TKOM - Dokumentacja wstępna

Dawid Kaszyński

Wymagania funkcjonalne

- Typowanie silne dynamiczne
- Deklarowanie zmiennych słowem kluczowym *var*. Dostępne typy danych to:
  - int
  - float
  - string
  - słownik
- Zmienna każdego z typów typu może mieć konkretną wartość albo wartość *null*.
- Istnieje operator *?*, który po dopisaniu na końcu zmiennej sprawia, że jeśli ma ona wartość null to zostanie wyrzucony wyjątek.
- Pętla *while*
- Instrukcje warunkowe *if*, *else*
- Definiowane funkcji z listą argumentów z użyciem słowa kluczowego *def*
- Wbudowana funkcja *print* służąca do wypisania tekstu na standardowe wyjście
- Wbudowana funkcja input wczytająca tekst od użytkownika ze standardowego wejścia
- Obsługa komentarzy, które są oznaczane znakiem #.
- Rzucanie i przechwytywanie wyjątków

Przykłady użycia

Deklaracje do zmiennych oraz przypisanie

```
var x = 3 * 3 + 2 - 1;
var y = 4;

var z;
z = x + y;

var x = 1.0;
var y = x + 2.0;

var a = "Ala" + " " + "ma" + " " + "kota";
```

Użycie słownika

```
var mp = {"a": 1, "b": 2, 3: "d"};
```

```
mp["a"] = 3;
var x = mp["a"];
var y = mp[3];
```

Użycia operatora *?* rzucającego wyjątek, gdy zmienna ma wartośc null

```
var a = null;

try {
    var b = a?;
} catch {
    print(“A ma wartosc null”);
}
```

Instrukcja *if* z wieloma warunkami

```
var a = 12;
var b = 3.5;

if ((a == 2 && a == 3) || a > b) {
      print("a");
} else {
      print("b");
}
```

Deklaracja i wywołanie funkcji, użycie instrukcji *if*, *else* oraz *while*

```
def fib(n) {
      var a = 0;
      var b = 1;

      if (n == 0) {
            return a;
      } else if (n == 1) {
            return b;
      }

      var i = 2;
      while (i <= n) {
            var tmp = b;
```

```
            b = a + b;
            a  = tmp;
      }

      return b;
}

var x = fib(10);
print(x);
```

Brak danego klucza w słowniku

```
var x = {};

try {
   var y = x["a"];
} catch (e) {
   print("Brak klucza w slowniku");
   print(e);
}
```

Rzucanie wyjątku i wczytywanie ze standardowego wejścia

```
print("Podaj wiek");
age = input();
if (age <= 0) {
   throw "invalid age";
}
```

Konkatenacja i mnożenie różnych typów

```
var a = "1" + "1";  # "11"
var b = "1" * 3;  # Błąd - niekompatybilne typy
var c = 5 + 5.5;  # 10.5
```

Zapis gramatyki

```
program = {variableDeclaration | functionDefinition};
```

```
statement = ifStatement |
            whileStatement |
            variableDeclaration |
            assignment |
            functionCall |
            returnStatement |
            tryCatchStatement
            throwStatement;

functionDefinition = "def" identifier parameterList statementBlock;

parameterList = "(" [identifier, {"," identifier}] ")";

statementBlock = "{" {statement} "}";

ifStatement = "if" "(" expression ")" statementBlock ["else"
statementBlock];

whileStatement = "while" "(" expression ")" statementBlock;

variableDeclaration = "var" identifier ["=" expression] ";";

assignment = identifier "=" expression |
             identifier "[" dictionaryKey "]" "=" expression ";";

dictionaryKey = identifier | literal;

functionCall = identifier "(" [expression, {"," expression}] ")" ";";

returnStatement = "return" expression ";";

tryCatchStatement = "try" statementBlock "catch" ["(" identifier ")"]
statementBlock;

throwStatement = "throw" expression ";";

expression = multiplicativeExpression {additiveOperator
multiplicativeExpression};

multiplicativeExpression = orExpression {multiplicativeOperator
orExpression};

orExpression = andExpression {orOperator andExpression};

andExpression = relationExpression {andOperator relationExpression};

relationExpression = equalExpression {relationOperator equalExpression};
```

```
equalExpression = possibleNegatedSingleExpression {equalExpression
possibleNegatedSingleExpression};

possibleNegatedSingleExpression = singleExpression | ("!" singleExpression);

singleExpression = identifier |
                   literal |
                   "(" expression ")" |
                   identifier "[" dictionaryKey "]" |
                   functionCall |
                   nonNullableExpression;

nonNullableExpression = singleExpression "?";

equalOperator = "==" |
                "!=";

relationOperator = "<" |
                   ">" |
                   "<=" |
                   ">=";

orOperator = "||";

andOperator = "&&";

additiveOperator = "+" |
                   "-";

multiplicativeOperator = "*" |
                         "/"
                         "%";

identifier = letter {letter | digit | "_"};

literal = stringLiteral |
          numberLiteral |
          dictionaryLiteral |
          nullLiteral;

letter = "a".."z" | "A".."Z";


nonZeroDigit = "1" .. "9";

digit = "0" | nonZeroDigit;

stringLiteral = "\"" .. "\"";
```

```
numberLiteral = nonZeroDigit {digit} ["." digit {digit}];

dictionaryLiteral = "{" [singleExpression ":" singleExpression {","
singleExpression ":" singleExpression }] "}";

nullLiteral = "null";
```

Realizacja techniczna

- Implementacja w Javie
- Budowanie projektu z gradle
- Testy jednostkowe i integracyjne JUnit