Report

Object-Oriented Design

Overall OOP Design

Our program is designed with a clear and modular object-oriented structure to enhance code organization, maintainability, and extensibility. The primary classes include `GenericsKbBSTApp`**,** `GenericsKbArrayApp`**,** `BinarySearchTree`**,** `BinaryTreeNode`**, and** `Statement`**.**

- **GenericsKbBSTApp: Represents the main application for the Binary Search Tree (BST) version.**
- **GenericsKbArrayApp: Represents the main application for the Array version.**
- **BinarySearchTree: Implements the Binary Search Tree data structure for storing** `Statement` **objects.**
- **BinaryTreeNode: Represents a node within the Binary Search Tree.**
- **Statement: Represents a knowledge statement with fields for term, sentence, and confidence score.**

The separation of concerns is achieved by assigning specific responsibilities to each class. For example, the `BinarySearchTree` **class is responsible for the implementation of the BST structure, and the** `Statement` **class encapsulates the properties and behavior related to a knowledge statement.**

Traditional Array Implementation

In `GenericsKbArrayApp`, we opted for a simple array-based approach to store knowledge statements. The `Statement` objects are stored in a traditional array, allowing for straightforward management and indexing.

BST Implementation

In `GenericsKbBSTApp`, we utilized a Binary Search Tree to store and organize knowledge statements. The `BinarySearchTree` class provides methods for insertion, searching, and displaying the tree. The use of a BST facilitates efficient searching and retrieval operations.

Experimental Tests

We conducted experimental tests to verify the functionality of both implementations. Tests involved populating the knowledge base, updating statements, and searching for items. These tests confirmed the correctness of our OOP design, ensuring that each class operates as intended and collaborates seamlessly.

Description of Creativity

While adhering to the assignment's requirements, we introduced creativity in the design by incorporating a modular structure that allows for easy expansion. The separation of the main application class (`GenericsKbBSTApp` and `GenericsKbArrayApp`) from the data structure implementation (`BinarySearchTree` and array-based approach) enables future modifications and enhancements without extensive code modifications.

Our design also supports the potential integration of additional data structures or functionalities, providing a foundation for future improvements.

Testing

Testing Protocol Description

For testing, we employed a systematic approach to cover key functionalities. The testing protocol aimed to ensure the correctness of populating, updating, and searching the knowledge base. Additionally, interactive interface scenarios and boundary tests were conducted to evaluate the robustness of the application.

Test Populating the Knowledge Base

2.2.1 Test Values

- Test adding a statement about "dog": "Dogs are loyal companions."
- Test adding a statement about "tree": "Trees produce oxygen through photosynthesis."

2.2.2 Expected Outcomes

Both tests are expected to successfully add new statements to the knowledge base.

Test Updating the Knowledge Base

2.3.1 Test Values

- Test updating the statement about "dog" with a higher confidence score.
- Test updating the statement about "tree" with a lower confidence score.

2.3.2 Expected Outcomes

The first test should successfully update the statement for "dog," while the second test should print a message indicating that the confidence score for "tree" is already higher.

Test Searching the Knowledge Base

2.4.1 Test Values

- Test searching for the term "dog."
- Test searching for the term "cat."

2.4.2 Expected Outcomes

The first test should find and display the statement about "dog," while the second test should print a message indicating that there is no statement for "cat."

Additional Statements Loading Test

2.5.1 Test Values

- Load additional statements from the file "GenericsKB-additional.txt."

2.5.2 Expected Outcomes

This test is expected to successfully load additional statements into the knowledge base.

Interactive Interface Testing

2.6.1 Scenarios

- Test loading a knowledge base from a file.
- Test adding a new statement through user input.
- Test searching for an item in the knowledge base by term.

2.6.2 Expected Outcomes

All interactive scenarios should execute without errors, providing the expected outputs and prompting the user for correct inputs.

Boundary Tests

2.7.1 Test Values

- Test adding statements until the knowledge base is full.
- Test searching for a term that doesn't exist.

2.7.2 Expected Outcomes

The first test should handle a full knowledge base gracefully, and the second test should indicate that no statement was found for the non-existent term.

Output Sample Screenshots

Error Handling

The application handles errors in the following ways:

- Invalid confidence scores: The application checks and rejects confidence scores outside the range [0, 1].
- Invalid menu choices: The application prompts the user to enter a valid menu choice.
- File not found: The application prints an error message if the specified file for loading is not found.

Creativity

User-Friendly Menu Interface

Implementation of a user-friendly menu interface in the `displayMenu()` method. The menu provides a clear and intuitive way for users to interact with the program, offering options to load a knowledge base, add new statements, search for items, and display the current knowledge base. The menu-driven interface enhances user experience by guiding them through available functionalities, making the program accessible and easy to navigate.

Robust Error Handling

To ensure a robust user experience, I implemented error handling mechanisms throughout the program. For instance, when loading knowledge from a file, we check for invalid lines and appropriately handle exceptions, preventing potential runtime errors. Additionally, the program notifies users when attempting to add a new statement with a lower confidence score, preventing unintended data modifications.

These creative elements collectively contribute to a well-structured, user-friendly, and extensible Java program for managing and querying knowledge bases.

Git Usage

Git Log Summary

git log | (ln=0; while read l; do echo $ln\: $l; ln=$((ln+1));

```
    done) | (head -10; echo ...; tail -10)
0: commit 321abf71bcc06a8d35304b5c4ddbb8539b327a05
1: Author: WAVYTERRY <codeeeterry@gmail.com>
2: Date:   Fri Mar 1 12:38:10 2024 +0200
3:
4: Full Implementation of GenericsBST and GenericsArray
5:
6: commit 8970017f02c40b7974ddec52ef5ccbb88b252586
7: Author: WAVYTERRY <codeeeterry@gmail.com>
8: Date:   Tue Feb 27 17:46:22 2024 +0200
9:
../..
10: I have got the program working it just needs refinement
11:
12: commit 2fd165562d735ba4cc220147a44cee7023f92702
13: Author: WAVYTERRY <codeeeterry@gmail.com>
14: Date:   Wed Feb 14 08:18:14 2024 +0200
15:
16: Initial commit
```

Dev Tools

- I used GitHub for source code management.
- javadoc for documentation generation.
- CLI commands for easy debugging
- And make for automation of compilation and documentation generation.
- The IDE used was Visual Studio Code with Vim installed.