Język obiektowy

Podstawowe elementy

1. **Int** - liczba całkowita
   Object - na wzór klasy z języków programowania jak Python czy C++
   [ ] - lista
   None - wykorzystywany np. w momencie, gdy chcemy zadeklarować atrybut, a nie wiemy co będzie się w nim zawierało

2. **Stałe znakowe**: "txt", 'txt'

3. **Operatory arytmetyczne**: $+ \quad - \quad * \quad /$
   Operatory logiczne: $< \quad > \quad <= \quad >= \quad == \quad != \quad || \quad \&\& \quad !$
   Operator przypisania: $=$

4. **Komentarz**: $--$
   Wypisanie danych: out<<
   Wczytanie danych: in>>

5. **Wyrażenia**

```
a = 3;
b = 3 * a;
c = silnia(3);
d = Obiekt(3);
out<< "tekst";
silnia(5);
```

   Każde wyrażenie musi być zakończone średnikiem.

Język obiektowy

Złożone elementy

1. Intrukcja warunkowa **when**

```
when <warunek> {
    ...
} else when <warunek> {
    ...
} else {
    ...
}
```

   Kod umieszczony wewnątrz { } jest wykonany w przypadku spełnienia warunku lub dla intrukcji else w przypadku, gdy żaden z wcześniejszych warunków nie został spełniony.

2. Pętle **loop**

```
loop(<start>; <koniec>; <krok>) {
    ...
}

loop(<zmienna>:<lista>) {
    ...
}
```

   Pierwsza pętla w działa dopóki jest spełniony warunek końca.
   Druga pętla w każdej kolejnej iteracji bierze kolejny element z listy.

3. Metody wbudowane w liście:

   (a) append(element) - dodaje element na koniec listy

   (b) remove(index) - usuwa element znajdujący się na pozycji będącej argumentem metody

   (c) get(index) - zwraca element znajdujący się na pozycji będącej argumentem metody

4. Definiowanie metod

```
fun nazwaMetody([<argumenty>])[extends <NazwaObiektu>] {
    ...
}
```

Argumenty nie są obligatoryjne, występują 3 typy argumentów:

```
arg1 IN, arg2 OUT, arg3 IN OUT
```

Słowo kluczowe **IN** oznacza, iż argument jest tylko do odczytu, **OUT** tylko do zapisu. W przypadku, gdy chcemy przekazać zmienną do funkcji i ją w niej edytować stosujemy **IN OUT**. Domyślnie wszystkie argumenty są traktowane jakby występowało po nich słowo kluczowe **IN**.

Gdy nie chcemy definiować metod wewnątrz obiektu stosujemy zwrot **extends nazwaObiektuDlaKtóregoDefiniujemyMetodę**

5. Obiekt **object**

```
object NazwaObiektu([<ObiektBazowy>]){
    attr [{get; set;}] = int | object | [] | none;

    -- Konstruktor obiektu z wywołanym domyślnym
    -- konstruktorem dla obiektu bazowego
    NazwaObiektu() {}

    -- Wykorzystanie słowa kluczowego this
    NazwaObiektu(attr) {
        this.attr = attr;
    }

    -- Konstruktor obiektu z jawnie wywołanym
    -- konstruktorem dla obiektu bazowego
    NazwaObiektu([<argumenty>]):ObiektBazowy([<argumenty>]) {}

    -- Przeciążenie operatora
    operator == (lhs, rhs, result OUT) {
        ...
    }

}
```

Język obiektowy

Obiekt jest najbardziej złożonym elementem języka.

- Może się zdarzyć, że argument metody będzie miał taką samą nazwę jak atrybut obiektu. Korzystamy wtedy ze słowa kluczowego **this** dla rozróżnienia.

- Przez konstruktor domyślny będę rozumiał kontruktor bezargumentowy.
Aby wywołać konstuktor domyślny musi on zostać zaimplementowany.
Możliwe jest dziedziczenie jednobazowe poprzez podanie nazwy obiektu wewnątrz ( ).
W przypadku, gdy wywołujemy konstruktor dla obiektu i nie podamy konstruktora obiektu bazowego zostanie wywołany konstruktor domyślny dla owego obiektu.

- Atrybuty są publiczne i podczas deklaracji muszą być zainicjowane. Dodatkową funkcjonalnością jest możliwość skorzystania z wbudowanych getterów i setterów. Odwołując się do obiektu zanim otrzymamy wartość zostanie wywołana metoda, która tę wartość zwróci.

```
object Obiekt(){
    x {get; set;} = 5;
}
zmienna = Obiekt();
-- Odwołujemy się bezpośrednio do x,
-- lecz wykonana zostanie metoda get dla atrybutu x.
out>> zmienna.x;
```

- Wypisywanie obiektu z wykorzystaniem **out**<<
Domyślnie wypisana zostanie nazwa obiektu. Można jednak napisać własną metodę.

```
fun _str_(tekst OUT) extends NazwaObiektu {
    ...
}
```

Tworzenie programu

1. Interpreter będzie poszukiwał funkcji main, którą wywoła

```
fun main() {}
```

2. Poza funkcją możemy tworzyć inne funkcje i obiekty. Należy pamiętać jednak, że odwołanie się do obiektu/metody może nastąpić dopiero po ówczesnej deklaracji.

```
fun wypisz(tekst) {
    out<< tekst;
}

fun main() {
    -- Zadziała
    wypisz("Hello, world");

    -- Nie zadziała
    a = 6;
    podziel(a, 2);
}

fun podziel(dzielna IN OUT, dzielnik) {
    dzielna = dzielna / dzielnik;
}
```

3. Tak jak widać w przykładach wyrażenia muszą być zakończone średnikiem. Maksymalną długość identyfikatorów ustalam na 30 znaków.

Przykład

```
object Vehicle {
    id = None;
    company_name = None;

    fun Vehicle() {}
    fun Vehicle(id, cn) {
        this.id = id;
        company_name = cn;
    }

}

object Car {
    max_speed {get; set;} = 100;

    fun Car(){}
    fun Car(ms, id, cn):Vehicle(id, cn) {
        max_speed = ms;
    }
}

fun _str_(txt OUT) extends Car {
    txt = id;
}

fun main() {
    cars = []
    for (i = 1; i < 5; i = i + 1) {
        cars.append(Car(100 + i, i, "Company"));
    }

    for (car : cars) {
        out<< car;
    }

}
```

Język obiektowy

```
                                    (*Notacja EBNF*)
program = { obiekt | funkcja | komentarz }, funkcja_startowa, { komentarz };

obiekt = słowo_kluczowe_object, identyfikator, [ lewy_nawias, identyfikator,
    prawy_nawias ], lewy_nawias_klamrowy,{ atrybut, średnik }, { { funkcja }
    | { przeciążanie_operatora } }, prawy_nawias_klamrowy;

funkcja = słowo_kluczowe_fun, wywołanie_metody, [ dwukropek, wywołanie_metody |
    słowo_kluczowe_extends, identyfikator ] lewy_nawias_klamrowy,
{ wyrażenie }, prawy_nawias_klamrowy;

funkcja_startowa = słowo_kluczowe_fun, słowo_kluczowe_main, lewy_nawias,
    [ lista_argumentów ], prawy_nawias, ciało_funkcji;

atrybut = identyfikator, [ lewy_nawias_klamrowy, { identyfikator, średnik },
    prawy_nawias_klamrowy ], operator_przypisania, ( int | stała_znakowa |
    słowo_kluczowe_none | operator_indeksu | wywołanie );

przeciążanie_operatora = słowo_kluczowe_operator, operator, lewy_nawias,
    [ lista_argumentów ], prawy_nawias, ciało_funkcji;

wywołanie = wywołanie_metody | wywołanie_metody_jako_składowej;
wywołanie_metody_jako_składowej = identyfikator, kropka, wywołanie_metody;
wywołanie_metody = identyfikator, lewy_nawias, [ lista_argumentów ], prawy_nawias;

instrukcja_warunkowa = słowo_kluczowe_when, warunek, ciało_funkcji,
    [ słowo_kluczowe_else, [ słowo_kluczowe_when, warunek ], ciało_funkcji ];

pętla = słowo_kluczowe_loop, lewy_nawias, ( przypisanie, średnik, warunek,
    średnik, krok | identyfikator, dwukropek, identyfikator ),
    prawy_nawias, ciało_funkcji;

ciało_funkcji =  lewy_nawias_klamrowy, { wyrażenie }, prawy_nawias_klamrowy;

wyrażenie = ( przypisanie, średnik ) | ( wywołanie, średnik ) |
    instrukcja_warunkowa | pętla | komentarz | ( obsługa_wejścia_wyjścia, średnik );

przypisanie = [ słowo_kluczowe_this ], identyfikator, operator_przypisania,
    ( stała_znakowa | int | słowo_kluczowe_none | operator_indeksu |
    wyrażenie_arytmetyczne | wywołanie_metody );
```

Język obiektowy

```
(*
Warunek pierwszeństwo:
1. operator_negacji
2. operator_relacyjny
3. operator_logiczny
*)

warunek = składowa_warunku, { operator_logiczny, składowa_warunku };
składowa_warunku = czynnik, { operator_relacyjny, czynnik };
czynnik = int | identyfikator | stała_znakowa | [ operator_negacji ],
    lewy_nawias, warunek, prawy_nawias | wyrażenie_arytmetyczne;


obsługa_wejścia_wyjścia = obsługa_wejścia | obsługa_wyjścia;
obsługa_wejścia = "in>>", identyfikator, średnik;
obługa_wyjścia = "out<<", ( identyfikator | stała_znakowa ), średnik;

krok = identyfikator, operator_przypisania, ( wyrażenie_arytmetyczne | wywołanie )

wyrażenie_arytmetyczne = składowa, {operator_dodawania, składowa};
składowa = element, {operator_mnożenia, element};
element = int |  identyfikator | lewy_nawias, wyrażenie_arytmetyczne, prawy_nawias;

(*
Wyrażenie_arytmetyczne pierwszeństwo:
1. operator_mnożenia
2. operator_dodawania
*)

identyfikator = [:alpha:], [:word:]*;

lista_argumentów = { argument_z_przecinkiem }, argument;
argument = identyfikator, [ słowo_kluczowe_in ], [ słowo_kluczowe_out ];
argument_z_przecinkiem = argument, ",";

operator = operator_matematyczny | operator_logiczny | operator_przypisania |
    operator_negacji | operator_relacyjny;

operator_przypisania = "=";
operator_negacji = "!";
operator_relacyjny = "<=" | ">=" | "==" | "!=" | "<" | ">";
operator_logiczny = "||" | "&&";
```

Język obiektowy

```
operator_matematyczny = operator_dodawania | operator_mnożenia;
operator_dodawania = "+" | "-";
operator_mnożenia = "*" | "/";
operator_indeksu = "[]";

słowo_kluczowe_else = "else";
słowo_kluczowe_when = "when";
słowo_kluczowe_loop = "loop";
słowo_kluczowe_object = "object";
słowo_kluczowe_fun = "fun";
słowo_kluczowe_operator = "operator";
słowo_kluczowe_main = "main";
słowo_kluczowe_none = "none";
słowo_kluczowe_extends = "extends";
słowo_kluczowe_in = "in";
słowo_kluczowe_out = "out";
słowo_kluczowe_this = "this";

dwukropek = ":";
średnik = ";";
kropka = "."
lewy_nawias = "(";
prawy_nawias = ")";
lewy_nawias_klamrowy= "{";
prawy_nawias_klamrowy = "}";

int  = "0" | ([ "-" ], cyfra_bez_zera, { cyfra });
stała_znakowa = ( "\"", ciag_znakow, "\"") |  ("\'", ciag_znakow, "\'" );

komentarz = "#", ciag_znakow;

cyfra_bez_zera = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
cyfra = [:digit:];

ciag_znakow = [:alnum:]* | [:punct:]* | [:blank:]*;

[:alnum:] = [a-zA-Z0-9];
[:alpha:] = [a-zA-Z];
[:digit:] = [0-9];
[:word:] = [a-zA-Z0-9];
[:punct:] = [ ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~ ];
[:blank:] = space or tab;
```

Język obiektowy