Design section

**Choice to implement tool as command line interface**

We made the design decision to implement the JSONTalk tool as a command line tool for the following reasons:

Accessibility: Command line tools can easily be used with screen readers, this would allow users who require screen readers to easily read the description with their own personal screen reader and keep their personal preferences.

Usability: The use of a command-line tool can offer a more convenient solution for visually impaired programmers than a graphical interface that relies on visual cues. Additionally, command-line tools can be used in conjunction with other tools, providing a versatile solution for programming tasks faced by the user.

Efficiency: Command-line tools can be faster and more efficient than graphical interfaces, as they do not require the rendering or processing of graphical elements.

Flexibility: Command-line tools are scalable and can be adapted to work with larger datasets or complex programming projects, providing a flexible solution that can meet various programming needs.

Integration: Command-line tools can be easily integrated with other programming tools and workflows, making them a valuable addition to any programmer's toolkit. This can improve collaboration among team members and streamline programming tasks.

Implementing the JSONTalk tool as a command-line tool offers a highly accessible, efficient, flexible, and integrated solution for visually impaired programmers working with JSON data.

Implementation section

**Resources used**

To accelerate the project's development, we leveraged the benefits of various existing libraries. In particular, we utilized the following libraries:

* Antlr4: For the crucial initial processing of the JSON files. Specifically, for its lexer and parser components, which were instrumental in creating a parse tree from the input JSON files.
* FreeTTS: For the ReadAloud functionality of the tool, which enabled users to have the generated natural language descriptions read aloud.
* Picocli: For implementing the tool as a Command Line Interface (CLI), making it easier for users to access and operate the tool. By utilizing these libraries, we were able to expedite the development process, enhance the tool's functionality, and improve the overall user experience.

ANTLR4 played a critical role in implementing the JSONTalk tool by providing a powerful parser generator that allowed us to construct a parse tree from the input JSON file. Utilizing ANTLR4's grammar syntax, lexer, and parser, we were able to extract the syntactic structure of the input and produce an accurate representation of the data. The subsequent use of a Visitor implementation to traverse the parse tree and generate natural language descriptions of the input added significant value to the tool's functionality. ANTLR4's customizable features and reliable performance enabled us to parse the JSON input with precision and efficiency. The use of the Visitor design pattern in combination with ANTLR4 provides a flexible and extensible approach to processing complex data structures.

**JSON Parsing and Data Extraction Using Visitor Design Pattern and Inheritance**

The visitor design pattern was employed to traverse each node of the abstract syntax tree generated from parsing the input JSON files. By doing so, we were able to extract the necessary information for creating a file description from each node, which was then stored in various objects.

The jsonElement object was utilized to store information regarding primitive JSON key-value pairs, including both named and anonymous pairs. Such values were defined as being of type STRING, NUMBER, BOOLEAN, or NULL.

To store information about complex JSON key-value pairs, the jsonComplexElement class was created as an extension of the jsonElement class. The jsonComplexElement class was used to store values that were of type OBJECT and ARRAY.

Furthermore, the jsonObject class extended the jsonComplexElement class and provided additional functionality that catered specifically to JSON objects. Similarly, the jsonArray class was also created as an extension of the jsonComplexElement class and provided functionality tailored to JSON arrays.

In order to provide users with precise nesting and depth information for each JSON element, we implemented a system in which nodes were added to a list of their parent node's "children" while traversing the abstract syntax tree. This ensured that each parent node was always a jsonComplexElement.

By utilizing this approach and storing each element as a child of its respective parent, we were able to generate a comprehensive file description. Specifically, we iterated through each jsonComplexElement object that was created and listed the details of its children. This allowed us to accurately represent the hierarchical structure of the JSON file and provide the user with detailed information on the contents of each element.

**Command line interface**