Shoshana Altman, Zander Gall, and Farhan Mohamud

Professor Duerre

CSC 330: Language Design and Implementation

Spring Semester 2025: 23 April 2025

**A Domain Specific Language for Banking**

Assignment 7 asked us to work in groups to build a function domain specific language (DSL) for banking. For this DSL, we were required to include four banking functions (create account, deposit, withdraw, balance) and an exit function. We were asked to create our own syntax, lexer, parser, and interpreter, as well as include specification tests and an operational mode.

**Work Distribution**

Group 1 is composed of Shoshana Altman, Zander Gall, and Farhan Mohamud. Group 1 collaboratively created our EBNF, with all members participating in its writing and editing. The python code was split evenly between the three members based on the compiler stage. Shoshana created the Lexer and Token classes and wrote the specification test and main functions in banking.py. Zander created the Parser and ASTNode classes and contributed to banking.py. Farhan created Interpreter, BankAccount and Accounts.txt; Farhan also contributed to load\_sample\_accounts and normalOperation methods in banking.py.

**Code Explanation**

Our code may be run in a terminal by running “python” and passing the name of the program to be run, i.e. *python Bank/banking.py.* Alternatively, you may open the project in an environment with tools to run python programs like any IDE or a light environment like VisualStudio Code.

The banking.py file is the main program that runs a simple banking language created for our class project. The main function displays a startup menu to select the program's operational mode. These modes include 1) normal operation, which lets users input DSL commands, 2) run specification tests, which run the prewritten specification tests, and 3) print intermediates, which run the specification tests while printing intermediates. banking.py also includes code that verifies a valid menu selection.

Selecting option one (normal operation) leads to an input line that takes in DSL commands. The input line is within a while loop that continues until an exit command is detected. Our DSL lets users type in commands like "create account," "deposit," or "check balance," and it processes those commands using different parts of the code.

Selecting option two or three runs the specification tests, which use prewritten DSL commands to check if the DSL commands work correctly. These tests try out commands and make sure the results match the expected value, like checking that depositing money actually increases the account balance.

**DSL Specifications**

The Language we designed contains 5 commands. Each command is a word composed of letters and underscores, followed by several parameters depending on the command entered. The commands are “create\_account”, “deposit”, “withdraw”, “balance\_of”, and “exit” or “quit”.

Create Account Function

e.g.: create\_account SA100000 Shosh Altman $1000000000

The create account command has the syntax, “create\_account account\_number first\_name last\_name (optional\_amount)”. An account number is any two letters followed by 6 digits and will fail to compile if the first two letters do not match the first letter initials of the first and last name given. The first and last name are any series of letters, and correspond with the first and last name of the customer whose account is being created. And an amount is the character “$” followed by one or more digits (0-9), with an optional cent amount (a period followed by two more digits, e.g. “$10.99”).

Deposit and Withdraw

e.g.: deposit $20 into SA100000

withdraw $999999999 from SA100000

Depositing has the syntax “deposit amount into account\_number”, where amount and account number are the same syntax as described above, and “into” is the literal text “into”, serving as a keyword in the deposit syntax. Withdraw, similarly, is “withdraw amount from account\_number”, with amount and account number acting the same as before, and “from” being a keyword used to make the command more readable.

Balance

e.g.: balance\_of SA100000

Checking the balance of an account can be done with “balance\_of account\_number” providing the number of the account you’d wish to check the balance of.

Exit and Quit

e.g.: exit

quit

Exit and quit both serve as ending statements that close a program and take no parameters.

References

“enum – Support for enumerations.” *Python documentation*,<https://docs.python.org/3/library/enum.html>

“Extended Backus-Naur Form(EBNF).” *YouTube*, uploaded by Jacob Schrum, 6 Jul.2019,<https://www.youtube.com/watch?v=JO_0e9mPofY>

"Introduction to Grammars and EBNF." *YouTube*, uploaded by Mary Elaine Califf, 13 Aug.2021,[www.youtube.com/watch?v=F25ez8s3AsQ](http://www.youtube.com/watch?v=F25ez8s3AsQ).

JetBrains. "Domain-Specific Languages." *JetBrains MPS*, JetBrains,<https://www.jetbrains.com/mps/concepts/domain-specific-languages/>. Accessed 17 Apr. 2025.

“Make YOUR OWN Programming Language – EP 1 – Lexer.” *YouTube*, uploaded by CodePulse, 4 December 2018,<https://www.youtube.com/watch?v=Eythq9848Fg&list=PLZQftyCk7_SdoVexSmwy_tBgs7P0b97yD>

“Make YOUR OWN Programming Language – EP 3 – Interpreter.” *YouTube*, uploaded by CodePulse, 24 Jan 2019, https://www.youtube.com/watch?v=YYvBy0vqcSw&list=PLZQftyCk7\_SdoVexSmwy\_tBgs7P0b97yD&index=3

“Python File Open.” *W3Schools*,<https://www.w3schools.com/python/python_file_open.asp>

“Python Raise an Exception.” *W3Schools*,<https://www.w3schools.com/python/gloss_python_raise.asp>

“Python RegEx.” *W3Schools*,<https://www.w3schools.com/python/python_regex.asp>

“unittest – Unit testing framework.” *Python documentation*,<https://docs.python.org/3/library/unittest.html>