Design Patterns - Load balancer

Autor: Kamil Barszczak

1. Opis projektu

System stanowi warstę pośredniczącą pomiędzy oprogramowaniem a bazą danych. Celem tej warsty jest implementacja mechanizmu Load Balancera, który odpowiedzialny będzie za synchornizację danych pomiędzy bazami oraz wykonywanie zapytań Select na jednej z wybranych za pomocą algorytmu baz. Projekt jest wykonany w postaci modułu w Javie, który będzie można przenieść do różnych projektów.

2. Funkcjonalności systemu

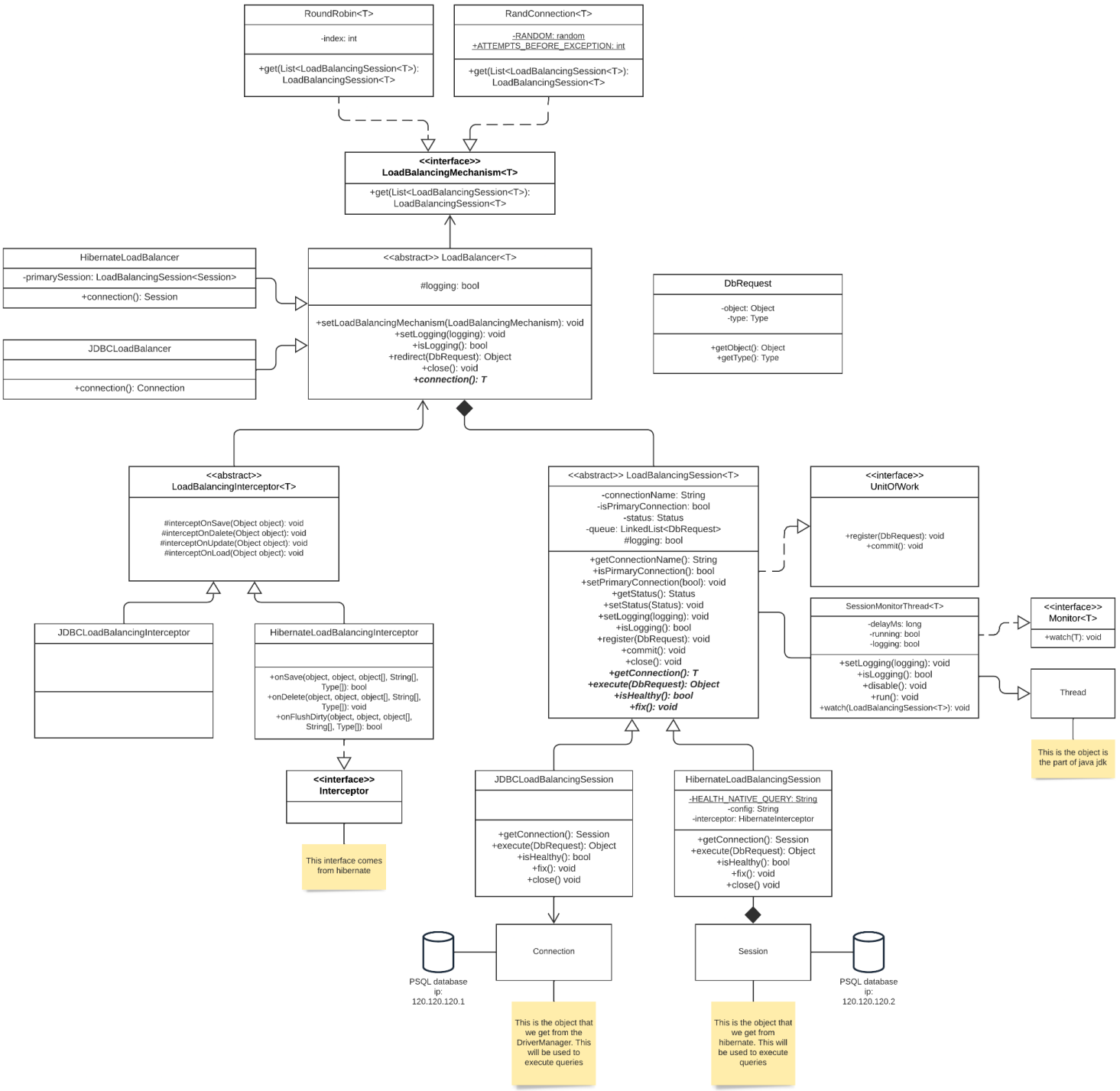
System umożliwa wykonanie następujących czynności:

- utworzenie LoadBalancera ze zdefiniowanymi połączeniami do wielu baz stworzonych z wykorzystaniem tej samej technologii.
- synchronizację danych pomiędzy bazami
- wykonywanie zapytań typu Select na jednej z wybranych baz
- zmienianie parametrów działania LoadBalancera (tj. rodzaj użytego algorytmu load balancingu) w trakcie działania systemu.

3. Architektura

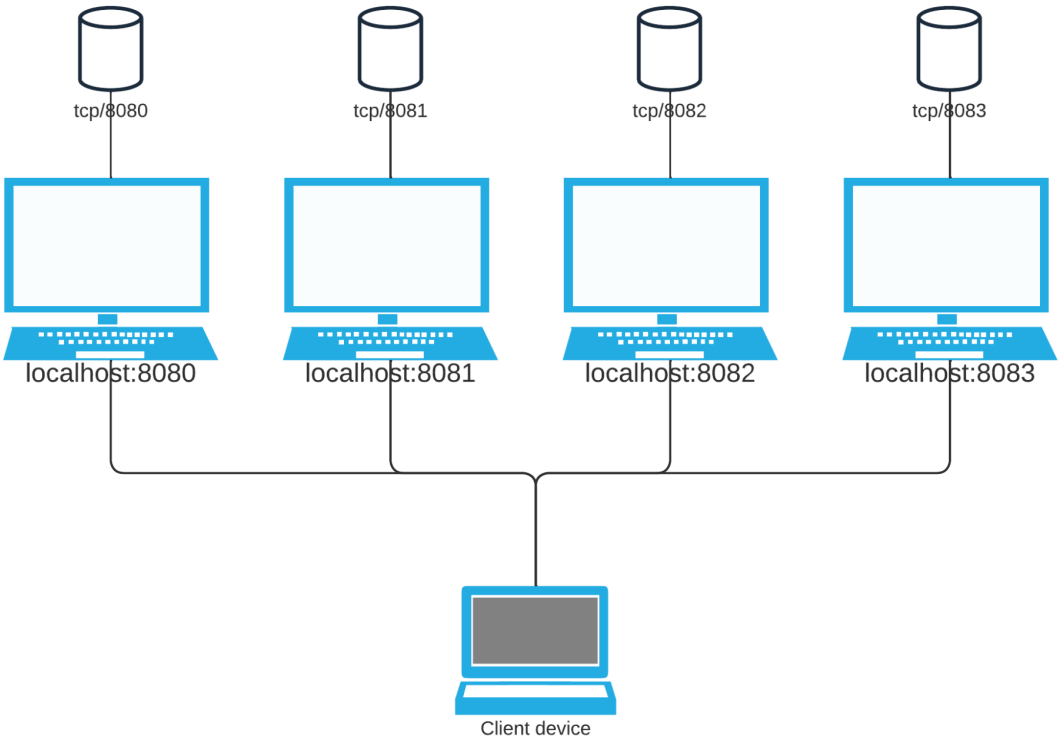
3.1. Architektura logiczna aplikacji

Poniższy diagram uml przedstawia wszystkie klasy znajdujące się w projekcie wraz z ich powiązaniami i relacjami. Na diagramie zawarte są również wszystkie metody oraz atrybuty dostępne w poszczególnych klasach.



3.2. Architektura fizyczna aplikacji

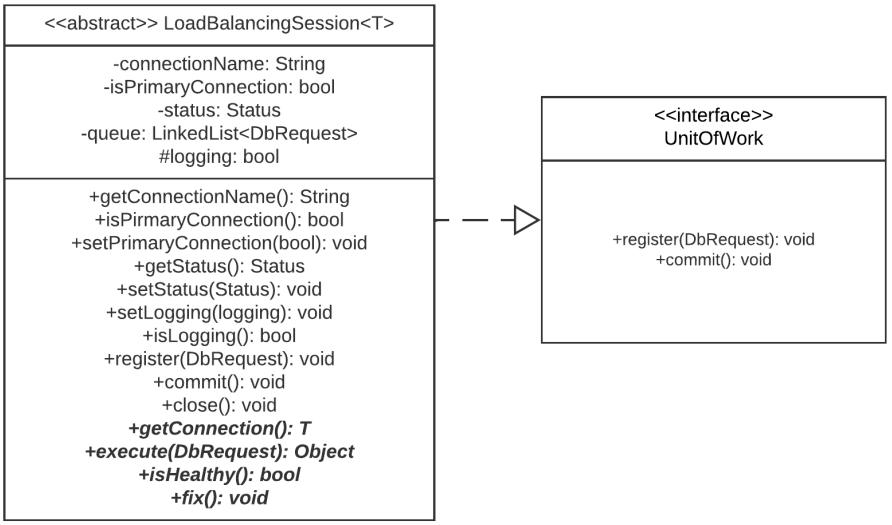
Architekturę fizyczną można opisać jako system składający się z 4 serwerów, każdy z nich podłączony jest do swojej bazy danych. Do tych 4 serwerów łączy się urządzenie klienta z odpowiednio skonfigurowanym LoadBalancerem, który zarządza bazami dostępnymi na każdym z serwerów. Architektóra ta wraz z przykładowymi adresami zobrazowana jest na poniższym obrazku.



3.3. Wzorce projektowe

3.3.1. Unit of work

Wzorzec odpowiada za spójność danych znajdujących się na poszczególnych serwerach. Dzięki niemu możliwe jest wykonywanie zapytań typu insert, update, delete na serwerach, które nie są aktywne. Dane zostaną odpowiednio zapisane w pamięci podręcznej i wysłane do serwera, gdy tylko ten znowu będzie aktywny. Implementacja w projekcie przedstawiona jest na poniższym diagramie.

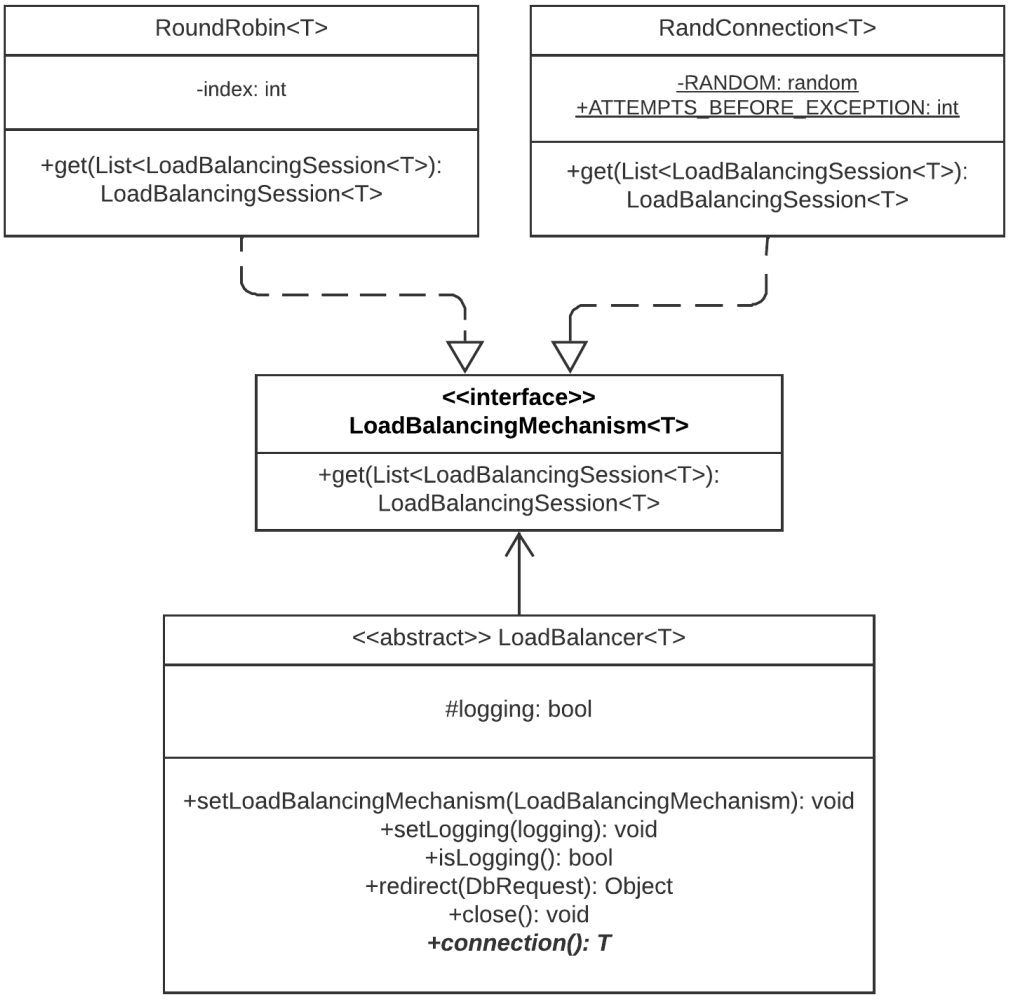


Interface UnitOfWork definiuje 2 metody tj.:

- register: metoda odpowiedzialna za zapisanie danych w pamięci podręcznej
- commit: metoda odpowiedzialna za wysłanie danych z pamięci podręczniej do serwera

3.3.2. Strategia

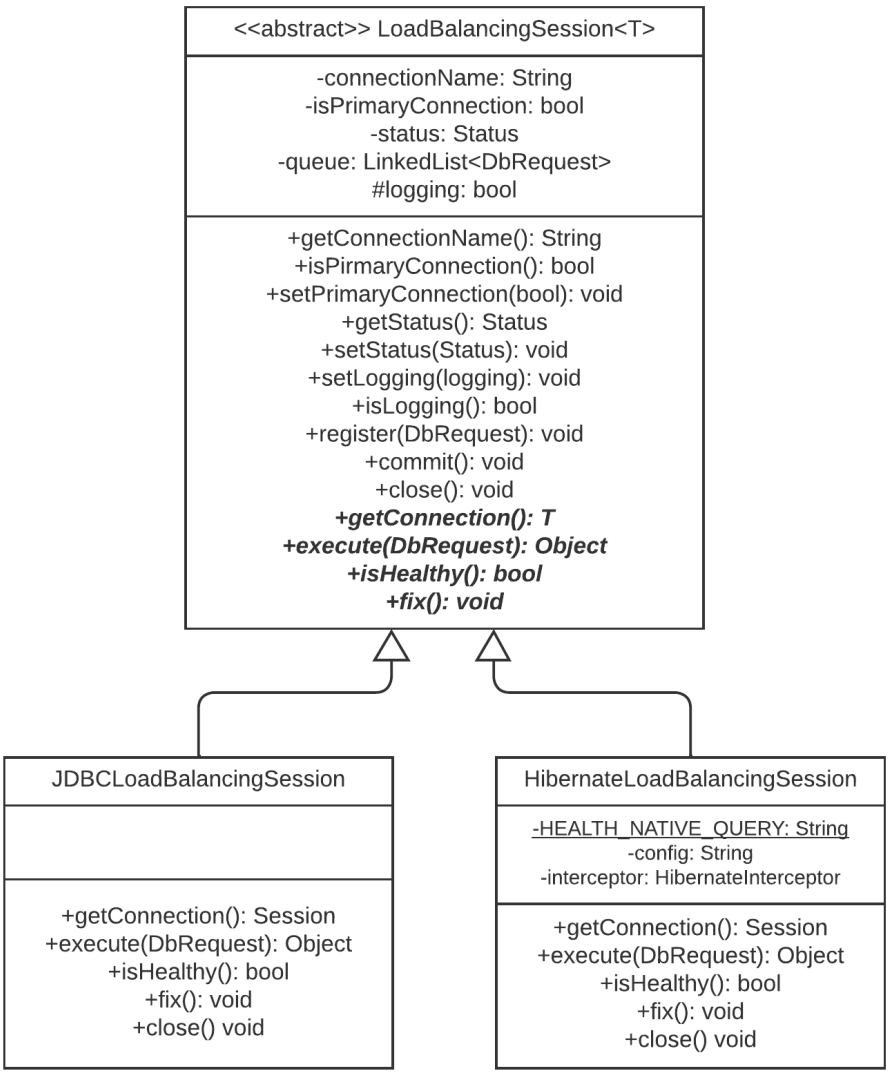
Wzorzec umożliwia łatwą podmianę algorytmu, który odpowiedzialny jest za wybieranie serwera, do którego wysłane zostanie zapytanie typu select. Wzorzec przedstawiony jest na poniższym diagramie.



Wzorzec zaimplementowany jest poprzez interface LoadBalancingMechanism, który definiuje metodę get. Metoda ta odpowiedzialna jest za wybranie odpowiedniego serwera i zwrócenie referencji do niego.

3.3.3. Metoda szablonowa

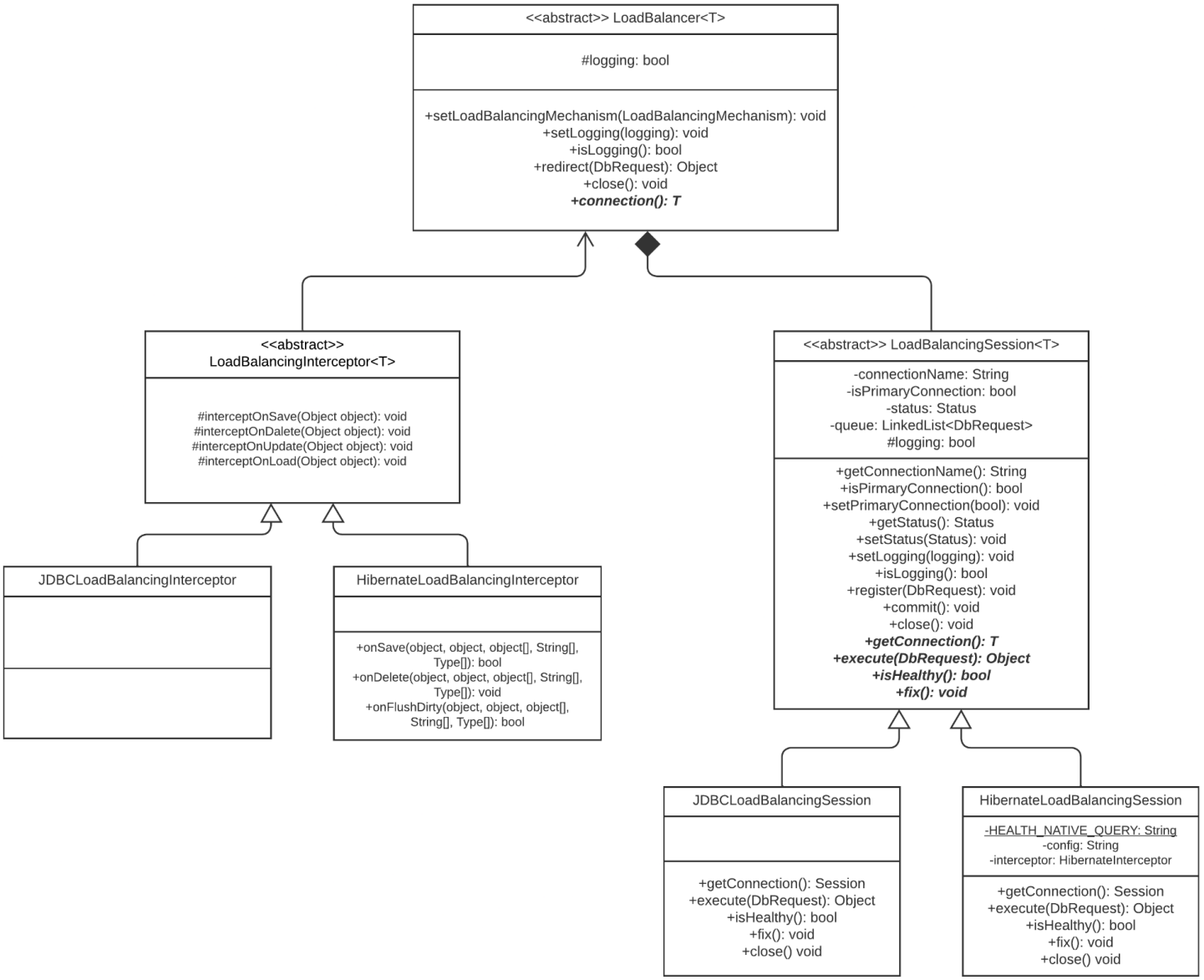
Wzorzec umożliwia zdefiniowanie szablonu pewnej metody, której ostateczne działanie będzie zależało od niezaimplementowanej abstakcyjnej metody, która musi być przedefiniowana w klasie potomnej. W projekcie prezentuje się to następująco.



Metodą szablonową jest metoda commit w klasie LoadBalancingSession. Jej definicja wykorzystuje metodę execute, która w tej klasie jest metodą abstrakcyjną. Definicja tej metody znajduje się w klasach JDBCLoadBalancingSession oraz HibernateLoadBalancingSession.

3.3.4. Most

Wzorzec umożliwia rozdzielenie 1 skomplikowanej hierachi dziedziczenia na 2 pomniejsze połączone właśnie przez most. Dzięki temu wzorcowi możliwe jest napisanie bazowych klas LoadBalancera w taki sposób, że jego przeniesienie na inny framefork jest proste i bezproblemowe.



Wzorzec rozdziela następujące hierachie dziedziczenia: LoadBalancingInterceptor oraz LoadBalancingSession. Hierarchie te połączone są przez klasę LoadBalancer, który odgrywa rolę mostu.

4. Szczegóły implementacyjne dla Hibernate

Działanie LoadBalancera dla hibernate'a opiera się głównie o 2 klasy. HibernateLoadBalancingInterceptor oraz HibernateLoadBalancingSession. Pierwsza z tych klas implementuje interface Interceptor, który jest udostępniany przez hibernate. Klasa ta przedefiniowuje w tym interfejsie metody, które wywoływane są w momencie dodawania, aktualizowania oraz usuwania obiektu z bazy danych. Implementacja tego interfejsu jest konieczna, aby hibernate uruchamiał te metody podczas wykonywania odpowiednich zapytań. Klasa HibernateLoadBalancingSession składa się z 2 obiektów typu Session, które odpowiedzialne są za połączenie do bazy danych. Jeden z nich jest interceptowany przez wyżej wspomniany interceptor a drugi z nich nie jest w żaden sposób modyfikowany. Interceptowany obiekt klasy Session wykorzystywany jest do wykonywania poleceń, które przychodzą od użytkownika natomiast obiekt Session, który nie jest interceptowany wykorzystywany jest na potrzeby wysyłania zapytań do baz wewnątrz systemu (nie chcemy, aby zapytania pochodzące z wnętrza aplikacji były przerywane, ponieważ doprowadziłoby to do duplikacji danych. Stąd koniecznie było takie rozdzielenie).

5. Sposób używania aplikacji

Systemu można używać tylko w środowisku powizanym z językiem Java. Aby użyć LoadBalancera należy dodać opowiednie dependency w zależności od frameworku, którego chcemy użyć. Dla hibernate wygląda to następująco:

```
<dependency>
    <groupId>agh.isi</groupId>
    <artifactId>hibernate-load-balancer</artifactId>
    <version>1.0</version>
</dependency>
```

Od tego momentu mamy dostęp do klas LoadBalancer oraz HibernateLoadBalancer. Teraz potrzebujemy plików koniguracyjnych dla hibernate, które należy umieścić w katalogu resources projektu. Musimy w nich zdefiniować wszystkie dane potrzebne do wykonania połączenia oraz wszystkie klasy, które będę encjami w bazie. Każda baza wymaga osobnego pliku konfiguracyjnego. Następnie listę nazw plików konfiguracyjnych należy podać jako argumenty w konstruktorze klasy HibernateLoadBalancer. Cała reszta wykona się automatycznie. Od tego momentu, uzyskujemy dostęp do metody connection w klasie HibernateLoadBalancer, która zwraca obiekt Session, na którym można wykonywać operacje w bazie. Ważne jest to aby po każdej zakończonej operacji ponownie wywołać metodę connection. Niepoprawnym sposobem wykorzysywania LoadBalancera jest ciągłe wykorzystywanie obiektu Session, który zostanie zwrócony z pierwszym wywołaniem metody connection.

Poniżej pokazany jest poprawny przykład wykorzystania klasy HibernateLoadBalancer.

```
try (LoadBalancer<Session> loadBalancer = new HibernateLoadBalancer(List.of(
        "/hibernate/hibernate-psql-1.cfg.xml",
        "/hibernate/hibernate-psql-2.cfg.xml",
        "/hibernate/hibernate-psql-3.cfg.xml",
        "/hibernate/hibernate-psql-4.cfg.xml"
))) {
    try {
        Session session = loadBalancer.connection();
    } catch (PersistenceException | IllegalArgumentException | IllegalStateException exception) {
        DBLogger.getLogger(Main.class).warning( msg: "The following exception occurred: " + exception.getMessage());
    }
} catch (Exception exception) {
    DBLogger.getLogger(Main.class).severe( msg: "The following critical exception occurred: " + exception.getMessage());
    exception.printStackTrace();
}
```

6. Stos technologiczny

Projekt wykorzysta następujące technologie:

- Java (17) jako język programowania
- Java Hiernate (6.1.6 final) Interceptor jako narzędzie do stworzenia warstwy pomiędzy oprogramowaniem a bazami danych
- PSQL (14) jako system zarządzania bazą danych