ConsoleUtils

<<utility>> + ConCursorMove

+ ANSI_CURSOR_LEFT: String = "\u001B[D" {ReadOnly}
+ ANSI_CURSOR_RIGHT: String = "\u001B[C" {ReadOnly}
+ ANSI_CURSOR_UP: String = "\u001B[A" {ReadOnly}
+ ANSI_CURSOR_DOWN: String = "\u001B[B" {ReadOnly}

+ Left(steps: int)
+ Right(steps: int)
+ Up(steps: int)
+ Down(steps: int)

+ ConProgressBar

- MAX_PERCENTAGE: int = 100 {ReadOnly}
- MIN_PERCENTAGE: int = 0 {ReadOnly}
- Completed_block: char = ' # ' {ReadOnly}
- Empty_block: char = ' . ' {ReadOnly}

- Percentage: int

+ ConProgressBar(name: String, width: int, startPercentage: int)
+ ConProgressBar(name: String, width: int, startPercentage: int background: ConColor, foreground: ConColor)
+ getPercentage(): int
+ setPercentage(perc: int)
+ printControl()
+ printControlWithoutColors()

<<enumeration>> + ConColor

ANSI_RESET: String = "\u001B[30m"
ANSI_FOREGROUND_RED: String = "\u001B[31m"
ANSI_FOREGROUND_GREEN: String = "\u001B[32m"
ANSI_FOREGROUND_YELLOW: String = "\u001B[33m"
ANSI_FOREGROUND_BLUE: String = "\u001B[34m"
ANSI_FOREGROUND_PURPLE: String = "\u001B[35m"
ANSI_FOREGROUND_CYAN: String = "\u001B[36m"
ANSI_FOREGROUND_WHITE: String = "\u001B[37m"
ANSI_BACKGROUND_BLACK: String = "\u001B[40m"
ANSI_BACKGROUND_RED: String = "\u001B[41m"
ANSI_BACKGROUND_GREEN: String = "\u001B[42m"
ANSI_BACKGROUND_YELLOW: String = "\u001B[43m"
ANSI_BACKGROUND_BLUE: String = "\u001B[44m"
ANSI_BACKGROUND_PURPLE: String = "\u001B[45m"
ANSI_BACKGROUND_CYAN: String = "\u001B[46m"
ANSI_BACKGROUND_WHITE: String = "\u001B[47m"

- ANSI_Color: String

- ConColor(ansi_color: String)
+ getANSI_Color(): String
+ setANSI_Color(ansi_color: String)

uses for coloring

+ ConControl

- MAX_WIDTH: int = 80 {ReadOnly}
- MIN_WIDTH: int = 10 {ReadOnly}

- Width: int
- Name: String
- BackgroundColor: ConColor
- ForegroundColor: ConColor

+ ConControl(name: String, width: int)
+ ConControl(name: String, width: int, background: ConColor, foreground: ConColor)
+ getName(): String+ getWidth(): int
+ getBackgroundColor(): ConColor
+ getBackgroundColorANSI(): String
+ getForegroundColor(): ConColor
+ getForegroundColorANSI(): String
+ setName(name: String)
+ setWidth(width: int)
+ setBackgroundColor(ansi_color: ConColor)
+ setForegroundColor(ansi_color: ConColor)
+ printControl()
+ printControlWithoutColors()

ExtendedMath

<<enumeration>> + ComplexNumberForm

Rectangular
Polar
Exponential

uses for classification

+ ComplexNumber

- initial_form: ComplexNumberForm
- a_coef: double
- b_coef: double
- r_coef: double
- fi_angl_deg: double
- fi_angl_rad: double

+ ComplexNumber(form: ComplexNumberForm, coef1: double, coef2: double)
+ Re(): double
+ Im(): double
+ AbsValue(): double
+ Arg_deg(): double
+ Arg_rad(): double
+ ToString(): String
+ ToString(form: ComplexNumberForf): String
+ equals(obj: Object): boolean

- CalculateAngleInRadians(coef_re: double, coef_im: double)
+ getAllForms(num: ComplexNumber): String

<<utility>> + HyperbolicFunction

+ sh(rad: double): double
+ ch(rad: double): double
+ th(rad: double): double
+ cth(rad: double): double

uses for calculations

Uses as the main data type that all functions work with

<<utility>> + ComplexNumberOperation

+ add(num1: ComplexNumber, num2: ComplexNumber): ComplexNumber
+ sub(num1: ComplexNumber, num2: ComplexNumber): ComplexNumber
+ mul(num1: ComplexNumber, num2: ComplexNumber): ComplexNumber
+ div(num1: ComplexNumber, num2: ComplexNumber): ComplexNumber
+ sin(num: ComplexNumber): ComplexNumber
+ cos(num: ComplexNumber): ComplexNumber
+ tg(num: ComplexNumber): ComplexNumber
+ ctg(num: ComplexNumber): ComplexNumber
+ pown(num: ComplexNumber, n: int): ComplexNumber
+ sqrtn(num: ComplexNumber, n: int): ComplexNumber
+ getComplexConjugate(num: ComplexNumber): ComplexNumber

Uses to display the file processing process

ProgramFunctionality

Algorithms

Uses algorithms to work

<<utility>> + ComNumAlgorithm

+ Tokenization(str: String, memory: Memory): ArrayList<Token>
+ ShuntingYardAlgorithm(inputList: ArrayList<Token>): ArrayList<Token>
+ CalculateExpressionInRPN(list:ArrayList<Token>, memory: Memory)
- getVariableOrLiteralValueFromToken(token: Token, memory: Memory): ComplexNumber
- getIntegerFromComplexNumber(cn: ComplexNumber, funcName: String): int
- getDoubleFromComplexNumber(cn: ComplexNumber, funcName: String): double

+ Token

- Content: String
- Type: TokenType

+ Token(type: TokenType, content: String)
+ getContent(): String
+ getType(): TokenType
+ toString(): String
+ getComplexNumberFromToken(): ComplexNumber
+ isLeftAssociativeOperation(): boolean
+ isVariable(): boolean
+ isLiteral(): boolean
+ isOperation(): boolean
+ isFunction(): boolean
+ isOpeningBracket(): boolean
+ isClosingBracket(): boolean
+ isComma(): boolean

Uses as the main data type that all algorithms work with

Uses it to convert some types of tokens into complex numbers

uses for classification

<<enumeration>> + TokenType

Unnecessary: String = "[\s\\n\\t\\r\\v]+", byte = 0
ComLiteralRectangular: String = "-?\\d+\\.\\d+[\\+\\-]\\d+\\.\\d+*i", byte = 0
ComLiteralPolar1: String = "\\d+\\.\\d+*\\(cos\\(\\d+\\.\\d+\\)\\+i*sin\\(\\d+\\.\\d+\\)\\)", byte = 0
ComLiteralPolar2: String = "\\d+\\.\\d+*\\(cos\\(-\\d+\\.\\d+\\)\\+i*sin\\(-\\d+\\.\\d+\\)\\)", byte = 0
ComLiteralExponential1: String = "\\d+\\.\\d+*exp\\(i*\\d+\\.\\d+\\)", byte = 0
ComLiteralExponential2: String = "\\d+\\.\\d+*exp\\(i*\\(-\\d+\\.\\d+\\)\\)", byte = 0
NumLiteral1: String = "\\d+\\.\\d+", byte = 0
NumLiteral2: String = "-?\\d+\\.\\d+", byte = 0
Sin: String = "sin", byte = 0
Cos: String = "cos", byte = 0
Tg: String = "tg", byte = 0
Ctg: String = "ctg", byte = 0
Pown: String = "pown", byte = 0
PrintSqrtn: String = "printSqrtn", byte = 0
Print: String"print", byte = 0
PrintAllForms: String = "paf", byte = 0
Add: String = "\\+", byte = 3
Sub: String = "-", byte = 3
Mul: String = "*", byte = 4
Div: String = "\\/", byte = 4
ComplexConjugate: String = "\\$", byte = 5
Assignment: String = "=", byte = 2
OpeningBracket: String = "\\(", byte = 1
ClosingBracket: String = "\\)", byte = 1
Comma: String = ",", byte = 0
Variable: String = "[a-zA-Z]+\\w*", byte = 0
Comment: String = "#", byte = 0
Unknown: String = ".+", byte = 0

- RegEx: String
- Priority: byte

- TokenType(regex: String, priority: byte)
+ getRexEx(): String
+ getPriority(): byte

+ ComNumProgram

- memory: Memory
- fileManager: FileManager
- arguments: String[]
- progressBar: ConProgressBar

+ ComNumProgram(args: String[])
+ Run()
- tryToExecuteCodeLine(line: String, lineNumber:int)

Starts the program

+ Program

+ main(args:String[])

Uses it to organize work with files

1

+ FileManager

- currentFile: File

+ FileManager(filePath: String)
+ getCurrentFile(): File
+ setFilePath(filePath: String)
+ getTextFromCurrentFile(): ArrayList<String>
+ CheckFileExtension(): boolean

Uses this to store variables, input program code text, console output text, and error messages

+ Memory

- VariableList: ArrayList<Variable>
- ProgramCode: ArrayList<String>
- ConsoleOutput: StringBuilder
- ErrorLog: StringBuilder

+ Memory()
+ setProgramCode(code:ArrayList<String>)
+ setNewVariableValue(varName: String, value:ComplexNumber)
+ getProgramCode(): ArrayList<String>
+ getValueOfVariable(name:String): ComplexNumber
+ getConsoleOutput(): String
+ getErrorLog(): String
+ containsVariable(name: String): boolean
+ createNewVariable(var: Variable)
+ clearConsoleOutput()
+ clearErrorLog()
+ addTextToConsoleOutput(text:String)
+ addTextToErrorLog(text:String)

1

Uses a type to return the values of variables

Variables

<<interface>> + Variable

T

+ getValue(): T
+ setValue(value: T)
+ getName(): String
+ getType(): String

{T -> ComplexNumber}

+ ComplexNumberVar

- Name: String {ReadOnly}
- Value: ComplexNumber

+ ComplexNumberVar(name: String, value: ComplexNumber)
+ getValue(): ComplexNumber
+ setValue(value: ComplexNumber)
+ getName(): String
+ getType(): String

Stores variables

Uses the type as the value of a variable