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

In this initial iteration, we introduced the foundational codebase for our Static Program Analyser (SPA) by implementing core grammars for both SIMPLE and PQL, focusing on front-end tokenization, parsing logic, and query handling. We successfully organized the project into separate modules (`frontend` for SIMPLE parsing, `pkb` for abstract syntax tree and data handling, and `pql` for query parsing) and leveraged Lombok annotations to reduce boilerplate. A significant achievement was the smooth integration of custom exceptions (like `TokenMatchingException`) to handle parsing errors in a clear, maintainable fashion. Nonetheless, we encountered challenges ensuring completeness of the grammar definitions—particularly for more intricate PQL constructs—and recognized the need for further refinements in subsequent iterations to fully capture complex expressions and relationships.

2. Describe project plans

2.1 Plan for the whole project

	Iteracja 1							Iteracja 2		
Team Member	Set up & Config	Frontend & Testing	Simple Parser	PQL Processor	PKB & AST Integration	Documentation	FULL SIMPLE	Expanded PKB	Such-that Query	With Clause & Boolean

			Fixes				Parser		Processor	
Adrian	X									
Aleksander		X								
Arkadiusz			X							
Krzysztof			X							
Paweł				X						
Łukasz						X				

2.2 Plan for this development iteration

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

3. UML diagrams

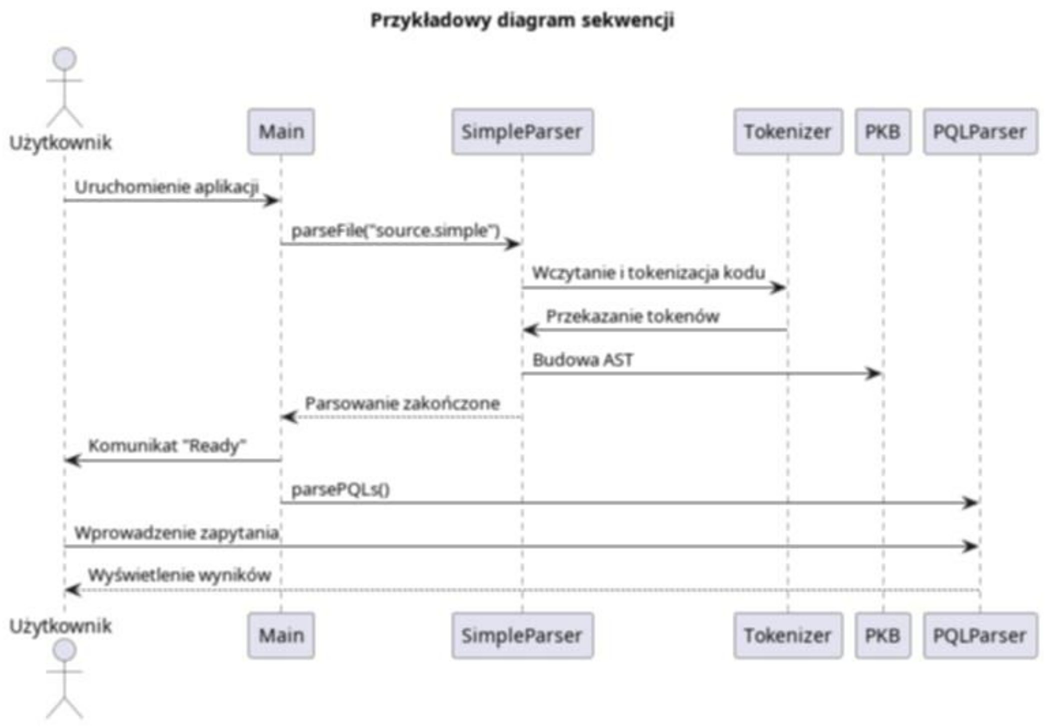

Diagram sekwencji stanowi kluczowe narzędzie w naszym projekcie, ponieważ wizualizuje przepływ komunikacji między głównymi modułami systemu. W ramach projektu praca została podzielona na kilka wyspecjalizowanych obszarów: jedna część zespołu koncentruje się na parsowaniu zapytań PQL, inna odpowiada za 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. Documentation of important design decisions

Data Representations (ADTs)

- ADTs (AST nodes, variables, statements) via unified interfaces.

- Singleton pattern for AST and VarTable ensures global state control.

- In-memory structures (ArrayList, HashMap) enable fast access.

Fast Data Access Solutions

- Caching of parsed data to avoid reprocessing.

- Reflection in PQLParser for dynamic, boilerplate-free object creation.

- Efficient string manipulation during parsing.

Data Fetching during Query Evaluation

- Multi-stage processing: tokenization → parsing → condition matching.
- Iterative search on in-memory collections for simplicity and speed.

Applied Design Patterns

- Singleton (AST, VarTable): simple control, but needs test isolation.
- Reflection (PQLParser): flexible, but adds error handling burden.

5. Describe coding standards adopted by your team

Our team adheres to a set of coding standards that emphasize readability, maintainability, and consistency across the entire codebase: we use descriptive, PascalCase names for classes and interfaces, while methods, variables, and packages follow standard Java naming conventions (lowerCamelCase for methods and variables, all-lowercase for packages); each class, interface, or enum resides in its own file for clarity and modularity; we consistently rely on Lombok annotations (e.g., `@Getter`, `@Setter`, `@NoArgsConstructor`) to reduce boilerplate and maintain clean code; we group files logically by functionality (e.g., `frontend`, `pkb`, `pql`) to clearly separate concerns and facilitate easier navigation; we use Maven for build management, ensuring that dependencies, compiler versions, and plugins are defined in a straightforward manner in the `pom.xml` file; we treat error handling carefully, favoring specific exceptions like the custom `TokenMatchingException` to provide meaningful feedback; we follow a consistent indentation style and organize imports in a standard way so that all team members can easily read and contribute to the code; finally, we incorporate frequent testing (supported by placing test files in dedicated `tests` folders) to verify functionality and guard against regressions, ensuring that all team members can confidently modify and extend the project.

6. Query processing

6.1 Query Validation Rules

- Relationship arguments: Verify each relationship (e.g., Modifies) has the correct number and type of arguments per PQL specifications.
- Declared types: Ensure variables in the query match their declared types (e.g., stmt, variable).
- Syntax/consistency: Confirm the query adheres to PQL’s format, and references declared entities correctly.

6.2 Query Evaluator

6.2.1 Data Representation for Program Queries

- Objects: Queries are represented by objects (e.g., a Query holding a Condition) which store details like Select targets and arguments (stmt s, variable v, etc.).
- Conditions: Each condition (e.g., Modifies) encapsulates its arguments to facilitate evaluation against the PKB (Program Knowledge Base).

6.2.2 Basic Query Evaluation (BQE)

1. Identify condition type (e.g., Modifies, Uses).
2. Gather candidates from the PKB that match the condition’s parameters.
3. Filter based on any constraints (e.g., statement vs. variable).
4. Produce results relevant to the query’s Select clause (e.g., statement numbers or variable names).

This approach supports simple relationships and can extend to more complex queries (multiple clauses, patterns, etc.) as needed.

7. Discussion

Appendix A: Abstract PKB API

AST (Abstract Syntax Tree)

- - EntityType (enum): PROGRAM, PROCEDURE, STMTLIST, STMT, ASSIGN, WHILE, PLUS, VARIABLE, CONSTANT

- - AST (class, Singleton): public static AST getInstance()

- - TNode (class): private EntityType type

Models

- - PqlObject (interface): String getName()
- - Condition (interface): String getName()
- - Variable (class): String name, implements PqlObject
- - Statement (class): String name, implements PqlObject
- - Modifies (class): String var1, var2, implements Condition

Zarządzanie zmiennymi

- - VarTable (class, Singleton): insertVar, getVarName, getVarIndex, getSize, isIn

PQL (Query Language)

- - Query (class): selectVar, condition, toString()
- - PQLParser (class): parsePQLs(), parseVariables(), parseQuery(), createVariable()

Frontend

- - SimpleParser (class): parseFile(), parseProcedure(), parseStatementList(), parseStatement(), parseAssign(), parseWhile(), parseExpression()
- - Tokenizer (class): matchToken(), matchName(), isNextToken(), removeFromCode(), throwMatchingException()

- - TokenMatchingException (class): extends RuntimeException

Klasa Main

- - Main (class): parsowanie kodu Simple i zapytań PQL