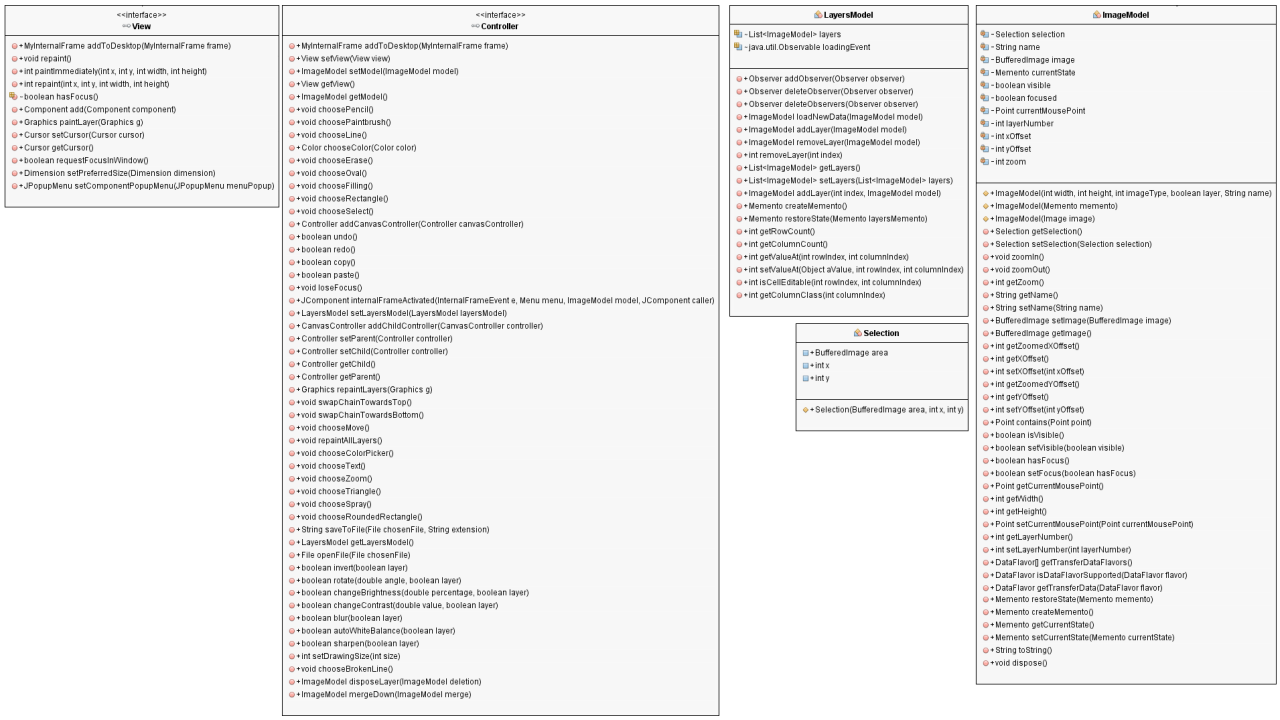
a) Model

I. ImageModel – reprezentuje dane pojedynczej warstwy za pomocą BufferedImage oraz innych pomniejszych elementów.

II. LayersModel – reprezentuje kilka warstw (ImageModel).

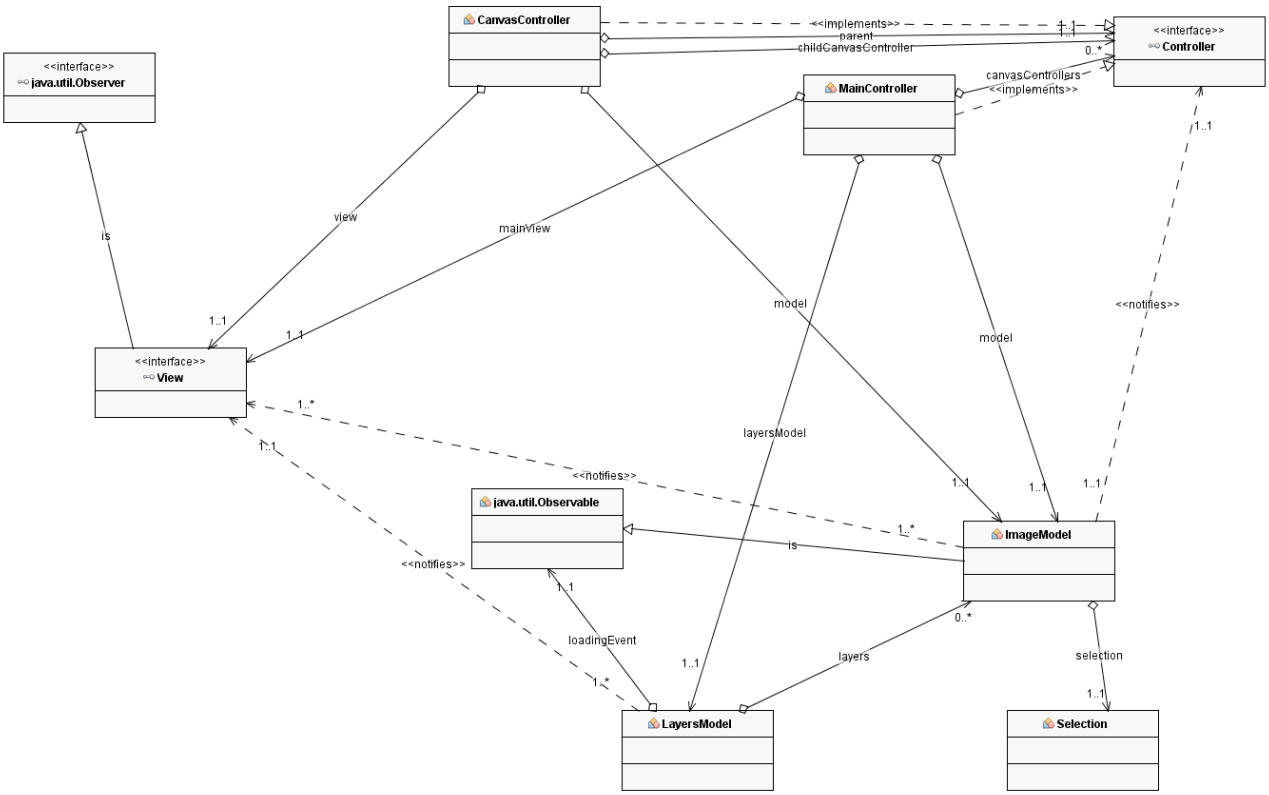
b) View – w skład tej warstwy wchodzą widoki dziedziczące z klas biblioteki Swing i implementujące interfejs View. Mamy tu między innymi: MainView (widok główny), InfoPanel (wyświetla obecną pozycję kursora), LayersPanel (wyświetla warstwy obecnego arkusza), Menu, ToolPanel, itp.

c) Controller – w skład tej warstwy wchodzą klasy implementujące interfejs Controller, który to z kolei dziedziczy z typowych interfejsów Swingowych takich jak MouseMotionListener czy MouseListener. Obiekty te zazwyczaj są dodawane jako słuchacze w widokach. Wyróżnione zostały dwie klasy kontrolerów: MainController (kontroler główny) oraz CanvasController (kontroler odpowiedzialny za obsługę pojedyńczej warstwy).



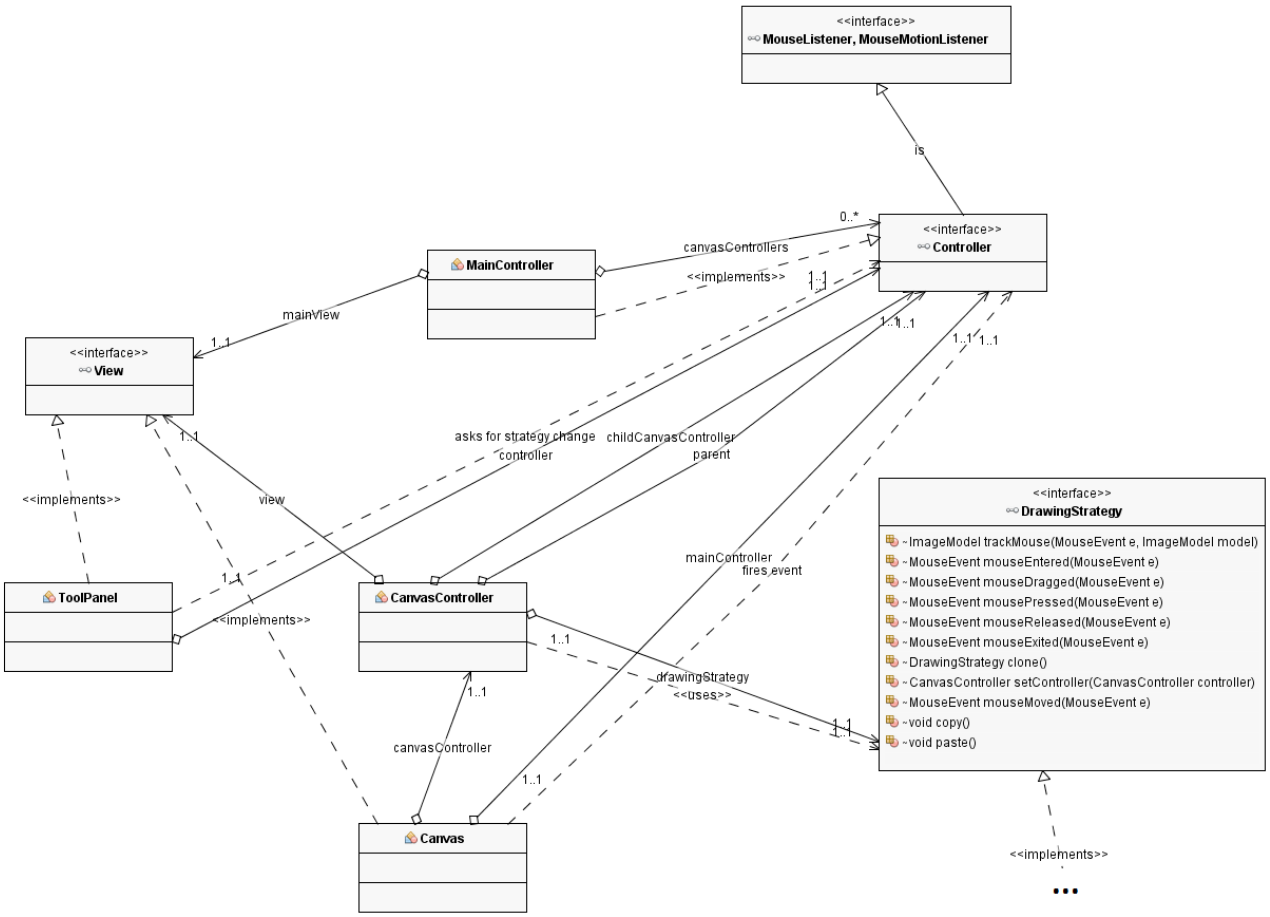
Ilustracja 1: Wzorzec – MVC. Interfejsy oraz klasy warstw MVC

2. Observer – obiekt obserwowany (model) informuje widoki o zmianach.



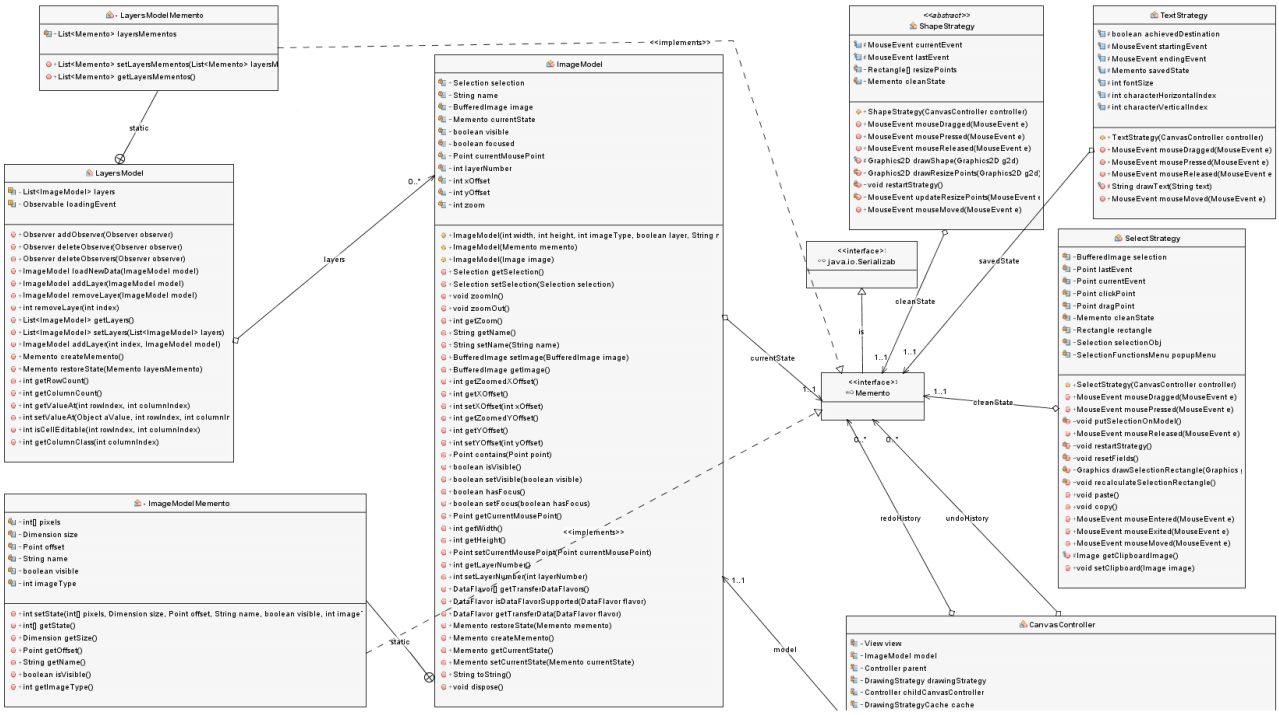
Ilustracja 2: Wzorzec – Observer. Komunikacja między warstwami MVC przy wykorzystaniu wzorca Obserwator

3. Strategy – wzorzec ten w projekcie realizuje różne formy algorytmu rysowania wybierane z panelu narzędzi takie jak np.: ołówek, pędzel, rysowanie figur, zaznaczanie, itp.



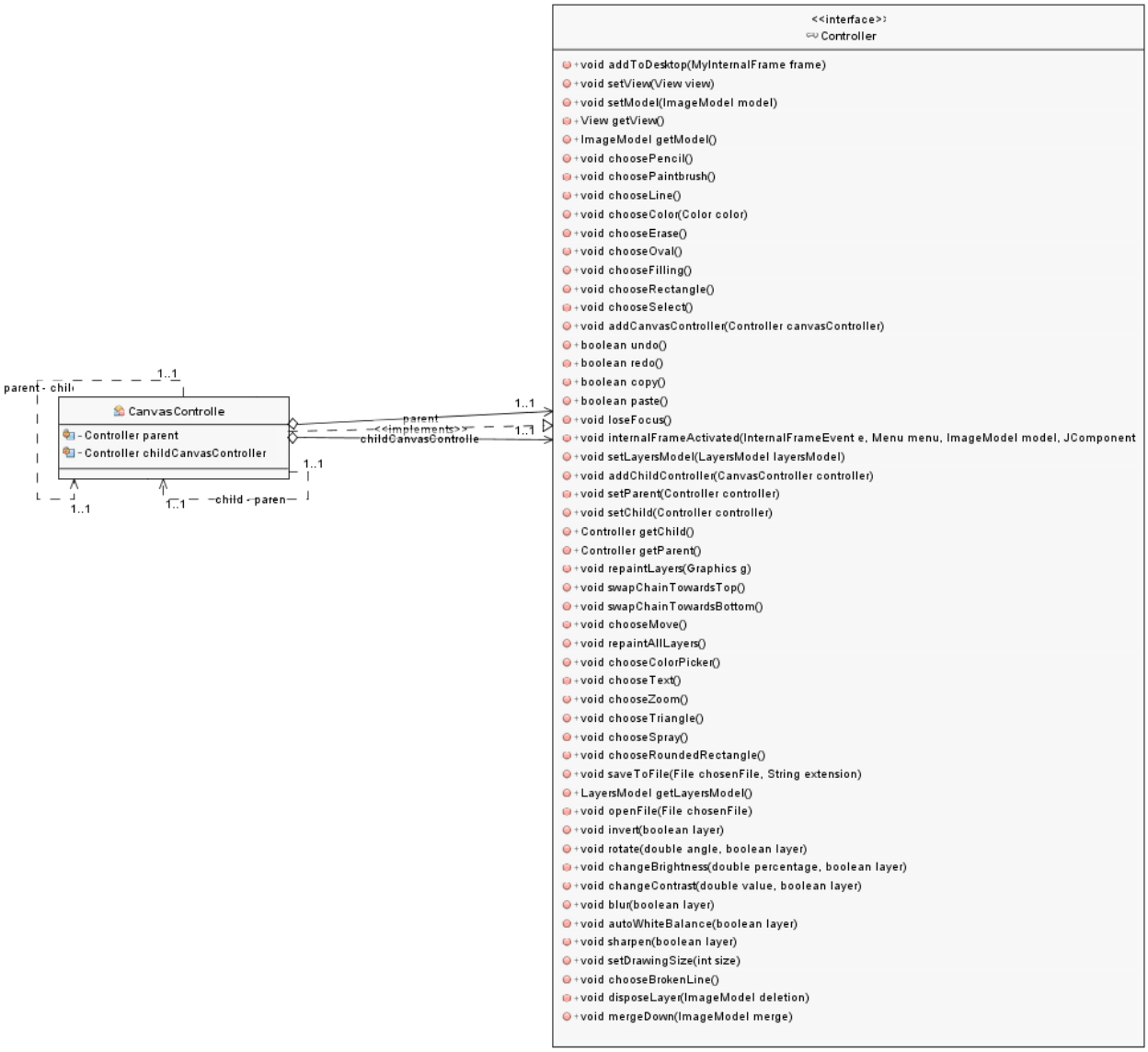
Ilustracja 3: Wzorzec – Strategy. ToolPanel informuje o zmianie strategii przez użytkownika aplikacji, Canvas wysyła natomiast zdarzenia na podstawie których kontekst (CanvasController) korzysta ze strategii. Hierarchia klas Strategii patrz: wzorzec Prototyp

4. Memento – zajmuje się zapamiętywaniem stanu modelu i udostępnia go na zewnątrz bez naruszania zasad hermetyzacji co pozwala na zrealizowanie mechanizmu "undo/redo" oraz zapisanie stanu aplikacji.



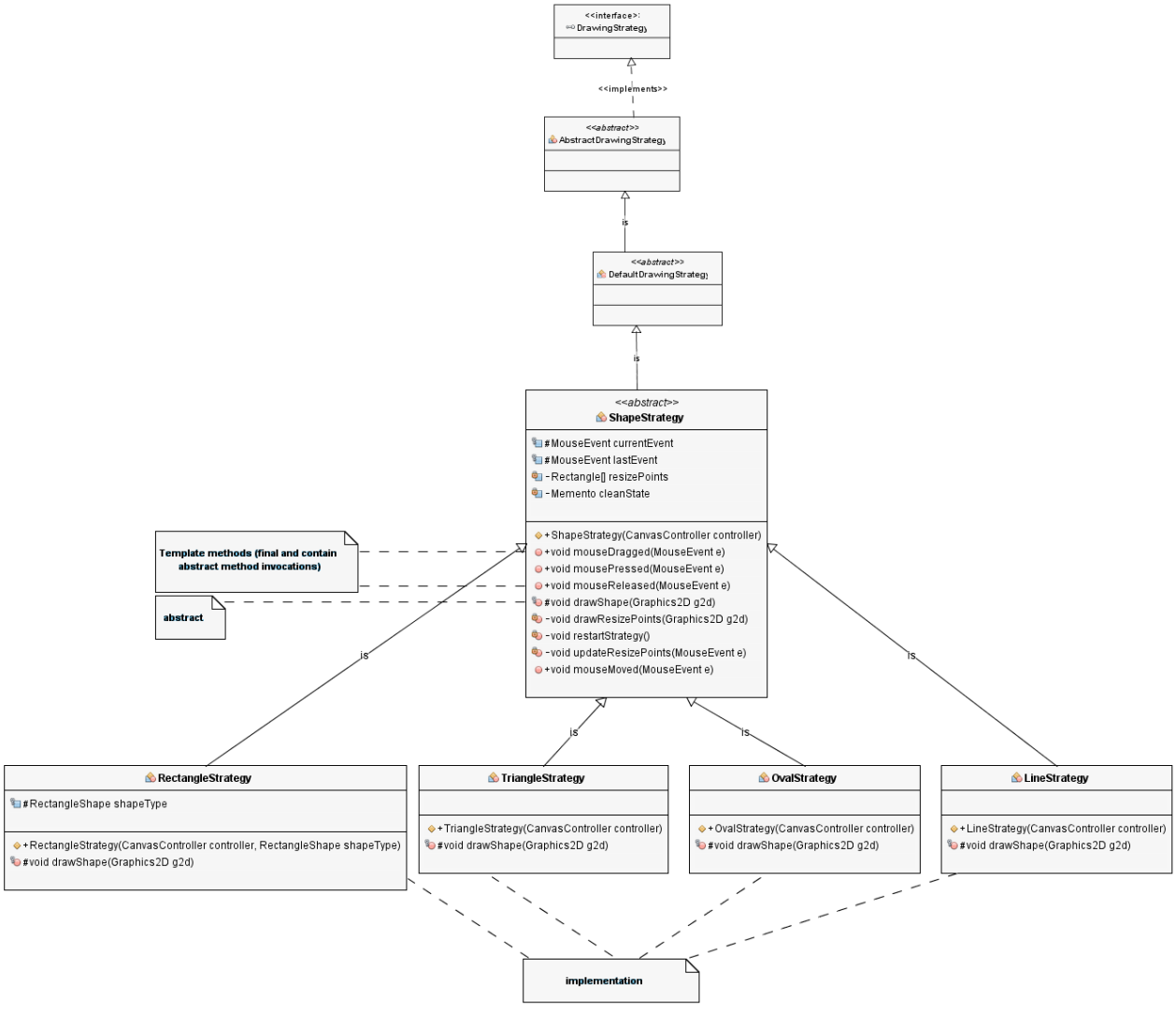
Ilustracja 4: Wzorzec Memento. Caretaker - Controller; Originator - LayersModel, ImageModel; Memento – interfejs wąski; Metody prywatnych, wewnętrznych klas statycznych jako interfejs szeroki dla Originatora. Klasy statyczne nie są związane łańcuchem statycznym podczas serializacji

5. Chain of responsibility – polecenia cofnięcia i ponowienia oraz kopiowania i wklejania podążają drogą od głównego kontrolera, poprzez kontrolery, dla których widoki mają fokus, skończywszy na kontrolerze, dla którego model jest aktywny, gdzie wykonywana jest akcja.



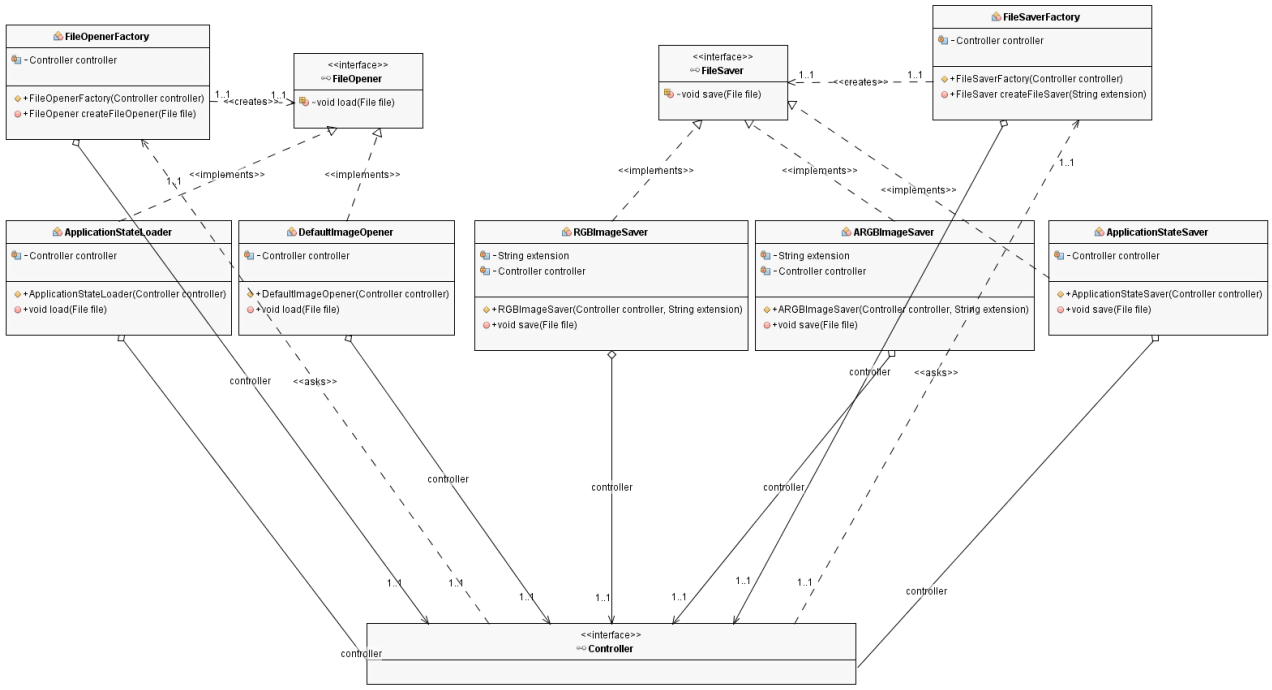
Ilustracja 5: Wzorzec Chain of responsibility. Każdy obiekt posiada referencję do rodzica i dziecka. Większość metod wywoływana jest w łańcuchu i realizowana w przypadku gdy model ma fokus

6. Template method – schemat rysowania figur jest dosyć podobny, składają się na niego między innymi rysowanie małych kwadracików na końcach figur (do zmiany rozmiaru poprzez przeciągnięcie) oraz obsługa rysowania, a w szczególności przeciągania w przeciwnym kierunku. W tym algorytmie elementem zmiennym jest rysowanie figury. Korzystając ze wzorca metody szablonowej w klasach pochodnych podana zostaje jedynie implementacja metody rysowania figury, co pozwala na łatwe i szybkie rozszerzenie strategii o kolejne figury.



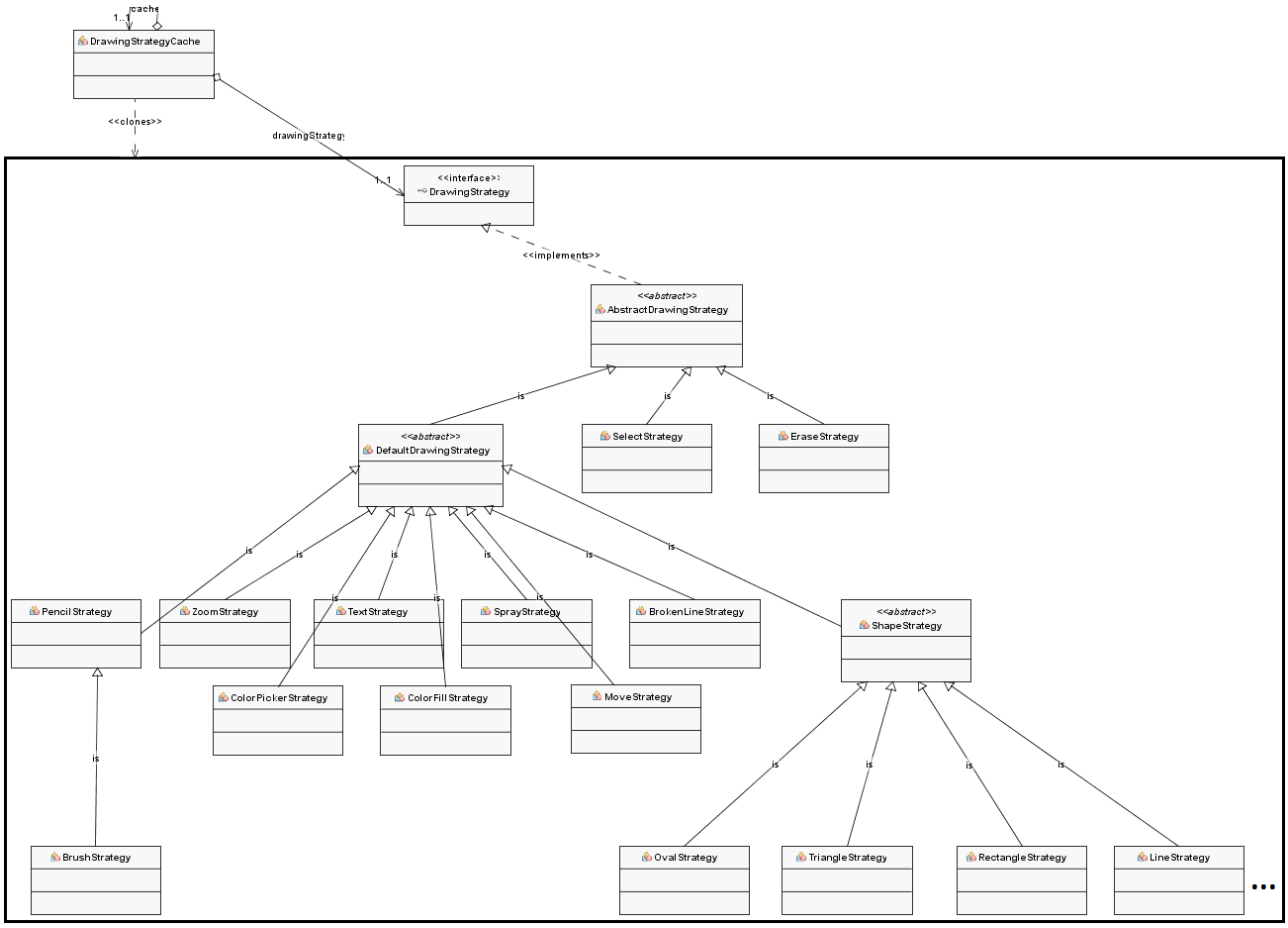
Ilustracja 6: Wzorzec – Template method. Abstrakcyjna metoda drawShape wywoływana jest w (finalnych) metodach szablonowych mouseDragged oraz mouseReleased. Klasy dziedziczące implementują tę metodę

7. Factory (Simple Factory) – przy otwieraniu obrazu z pliku, może być on różnie interpretowany (ARGB – png, gif; RGB – bmp, jpg; plik zawierający stan aplikacji), dlatego też potrzebne będą różne klasy ładujące dane do odpowiedniego formatu oraz odwrotnie – zapisujące do niego. Wykorzystując wzorzec "Fabryka", na podstawie rozszerzenia pliku można przydzielić odpowiednią klasę ładującą/zapisującą.



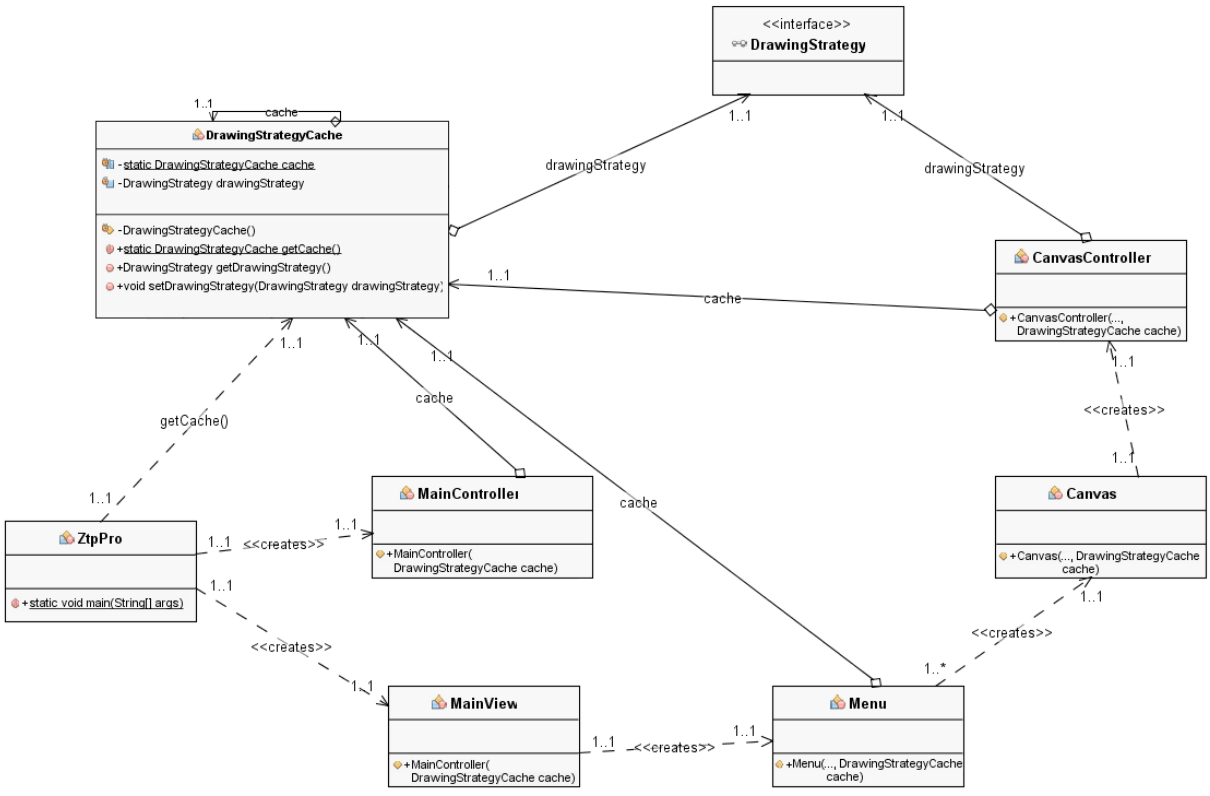
Ilustracja 7: Wzorzec – Factory (Simple Factory). Na zlecenie klas implementujących interfejs Controller, na podstawie pliku lub jego rozszerzenia zwracana jest klasa otwierająca lub zapisująca dany typ pliku (obrazu)

8. Prototype – podczas tworzenia nowego kontrolera powinien on mieć tę samą strategię, aby ta sama opcje rysowania nie różniły się pomiędzy arkuszami i warstwami. Strategia tworzona jest więc na podstawie prototypu przy czym zmieniany jest tylko jej kontroler.



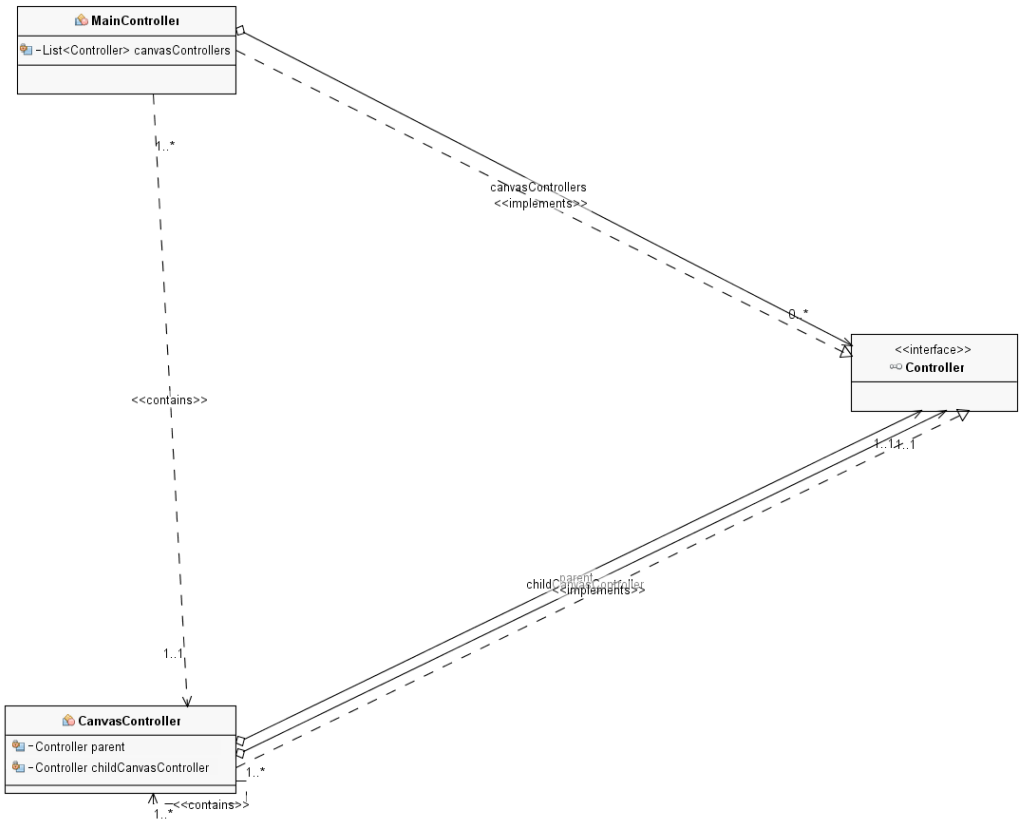
Ilustracja 8: Wzorzec - Prototype. Wybrana obecnie strategia jest klonowana podczas pobierania jej z cache przy pomocy gettera

9. Singleton – zajmuje się przechowywaniem prototypu strategii (jeden typ strategii dla całej aplikacji), tzw. "cache".



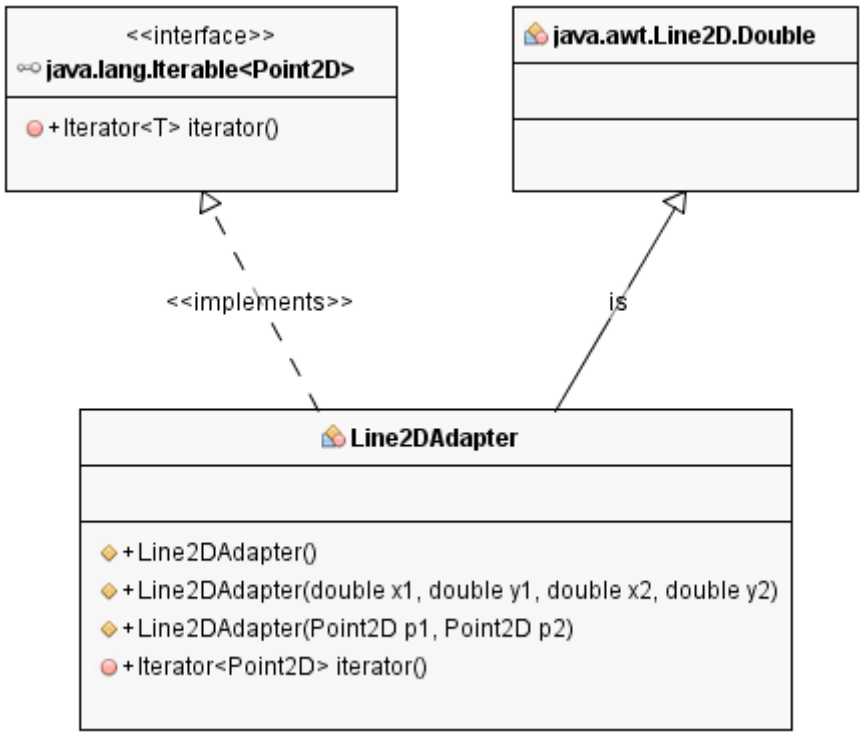
Ilustracja 9: Wzorzec - Singleton. Pozwala na pojedynczą instancję w programie. Globalny dostęp nie jest wykorzystany co pokazuje łańcuch tworzenia obiektów i przekazywania instancji cache

10. Composite – widoczny szczególnie w warstwie kontrolerów. Kontroler główny zawiera kontrolery warstw (po jednym dla każdego z arkuszy), które to kolejno zawierają referencję do kontrolera z wyższej warstwy.



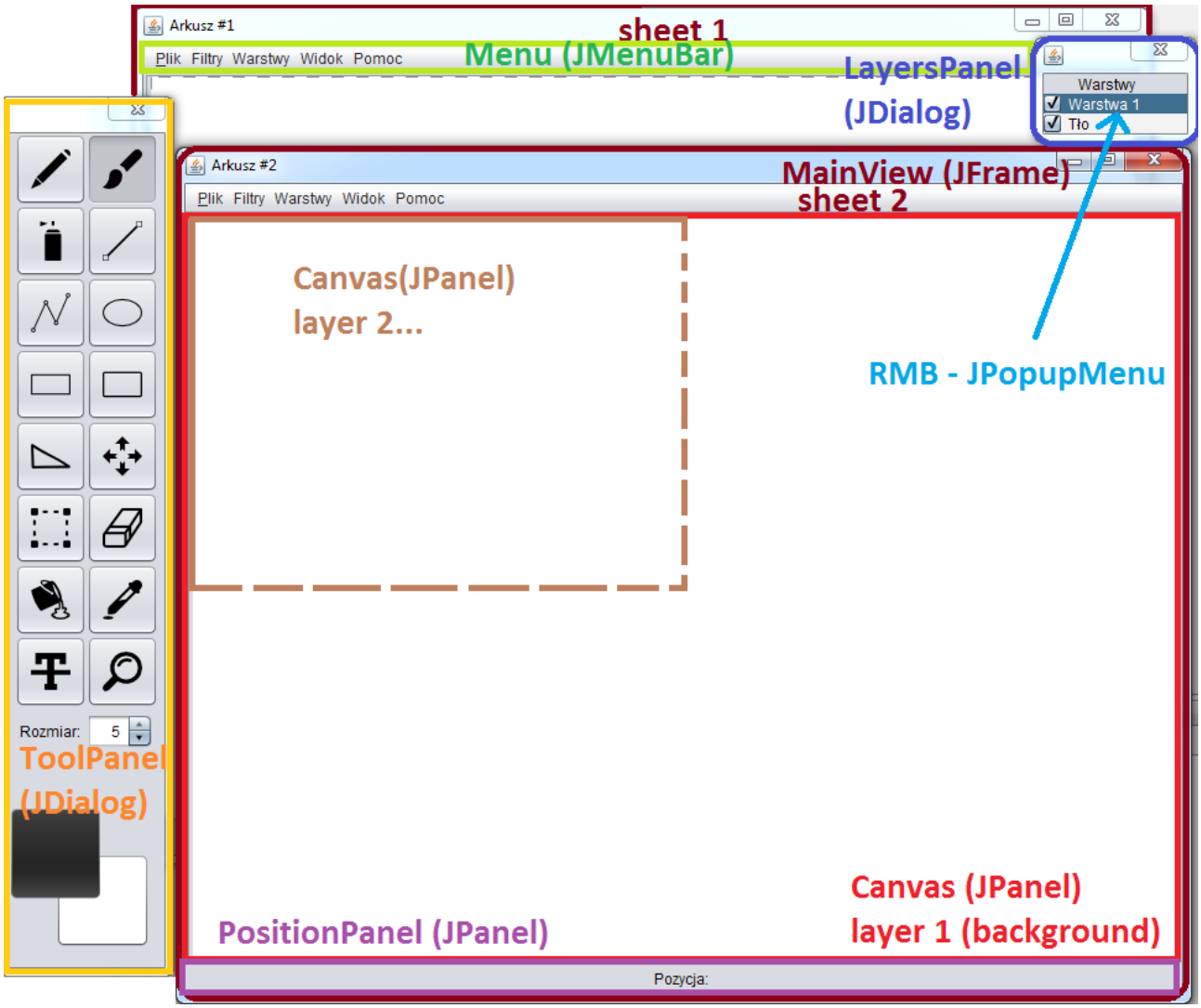
Ilustracja 10: Wzorzec - Composite

11. Adapter + Iterator – podczas rysowania za pomocą pędzla przekazywane informacje na temat trasy myszy mogą nie być ciągł e. W takim przypadku koła rysowane przez pędzel nie będą ze sobą połączone i nie otrzymamy poprawnego efektu. Należy więc wyznaczyć linię pomiędzy dwoma punktami i dla każdego punktu z tej linii narysować koło. Dzięki wykorzystaniu adaptera możliwe jest przystosowanie istniejącej już klasy (Line2D) do nowego interfejsu zawierającego metodę zwracającą stworzony dla linii iterator po punktach.



Ilustracja 11: Wzorzec - Adapter oraz iterator. Pozwala na łatwe uzyskiwanie kolejnych punktów linii przy użyciu pętli "for"

4. Prototyp GUI (Warstwa widokowa)



Ilustracja 12: Prototyp GUI