COP-4331 Final Project

Documentation

Argument
Contains an enum of types including INTEGER, DOUBLE, and STRING.
Its constructor has arguments of a name, a type, a validator (for type validation), and an error handler (for custom error handling).
It contains getter and setter methods.
It contains a validate function for validating the argument's value.

Command
This class contains a name, lists for parameters, and a consumer that accepts a list of strings as input.
Its constructor has arguments of a name, a list of arguments, and a consumer for strings.
It includes getters, setters, and an execute function to consume parameters.

Command Manager
This class is responsible for handling the different commands that will be registered. It contains a map that uses the name of the command as well as the actual command. It contains a register command method which is used to add a command to the map as well as execute command which takes in the name of the command as well as the required parameters of the command.

ErrorHandler
This class is responsible for catching an error that is thrown and displaying the appropriate message

Lexer
This is the class responsible for transforming the character passed into the program, through user input, into commands that will be used to output the appropriate result.

The format that our lexer recognizes is "(commandname[argument1, argument2,...]) with the amount of arguments depending on which command is used.

The lexer reads the user input and if something isn't in the specified format, such as a bracket missing or a wrong command name is used, a message will be printed telling the user what is wrong with what they input.

Main
This class is the entry point of the program. It initializes scenarios and provides a command-line interface for interacting with the system.

It initializes the scenarios, creates a scanner to continuously input from the command line until the exit command is called.
Finally, the main class parses and executes, and distributes the parsed command to the necessary file. This is the heart of the parser.

ParseException

The ParseException class represents an exception that occurs during command parsing. It is thrown when the program encounters an error in the parsing process.

ParsedCommand
The ParsedCommand class represents a parsed command and its associated tokens. This class stores the command name alongside a list of tokens that are associated with the command. It is the class that stores information about the command after the parser has been run.

Data Structures: String for commandName, List to store tokens

This class has a default constructor as well as a parametrized constructor. The default constructor creates an empty Array for tokens whereas the parametrized constructor involves the tokens being passed in as an argument to the constructor.
This class has an equals function which can be used to compare two different ParsedCommand classes.
This class has getters and setters to set the command name and tokens of the object. Another feature is that it has a hashCode() function which returns the hash code value of the object.

Parser
The parser class is a very important function in the implementation. This class represents a parser that parses tokens into a parsed command object. The Parser is initially provided with a stream of tokens to be parsed. The parse() method is arguably the most important method in the implementation as it parses through the tokens while looking for errors. If the parser encounters an error, it throws a ParseException.

The parser checks for syntactical errors, ensures that the identifiers are present and have the correct number of tokens, and checks to make sure all the tokens are of the correct types.
The peek method ensures that the tokens in the stream match each other.
The match function matches the provided objects against the next tokens in the stream
Finally, the test method tests if the provided object matches the given token.

The parser also contains a TokenStream class. This class represents a stream of tokens that the parser uses. This class has a constructor method and getter functions used by the main parser.

RetrieveCommand
This class contains a parse function that seals the command, making it accessible only within this class.

Scenarios
This class lets us define command line functions, as well as their acceptable input forms.
This class has a CommandManager that it registers commands to once they are created.
Validators and argument lists are used to fill the arguments parameter, and Consumers that define our functions in code are used to fill in the function parameter for Command instances. These Command instances are registered to the class' CommandManager with their corresponding CLI name.
Validators are defined in this class.

Token
The token class is declared as a record because its main use is to hold data, in this case it will be used to hold whether a sequence of characters is one of the following types: Number, Identifier, or operator.

The record consists of a string called value which will be the sequence of characters passed and read by the lexer and a Type called type which is used to identify the value string.

Validator
This interface contains a validate function.