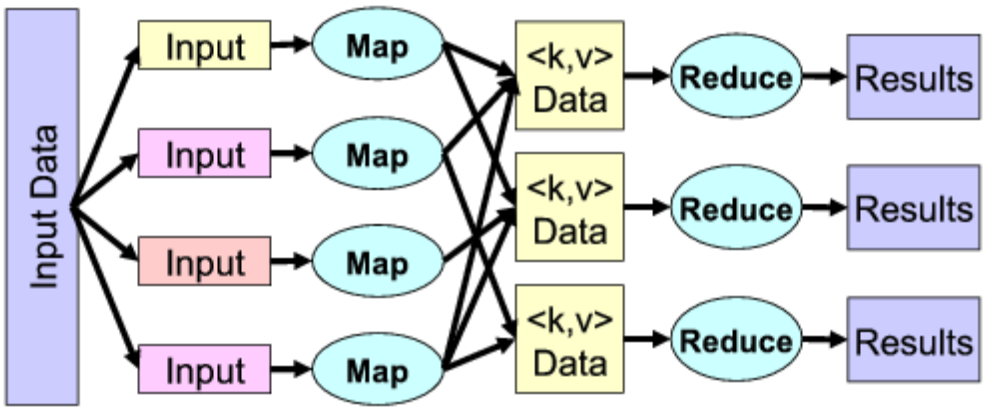
Implementacja frameworku MapReduce z przykładem zastosowania w Analizie Tekstu

1. Opis projektu

MapReduce jest frameworkiem zaproponowanym przez firmę Google do równoległego przetwarzania dużych zbiorów danych. Jego wydajność polega na podziale zbioru danych na części i rozproszeniu problemu na wiele komputerów które przetworzą powstałe podzbiory niezależnie od siebie.

Operacje realizowane są w dwóch krokach:
Krok “map” - główny program pobiera dane z wejścia i dzieli je na mniejsze podproblemy po czym przesyła je do robotników. Każdy robotnik może dokonać kolejnego podziału na podproblemy lub przetworzyć problem i zwrócić odpowiedź.
Krok “reduce” - przetworzone dane z kroku map są wysyłane do programów agregujących, które łączą je w wynik końcowy zwracany do master node’a.

W projekcie poza implementacją frameworku chcemy pokazać jego działanie na prostym przykładzie analizy dużego korpusu tekstu. W efekcie chcielibyśmy, aby dla danego korpusu program zwracał liczbę słów w tekście.



2. Opis implementacji

- Server

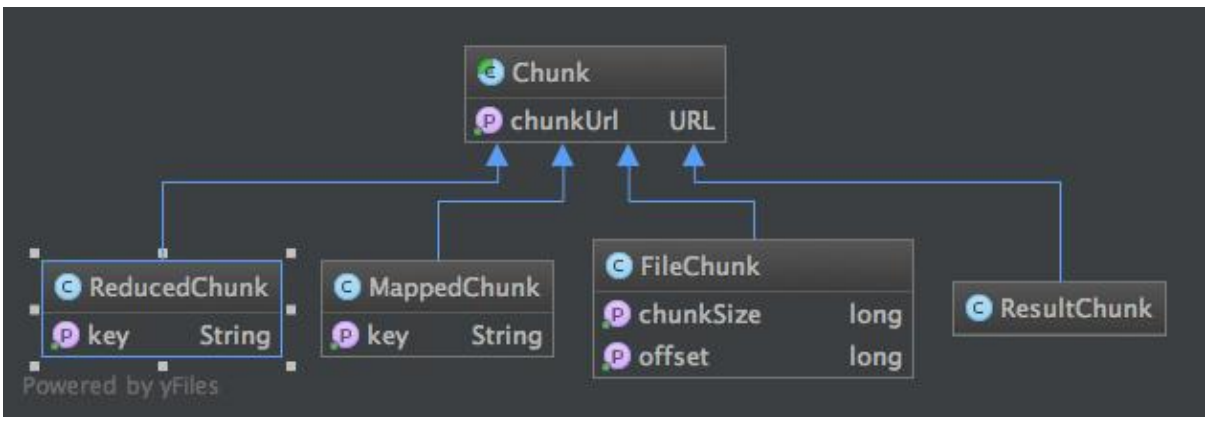
Stanowi główną część programu. Zostaje mu wskazany plik tekstowy, którego analizę należy przeprowadzić. Plik ten dzielony jest na mniejsze fragmenty o zadanym rozmiarze, a te przypisywane są do poszczególnych node'ów (klientów zdalnych). Po stronie servera przeprowadzany jest proces przydzielania zadań oraz jego kontrola (na wypadek niepowodzenia/awarii któregoś z klientów zadanie zostanie przekazane innemu), a także kompletowanie i przekazywanie dalej kolejnych wyników.
Klasy wykorzystywane przez serwer: Splitter, ConnectionHandler, LoadBalancer, Grouper

- Node

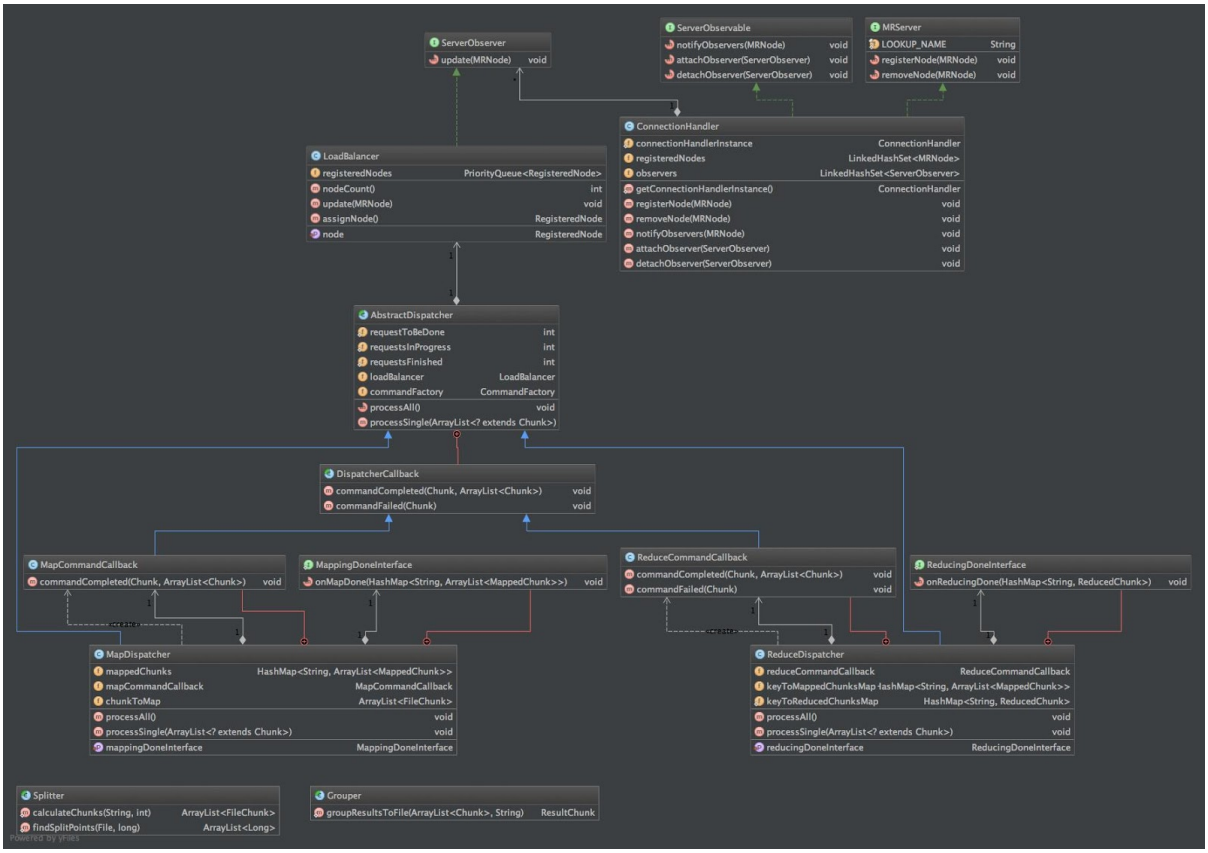
Jest to reprezentacja zdalnej jednostki (klienta). Łączy się z serwerem deklarując gotowość do pracy. W zależności od etapu działania programu, zostaje mu przyznany fragment pliku wejściowego, który ma za zadanie przeanalizować (wówczas pełni funkcję mappera), bądź też otrzymuje poszczególne wyniki pracy mapperów (połączone jednym kluczem, wówczas pełni funkcję reducera), które należy połączyć w jeden wynik dla zadanego klucza.

3. Diagramy klas

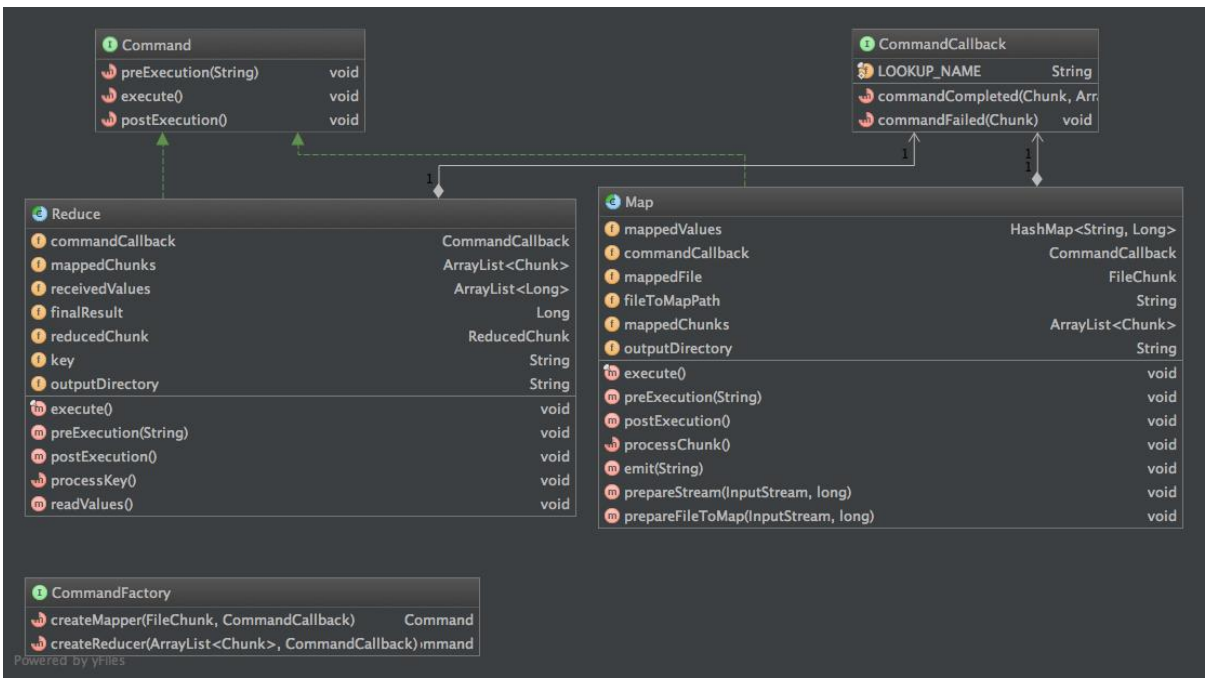
Pakiet Chunk:



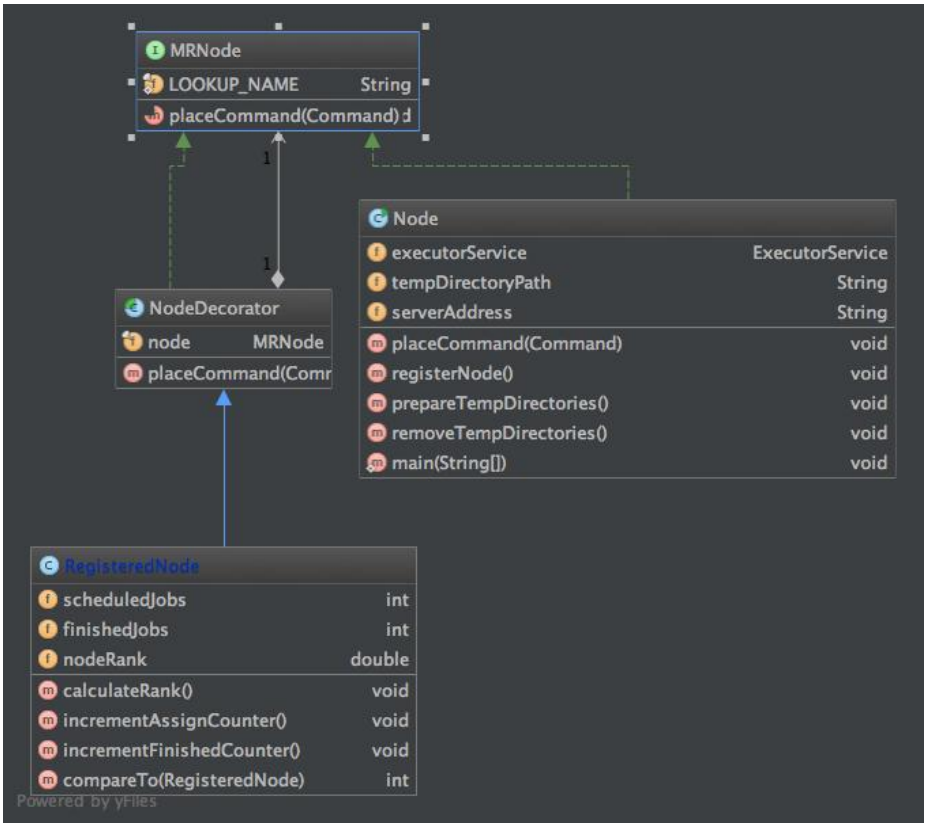
Pakiet Server:



Pakiet Command:



Pakiet Node:


MRNode
LOOKUP_NAME String
placeCommand(Command)
1
1
Node
executorService ExecutorService
tempDirectoryPath String
serverAddress String
placeCommand(Command) void
registerNode() void
prepareTempDirectories() void
removeTempDirectories() void
main(String[]) void
NodeDecorator
node MRNode
placeCommand(Comr
RegisteredNode
scheduledJobs int
finishedJobs int
nodeRank double
calculateRank() void
incrementAssignCounter() void
incrementFinishedCounter() void
compareTo(RegisteredNode) int
Powered by yFiles

3. Działanie

Po wybraniu pliku w Interfejsie Serwera, następuje jego odpalenie poprzez przycisk *Start Server*. Następuję wtedy inicjalizacja *LoadBalancera* oraz połączeń *ConnectionHandler* i nasłuchiwanie na zmiany w połączeniach od strony klientów zdalnych.
Następnie w interfejsie klientów zdalnych, którzy będą przetwarzali części plików, można wybrać ile wątków dana stacja będzie obsługiwała oraz miejsce na przechowywanie plików tymczasowych. Po połączeniu się za pomocą przycisku *Connect* następuje rejestracja klienta zdalnego - *Noda*.
Serwer, który implementuje metodę interfejsu *Obserwatora* na bieżąco informuje o ilości podpiętych klientów zdalnych.
Gdy klienci są podpięci, możliwe jest odpalenie algorytmu. Następuje obliczenie ilości kawałków, na które zostanie podzielony plik wejściowy; inicjalizuje jest *Dispatcher*, który wraz z *LoadBalancerem* przydziela kolejne części do klientów.
Gdy wszystkie pliki zostaną obsłużone, jedna ze stacji roboczych łączy wszystkie ukończone zadania.
GUI jest rozszerzalne, dlatego nie ma problemu z dodaniem funkcjonalności innego algorytmu.
Zaprezentowana implementacja zlicza wyrazy występujące w tekście, jednakże możliwe jest podpięcie innych algorytmów, między innymi wyszukujących wystąpienia słowa w tekście, znajdowanie słów podobnych do zadanego wzorca, czy zliczanie częstości występowania danych znaków. Przy dalszych implementacjach możliwe byłoby np dekodowanie zaszyfrowanych wiadomości, na podstawie częstości występowania danych znaków.

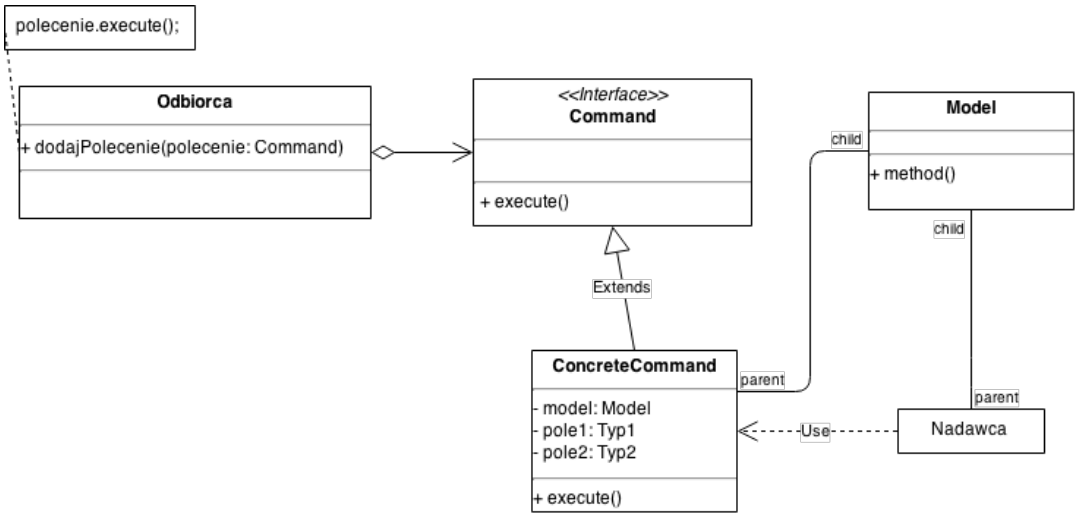
4. Zastosowane wzorce projektowe

a. Command

- **Opis wzorca**

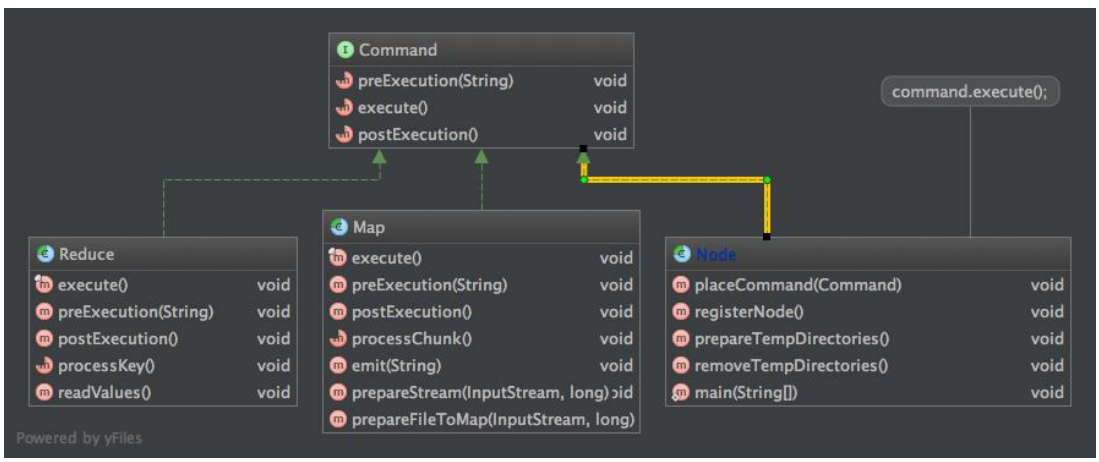
Wzorzec traktujący żądanie wykonania czynności jako obiekt. Umożliwia to użycie parametrów do polecenia w zależności od odbiorcy żądania. Obiektowe traktowanie żądań pozwala także na przechowywanie ich w kontenerach.

- **Struktura**



- **Zastosowanie w projekcie**

Poleceniami są klasy dziedziczące po Map oraz Reduce. Definiują one jaka akcje ma być wykonana na wykonana na poszczególnych Chunkach na danym etapie pracy programu. Tak zdefiniowane polecenia wysyłane są do Node'ów razem z potrzebnymi danymi w celu wykonania.

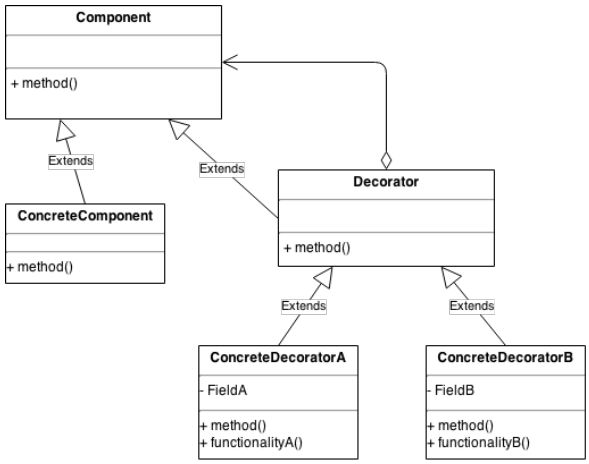


b. **Decorator**

- **Opis wzorca**

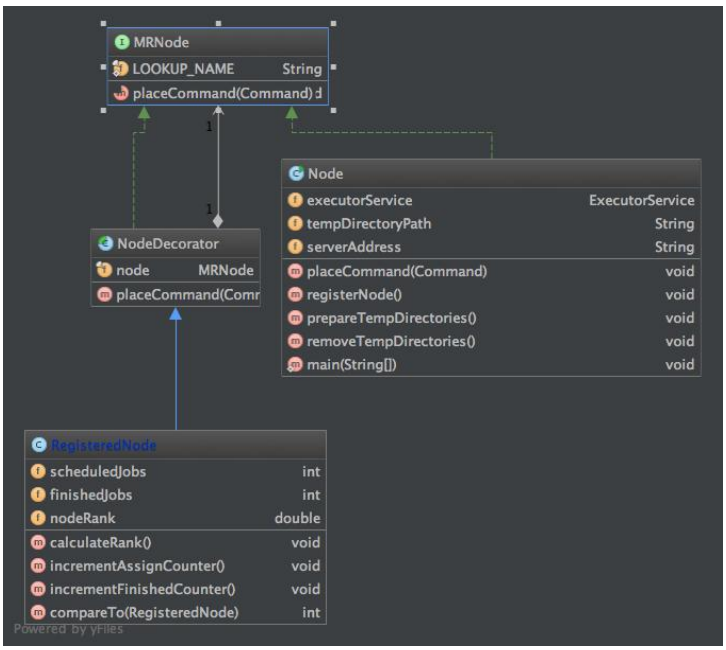
Użycie tego wzorca pozwala na dodanie nowej funkcjonalności do istniejących klas już w trakcie działania programu. Dzieje się to dzięki użyciu dodatkowej klasy (dekoratora), która opakowuje klasę podstawową. Przekazywany jest do niej obiekt klasy podstawowej, której metody wywoływane są przez metody dekoratora, dodatkowo umożliwiając implementację nowej funkcji.

- **Struktura**



- **Zastosowanie w projekcie**

Klasa RegisteredNode jest dekoratorem klasy Node. Dodatkowa użyteczność dekoratora wykorzystywana jest przez klasę LoadBalancer przy wyborze najlepszego Node'a któremu można zlecić kolejne polecenie.

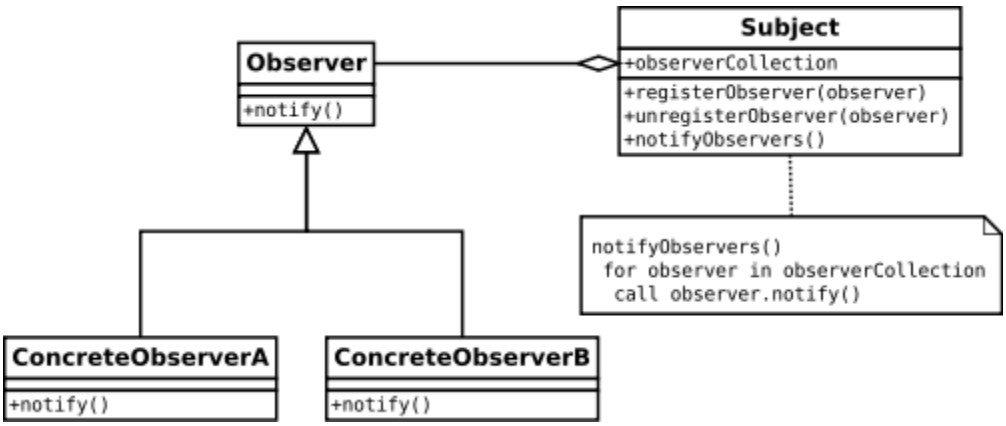


c. **Observer**

- **Opis wzorca**

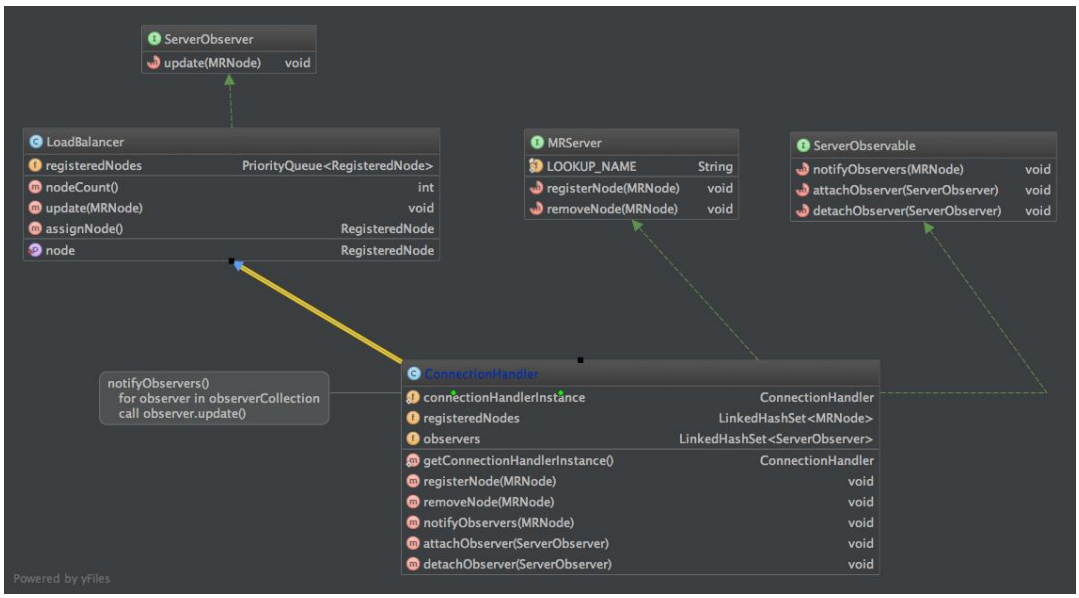
Wzorzec ten służy do implementacji mechanizmu powiadamiania pewnego obiektu (obiektów) o zmianach stanu innego obiektu (obiektów). Przydatny w sytuacji, gdy jakiś obiekt jest zależny od stanu innego, dlatego informacja o jego zmianie jest bardzo istotna.

- **Struktura**



- **Zastosowanie w projekcie**

Klasa ConnectionHandler jest obserwowana przez klasę LoadBalancer. W przypadku nowego połączenią do serwera, ConnectionHandler informuje LoadBalancera o nowym nodzie, któremu można zlecać zadania. Zaimplementowano model push w którym obiekt obserwowany przesyła w powiadomieniu dodatkowe informacje do klas obserwujących.

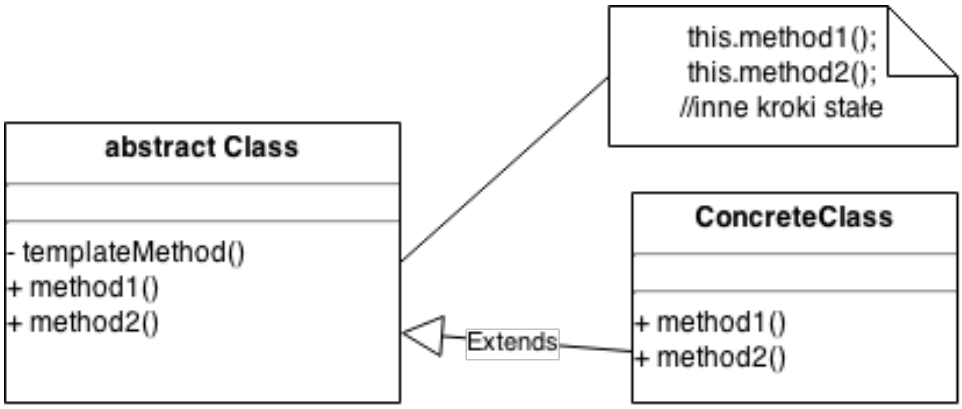


d. **Template Method**

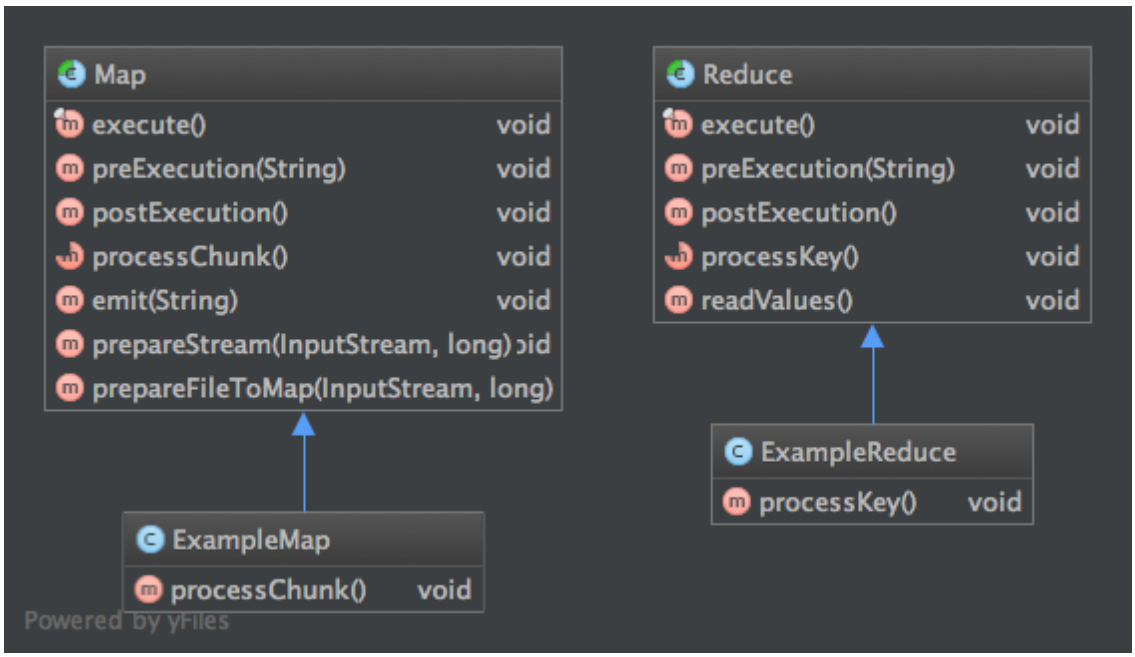
- **Opis wzorca**

Użycie tego wzorca polega na zdefiniowaniu metody klasy, która jest szkieletem algorytmu. Metoda szablonowa zawiera niezmienne kroki algorytmu i nie może zostać nadpisana w klasach pochodnych. W niej używane są metody reprezentujące kroki zmienne. Takie podejście pozwala na zdefiniowanie klasy pochodnej, która nadpisuje metody opisujące zmienne kroki algorytmu i dostosowuje jego działanie do aktualnych potrzeb.

- **Struktura**



- **Zastosowanie w projekcie**

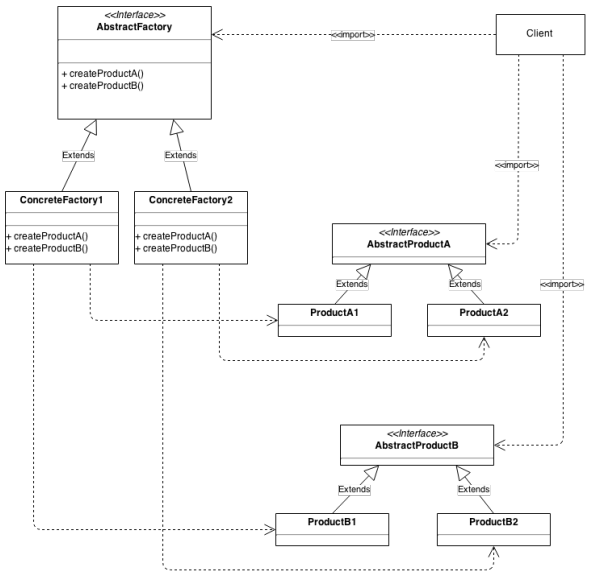


e. **Abstract Factory**

- **Opis wzorca**

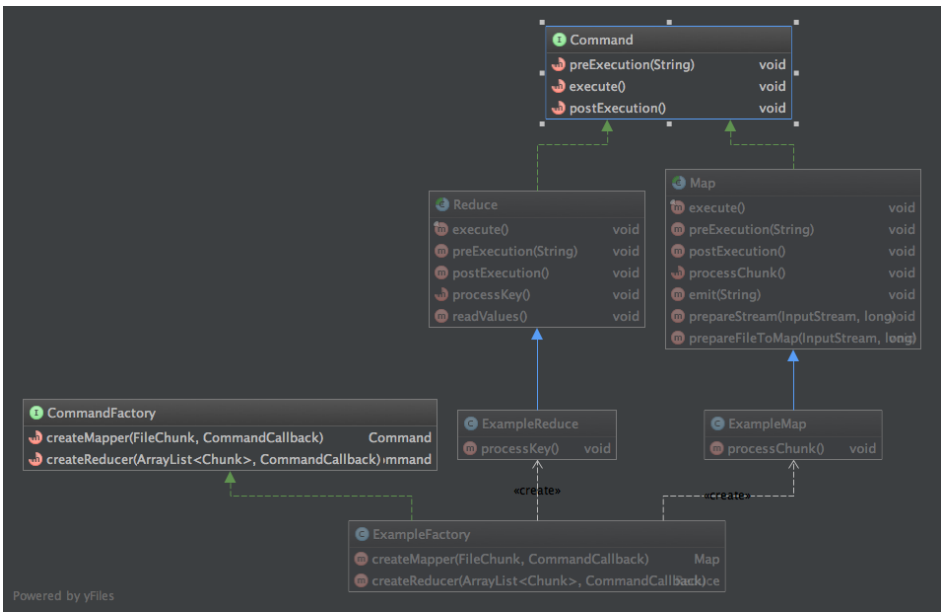
Wzorzec ten dostarcza interfejs do tworzenia różnych obiektów należących do jednej „rodziny”, bez określania klas konkretnych. Powstaje w ten sposób jeden obiekt posiadający możliwość tworzenia podobiektów, natomiast ich typ określany jest w trakcie działania programu.

- **Struktura**



- **Zastosowanie w projekcie**

Wzorzec fabryki został zastosowany przy tworzeniu klas typu Command (polecenie). Umożliwia łatwe dostosowanie frameworku do konkretnego zadania.

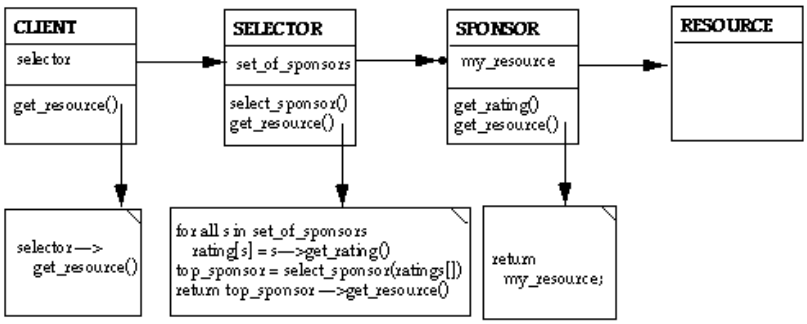


f. **Sponsor-Selector**

- **Opis wzorca**

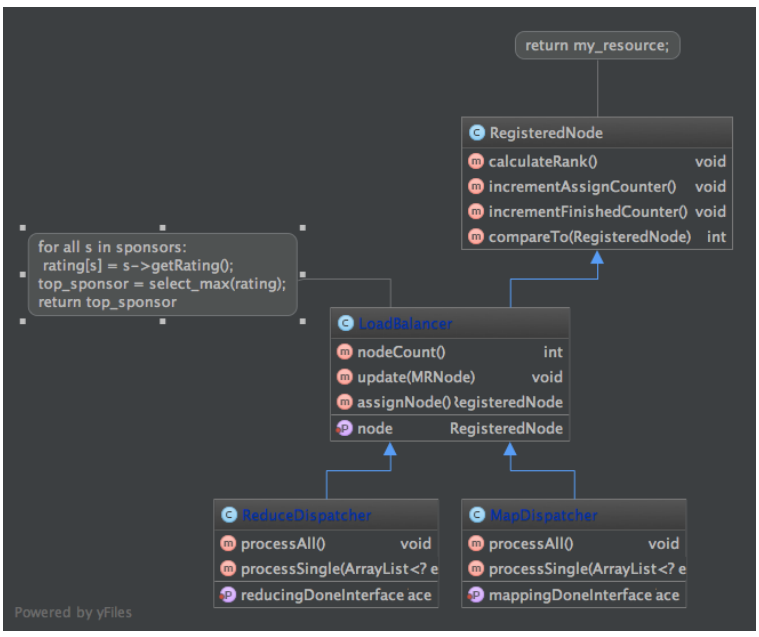
Wzorzec Sponsor-Selector stosowany jest w celu zapewnienia mechanizmu do wybierania najlepszego zasobu dla zadania, z zestawu środków, które zmieniają się dynamicznie. Pozwala oprogramowaniu na integrację nowych zasobów oraz pozyskaniu wiedzy o tych zasobach, w czasie wykonywania, w sposób przejrzysty dla użytkowników tych zasobów. Wzorzec ten jest oparty na idei odseparowania trzech rodzajów obowiązków: wiedzy kiedy zasób jest przydatny; wybieraniu pomiędzy zasobami oraz z korzystania z zasobów

- **Struktura**



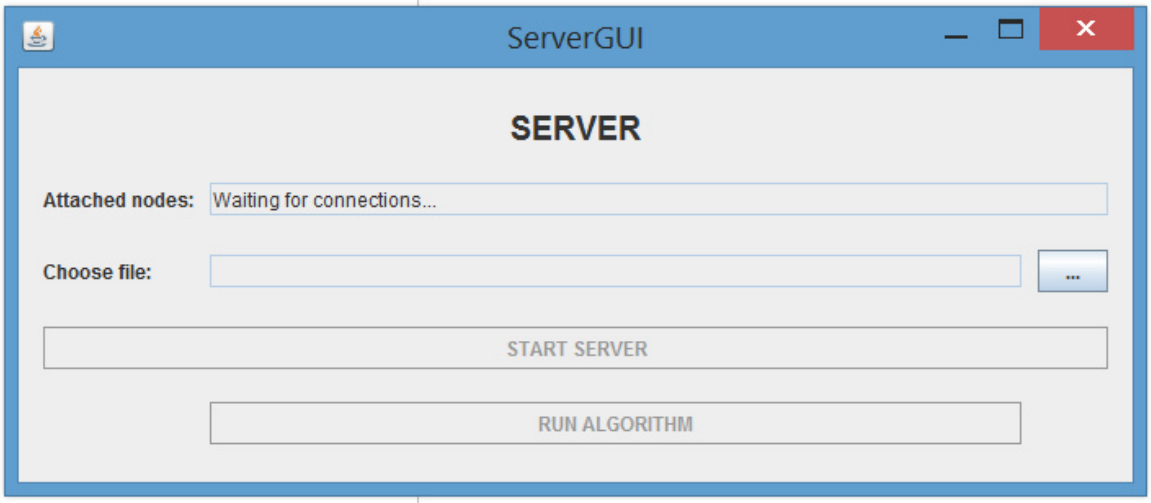
- **Zastosowanie w projekcie**

Klasa RegisteredNode pełni rolę sponsora, klasa LoadBalancer pełni rolę Selectora, klasy MapDispatcher oraz ReduceDispatcher pełnią rolę klientów. Wzorzec ułatwia wybór najlepszych Node'ów którym następnie zlecane są polecenia do wykoniania.



GUI

Wygląd aplikacji po stronie serwera.



Wygląd aplikacji po stronie klienta (node).