INF2ATS, 13.04.2023

Team name number 4

Team members:

Group-PKB:

Name	Matric number	E-mail
Paweł Król	110696	82705@student.pb.edu.pl
Krzysztof Kopczewski	109710	80452@student.pb.edu.pl
Arkadiusz Mroczkowski	109814	80561@student.pb.edu.pl

Group-PQL:

Name	Matric number	E-mail
Adrian Gołębiowski	99008	37463@student.pb.edu.pl
Łukasz Kowalewski	105588	74725@student.pb.edu.pl
Aleksander Anchimiuk	118968	94175@student.pb.edu.pl

1. Achievements and problems in this iteration

W drugiej iteracji wprowadziliśmy podstawową strukturę kodu dla naszego Static Program Analyser (SPA), implementując podstawowe gramatyki dla języków SIMPLE i PQL. Skupiliśmy się na rozwoju:

- Tokenizacji - analizie leksykalnej kodu źródłowego
- Logiki parsowania - przekształcaniu tokenów w struktury danych
- Obsługi zapytań - interpretacji zapytań PQL

Główne osiągnięcia:

- Podział projektu na moduły:
    - frontend - parsowanie SIMPLE
    - pkb - drzewo składniowe (AST) i zarządzanie danymi
    - pql - parsowanie zapytań
- Wykorzystanie adnotacji Lombok do redukcji boilerplate kodu
- Implementacja własnych wyjątków (np. TokenMatchingException) dla lepszej obsługi błędów
- Pełne parsowanie SIMPLE
- Działające poprawnie zapytania PQL z metodą Modifies

Napotkane problemy:

- Konieczność dopracowania obsługi złożonych zapytań PQL

2. Plan projektu

2.1 Plan całego projektu

	Iteracja 1								
Team Member	Set up & Config	Front end & Testing	Simple Parser Fixes	PQL Processor	PKB & AST Integration	Documentation	FULL SIMPLE Parser	Expanded PKB	Such-that Query Processor
Adrian				X					X
Aleksander					X				X
Arkadiusz			X					X	
Krzysztof		X						X	
Paweł	X		X				X		
Łukasz						X			

2.2 Plan iteracji

Imię	Parsowanie PQL	Query PQL -> PKB	PKB	Simple Parser	Doc + Simple
Adrian	X				
Aleksander		X			
Arkadiusz			X		
Krzysztof			X		
Paweł				X	
Łukasz					X

3. Diagramy UML

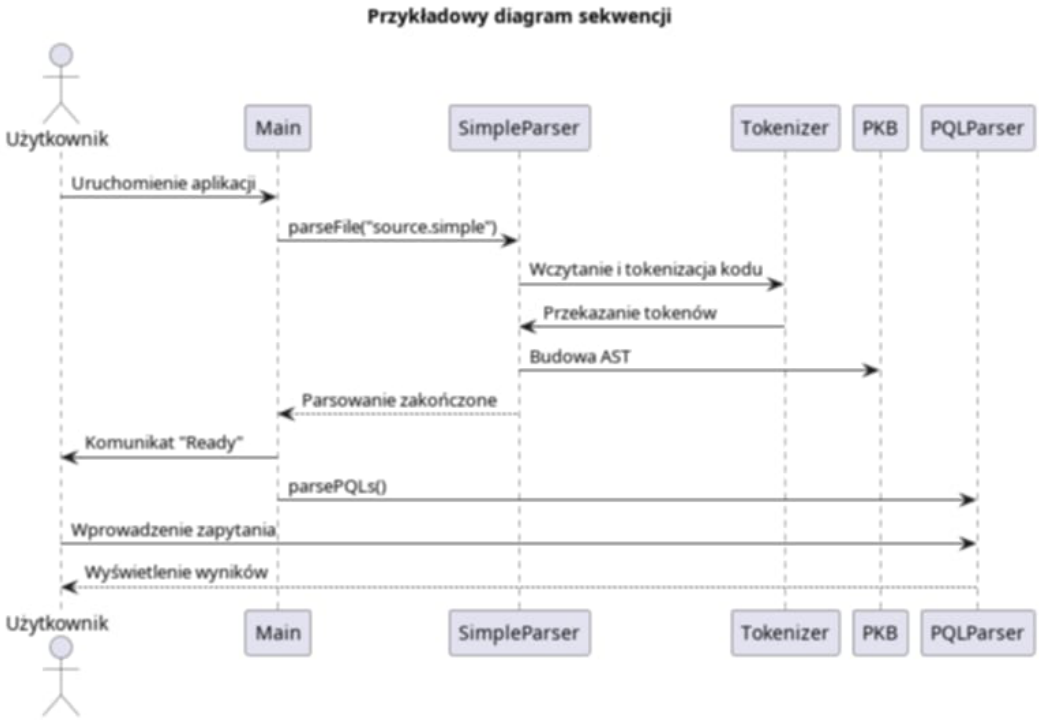

Diagram sekwencji stanowi kluczowe narzędzie w naszym projekcie, ponieważ wizualizuje przepływ komunikacji między głównymi modułami systemu. W ramach projektu praca została podzielona na kilka wyspecjalizowanych obszarów: jedna część zespołu koncentruje się na parsowaniu zapytań PQL, inna odpowiada za przekazywanie zapytań do modułu PKB, a oddzielna grupa zajmuje się obsługą danych w PKB. Dodatkowo, przetwarzanie kodu Simple realizowane jest przez dedykowany zespół, natomiast integracja dokumentacji z Simple Parserem stanowi kolejny, ważny obszar. Taki podział pozwala na precyzyjne określenie ról i obszarów odpowiedzialności, co usprawnia identyfikację potencjalnych problemów i optymalizację współpracy między modułami. Diagram ułatwia planowanie testów integracyjnych, ponieważ pozwala wyraźnie określić krytyczne punkty interakcji między systemami. Jego przejrzysta struktura jest cennym elementem spotkań zespołu, na których omawiane są kwestie integracji i wdrażania nowych funkcjonalności. W rezultacie, narzędzie to przyczynia się do spójności systemu oraz efektywności całego procesu rozwoju projektu.

4. Dokumentacja istotnych decyzji projektowych

4.1 Reprezentacja danych (ADT)

Struktury danych

AST (Abstract Syntax Tree)

Typy węzłów (enum EntityType):

java

PROGRAM, PROCEDURE, STMTLIST, STMT, ASSIGN, WHILE, OPERATOR, VARIABLE, CONSTANT

Implementacja:

Wzorzec Singleton (AST.getInstance()).

Przechowywanie węzłów w ArrayList<TNode>.

VarTable (tabela symboli):

Singleton z metodami:

java

insertVar(String name), getVarIndex(String name), isIn(String name)

Dane przechowywane w HashMap<String, Integer> (nazwa zmiennej → indeks).

Interfejsy

PqlObject – baza dla zmiennych (Variable) i instrukcji (Statement).

Condition – reprezentuje warunki PQL (np. Modifies, Uses).

4.2 Szybki dostęp do danych

Optymalizacje

Cache'owanie danych:

Wyniki parsowania SIMPLE są przechowywane w AST (unika wielokrotnego przetwarzania).

Refleksja w PQLParser:

Dynamiczne tworzenie obiektów warunków (np. Modifies) na podstawie nazwy:

java

// Przykład z PQLParser.java

Condition condition = (Condition) Class.forName("pql.conditions." + conditionName).newInstance();

Plusy: Eliminacja długich switch-case.

Minusy: Brak kontroli typów w czasie kompilacji (wyjątki runtime).

String pooling:

Optymalizacja operacji na stringach podczas tokenizacji (np. Tokenizer.matchToken()).

4.3 Pobieranie danych dla zapytań

Proces ewaluacji

Tokenizacja zapytania:

Podział na słowa kluczowe (np. Select, such that).

Budowa obiektu Query:

Przykład struktury:

java

```
public class Query {
   private String selectVar;        // np. "s"
   private List<Condition> conditions; // np. Modifies(s, "x")
}
```

Wykonanie zapytania:

Mapowanie warunków na metody PKB (np. Modifies(s, "x") → PKB.getModifies(s, "x")).

Wyniki zwracane jako List<String> (np. numery instrukcji).

4.4 Wzorce projektowe

Singleton

Zastosowanie: AST, VarTable.

Kod:

java

// Przykład z AST.java

```
public static synchronized AST getInstance() {
   if (instance == null) instance = new AST();
   return instance;
}
```

Konsekwencje:

Globalny dostęp do stanu.

Problemy z testowaniem (konieczność resetu między testami).

Refleksja

Zastosowanie: Dynamiczne tworzenie warunków PQL w PQLParser.

Ryzyko: Błędy w nazwach klas prowadzą do ClassNotFoundException.

Factory Method (w planach)

Zamiana refleksji na fabryki warunków (np. ConditionFactory.create("Modifies")).

4.5 Kompromisy i wnioski

Singleton vs. testy:

Wymusza użycie @BeforeEach w testach do czyszczenia stanu.

Refleksja vs. bezpieczeństwo:

Wymaga dodatkowej walidacji nazw warunków.

Wydajność:

ArrayList i HashMap w PKB są wystarczające dla małych programów, ale mogą wymagać optymalizacji dla dużych danych.

5. Standardy kodowania

Nasz zespół stosuje następujące standardy kodowania:

- **Konwencje nazewnicze:**
    - Klasy i interfejsy: PascalCase (np. SimpleParser)
    - Metody i zmienne: lowerCamelCase (np. parseStatementList())
    - Pakiety: małe litery (np. frontend, pkb, pql)
- **Organizacja kodu:**
    - Każda klasa/interfejs/enum w osobnym pliku
    - Logiczne grupowanie plików według funkcjonalności
    - Spójne wcięcia i organizacja importów
- **Narzędzia:**
    - Lombok (np. @Getter, @Setter) do redukcji boilerplate
    - Maven do zarządzania zależnościami
    - Specyficzne wyjątki (np. TokenMatchingException)
- **Testowanie:**
    - Regularne testy jednostkowe i integracyjne
    - Dedykowane katalogi na testy

6. Przetwarzanie zapytań

6.1 Reguły walidacji zapytań

Sprawdzanie argumentów relacji (liczba i typ)

Weryfikacja zgodności typów zadeklarowanych zmiennych

Poprawność składniowa i spójność zapytania

6.2 Ewaluator zapytań

Reprezentacja danych:

Obiekty reprezentujące zapytania (np. klasa Query)

Warunki (np. Modifies) przechowujące swoje argumenty

Podstawowa ewaluacja zapytań (BQE):

1. Identyfikacja typu warunku
2. Pobranie kandydatów z PKB pasujących do parametrów
3. Filtracja według ograniczeń
4. Generowanie wyników zgodnie z klauzulą Select

Załącznik A: Abstrakcyjne API modułu PKB

1. AST (Abstract Syntax Tree – Abstrakcyjne Drzewo Składniowe)

Kluczowe elementy:

EntityType (enum): Typy węzłów drzewa:

```
java

PROGRAM,     // Program główny
PROCEDURE,   // Procedura
STMTLIST,    // Lista instrukcji
STMT,        // Instrukcja (ogólna)
ASSIGN,      // Instrukcja przypisania
WHILE,       // Pętla while
OPERATOR,    // Operator (np. +, -)
VARIABLE,    // Zmienna
CONSTANT     // Stała
```

- **AST (klasa, Singleton):**

```
java

public static AST getInstance()  // Zwraca instancję drzewa
public void addNode(TNode node)  // Dodaje węzeł do drzewa
```

- **TNode (klasa):** Reprezentacja pojedynczego węzła:

```
java

private EntityType type;  // Typ węzła
private String value;     // Wartość (np. nazwa zmiennej)
```

---

2. Modele danych

Interfejsy:

PqlObject:

```
java

String getName();  // Zwraca nazwę obiektu (np. zmiennej)
```

- **Condition:**

```
java

String getName();  // Zwraca nazwę warunku (np. "Modifies")
```

Klasy implementujące:

- **Variable (zmienna):**

```
java
```

```
private String name;  // Nazwa zmiennej
```

- **Statement (instrukcja):**

```
java


private int lineNumber;  // Numer linii w kodzie
```

- **Modifies (relacja):**

```
java


private String stmt;  // Instrukcja
private String var;   // Zmienna
```

---

3. Zarządzanie zmiennymi (VarTable)

- **Singleton** przechowujący zmienne w programie.
- **Metody:**

```
java


public int insertVar(String name)  // Dodaje zmienną, zwraca jej indeks
public String getVarName(int index)  // Zwraca nazwę po indeksie
public boolean isIn(String name)  // Sprawdza, czy zmienna istnieje
```

---

4. PQL (Query Language – Język Zapytań)

Główne klasy:

- **Query:** Reprezentacja zapytania PQL.

```
java


private String selectVar;        // Zmienna do zwrócenia (np. "s")
private List<Condition> conditions; // Warunki (np. Modifies(s, "x"))
```

- **PQLParser:** Parsuje zapytania PQL.

```
java


public Query parseQuery(String input)  // Zamienia tekst na obiekt Query
```

---

5. Frontend (parsowanie SIMPLE)

Klasy:

- **SimpleParser:** Główny parser kodu SIMPLE.

```java
public void parseFile(String code)  // Parsuje cały plik
public void parseWhileStmt()       // Parsuje pętlę while
```

- **Tokenizer:** Zamienia kod na tokeny.

```java
public String matchToken(String expected)  // Sprawdza i konsumuje token
```

- **TokenMatchingException:** Wyjątek dla błędów parsowania.

---

6. Klasa Main

Punkt wejścia programu:

```java
public static void main(String[] args) {
    SimpleParser.parseFile("przyklad.simple");
    PQLParser.parseQuery("Select s such that Modifies(s, 'x')");
}
```