Архитектурное описание CLI. Афанасов Артем, Холодаева Екатерина.

Класс CLIKernel

Главный класс обработчика.
StartCLI -- точка входа в интерпретатор.

Производится обработка ввода пользователя следующим образом.

Один поток читает ввод пользователя и заполняет очередь InputQueue
Второй поток читает очередь InputQueue и обрабатывает прочитанное значение.

Действия во втором потоке:
Вызывается *Parser.ParseInput* для введенных пользователем данных для их обработки. Возвращает список кортежей: (название команды, список опций).
Затем вызываем *Creator.CreateCommandsList* для создания списка команд с аргументом списка кортежей, который получили из *ParseInput.*
Затем вызываем Activator.*HandlePipeline* с аргументом списка команд, который получили из *CreateCommandsList.*

Класс Parser
Класс, который методом *ParseInput* обрабатывает входную строку: работает с переменными окружения, двойными и одинарными кавычками, разбивает входную строку на команды, списки опций и списки stdin. Содержит таблицу *EnviromentVariable*, в которую загружаются переменные окружения пользователя.

Сначала строка нормализуется методом *NormalizeInput*: учитывая двойные и одинарные кавычки, производиться замена имен переменных на их значения; редуцируются пробелы.
Затем методом *ReadVariable* проверяется, задаёт ли пользователь во входной строке значение новой переменной и если задаёт, то производится запись или изменение переменной.
Затем нормализованная строка методом *ParseIntoCommandsAndOptionsAndStdins* преобразуется в список кортежей (название команды, список опций, список stdin).

NormalizeInput:
Сначала поиск вне двойных кавычек. Если есть строки с одинарными кавычками типа ’string’, то все, что находится внутри одинарных кавычек воспринимается буквально (даже $).
Затем идет поиск двойных кавычек. Двойные кавычки внутри двойных игнорируются. То, что внутри двойных кавычек после обратного слеша воспринимается буквально.

Внутри двойных кавычек и вне любых кавычек заменяем переменные с долларами на их значение (но это неверно для $ в одинарных кавычках которые находятся вне двойных кавычек).

Название переменной -- это строка между $ и пробелом, то есть строка «$x y», где x=5, должна замениться на строку «5 y».
Если переменная, которая встречается в строке, отсутствует в таблице переменных, то она заменяется на пустую строку.

Затем нужно редуцировать лишние пробелы: между опциями, stdin, командами и чертой пайпа | должно быть по одному пробелу; внутри двойных и одинарных кавычек пробелы сохраняются.

ReadVariable:
Проверяет, задаёт ли пользователь во входной строке значение новой переменной, и если задаёт, то производится запись или изменение переменной окружения в таблице *EnvironmentVariable.*

Если входная строка имеет вид spaceLessString=”anything” или spaceLessString=’anything’, то делаем запись в таблице (ключ -- spaceLessString, значение -- anything).
Если входная строка имеет вид spaceLessString=spaceLessSecondString, то делаем запись в таблице (ключ -- spaceLessString, значение -- spaceLessSecondString).

Далее методом ParseIntoCommandsAndOptionsAndStdins строка, которую мы в общем случае рассматриваем в виде
“команда1 arg1 … argN | команда2 arg1 … argM | … | командаK arg1 … argS” (пайплайн может отсутствовать, то есть команда может быть одна)
преобразуется в список кортежей (название команды, список аргументов), разделяя название команды, списки аргументов через черту пайплайна (в вырожденном случае список кортежей из одного элемента -- отсутствие пайплайна).

При возникновении ошибок (например, отсутствует закрывающая кавычка), парсинг прекращается и ошибка выводится на экран.

Класс Creator
Класс, в котором результат парсинга (название команды, список опций)[*] преобразуется в конкретные команды.

Важно:
“Список со строками названий внутренних команд *ListOfCommandsName* необходимо обновлять вручную после добавления/удаления внутренних команд, которые наследуются от абстрактного класса Command. Во время создания команд, введенных пользователем, проверяется наличие команды в этом списке“

CreateCommandsList вызывается из *CLIKernel* для создания списка экземпляров команд для пайплайна (вырожденный случай -- одна команда без пайплайна также должна обрабатываться. То есть создается список из одного элемента), принимая на вход список кортежей (название команды, список опций).

CreateCommand:
Создает экземпляры команд.
Далее проверяется список названий внутренних команд *ListOfCommandsName*:
если введенная пользователем команда внутренняя

- то создается экземпляр класса данной внутренней команды по строке с именем команды и заполняется поле списка *arguments*.
- иначе создается экземпляр класса внешней команды для попытки вызова внешней программы и заполняется поле списка *arguments* экземпляра, имя потенциальной внешней программы *externalProgramName*.

Класс Activator
*И*з *CLIKernel* вызывается метод *HandlePipeline,* в который передается список команд. Он по очереди вызывает метод *Process* конкретных команд, организуя пайплайн следующим образом.

Результат выполненной команды записывается в поле *previousResult (*изначально поле инициализировано состоянием отсутствием результата -- null*).* Если команда ничего не выводит, то *previousResult* нужно перевести в состояние отсутствия результата.

Очередная команда в пайплайне выполняется следующим образом:
В метод *Command.Process(previousResultForStdin : string, isFirstInPipe : bool)* передается *previousResult* и информация *isFirstInPipe*. Аргумент метода *previousResult* предназначен для stdin выполняемой команды (подробнее смотри описание *Command.Process*).

После каждого выполнения команды изменяется код возврата *returnCode (*приватно изменяется, а для получения значения имеет публичный доступ*)*:
0 в случае успешного выполнения команды
целое число 1 - 255 в случае ошибки.

В ходе выполнения команд обрабатываются ошибки:
При возникновении ошибки, *previousResult* переводится в состояние отсутствия результата. После обработки ошибки, произведенной для поддержания работоспособности программы, вызывается метод *StderrStdoutHandler.PrintStderr,* в который передается описание ошибки.
Обработка переходит к следующей команде пайплайна.

После выполнения последней команды из пайплайна результат последней команды передается в метод *StderrStdoutHandler.PrintLastCommandStdout* для вывода результата на экран пользователя.

StderrStdoutHandler

В будущем в этом классе можно реализовать перенаправление stderr и stdout потоков.

PrintLastCommandStdout -- выводит результат последней команды пайплайна на экран пользователя
PrintStderr -- выводит информацию об ошибке выполнения команды на экран пользователя.

Абстрактный класс Command
Абстрактный класс, который наследуется всеми классами команд. Содержит:
метод *Process*, который вызывает исполнение конкретной реализации команды;
поле *arguments,* которое представляет список аргументов.

Stdout предыдущей команды хранится в аргументе *previousResultForStdin* метода *Process.*

Команда внутри себя понимает, задал ли пользователь с помощью аргументов явный ввод в stdin из клавиатуры**.** Например, в командах cat, head аргумент “дефис” вызывает явный ввод в stdin из клавиатуры (если команда стоит на первом месте пайпа, либо эта команда единственна): “cat -” тут ввод в stdin будет производится с клавиатуры.

Также команда внутри себя должна понимать открыт или закрыт stdin команды в пайплайне. Примеры открытого/закрытого stdin для конкретных команд (wc, cat):
а)

- В аргументах команды может присутствовать данные для ввода, а stdin команды в пайпе не открыт. Например, echo 5 | wc “input.txt”. То есть пайплайн echo 5 | wc “input.txt” выведет обработку input.txt, а stdout команды echo не будет использован.
- Если же данные для ввода команде отсутствуют, и она следует за другой командой в пайплайне, то в stdin данной команды поступает stdout предыдущей (то есть в этой случае необязательно открывать stdin через аргумент). Например, “echo 5 | cat” выведет “5”. Случай с null значением previousResultForStdin должен корректно обрабатываться командой (например, cd / | wc должно вывести 0 0 0, хоть cd не имеет вывода)

- Причем, также имеет место быть следующий случай. Когда не у первой команды, имеющей входные данные в аргументе, в пайпе указано, что stdin открыт (например “echo 5 | cat a.txt -”), эта команда должна обработать входные данные из аргумента (в данном случае файл a.txt), а затем обработать stdout предыдущей команды. Получается, что stdout команды = вывод для аргументов + вывод для stdin. То есть, “echo 5 | cat a.txt -” выведет содержимое a.txt и затем “5”. Этот пункт верен и в случае, когда предыдущая команда ничего не возвращает. Например, “cd / | cat a.txt -” выведет только содержимое a.txt.

б)

- Если команда первая в пайпе (вырожденный случай -- команда одна), то открытие stdin (например, “cat -”) означает ввод в stdin с клавиатуры. Также

должен обрабатываться аргумент с входными данными, если он имеется (например, “cat a.txt -” сначала выведется содержимое a.txt, затем будет выводится на экран ввод через клавиатуру).

- Замечание: открытие stdin может быть ДО и ПОСЛЕ аргумента с входными данными. Например:

“echo 5 | cat a.txt -” выведет содержимое a.txt, затем 5
“echo 5 | cat - a.txt” выведет 5, затем содержимое a.txt.

Конкретные реализации команд:
CAT, ECHO, WC, PWD, EXIT;

ExternalCommand - попытка вызвать внешнюю программу по строке *ExternalProgramName*, создавая процесс, в случае, если ни одна из уже существующих команд не удовлетворяет вводу пользователя.

ExternalProgramName -- имя потенциальной внешней программмы.

Во внешней программе получаем информацию о возникших в ходе выполнение ошибках и информируем об этом *Activator.*