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**: $+ \quad - \quad * \quad /$

4. **Operatory logiczne**: $< \quad > \quad <= \quad >= \quad == \quad != \quad || \quad \&\& \quad !$

5. **Operator przypisania:** $=$

6. **Komentarz**:

7. **Wypisanie danych**: out<<

8. **Wczytanie danych**: in>>

9. **Wyrażenia**

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

   Każde wyrażenie musi być zakończone średnikiem.
   Język jest dynamicznie typowany.

Język obiektowy

Złożone elementy

1. Intrukcja warunkowa **when**

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

when i > 0 {
    out << "Wieksza od zera";
} else when i < 0 {
    out << "Mniejsza od zera";
} else {
    out << "Równa zero";
}
```

   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>) {
    ...
}

loop( i = 0; i < 5; i = i + 1){...}

zawodnicy = [];
loop(zawodnik : zawodnicy){...}
```

Pierwsza pętla 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, result out) - zwraca element znajdujący się na pozycji będącej argumentem metody

   Przykład

```
liczby = [];
liczby.append(2);
liczby.append(3);
liczby.remove(0);
liczba = none;
liczby.get(0, liczba)
```

4. Definiowanie metod

```
nazwaMetody([<argumenty>])[extends <NazwaObiektu>] {...}
dodaj(listaZakupów OUT, produkt IN)extends Klient{...}
```

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**

Język obiektowy

5. Obiekt **object**

```
NazwaObiektu [extends NazwaObiektuBazowego]{
    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) {
        ...
    }

}
```

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 użycie słowa kluczowego **extends** i podania nazwy obiektu bazowego. 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.

```
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.

```
_str_(tekst OUT) extends NazwaObiektu {
    ...
}
_str_(tekst OUT) extends Klient{
    tekst = this.id;
}
```

Język obiektowy

Tworzenie programu

1. Interpreter będzie poszukiwał funkcji main, którą wywoła

```
main() {}
```

2. Długość identyfikatorów ustalam na 30 znaków.

Przykład

```
Vehicle {
    id = None;
    company_name = None;
    Vehicle() {}
    Vehicle(id, cn) {
        this.id = id;
        company_name = cn;
    }
}
Car extends Vehicle{
    max_speed {get; set;} = 100;

    Car(){}
    Car(ms, id, cn):Vehicle(id, cn) {
        max_speed = ms;
    }
}
_str_(txt OUT) extends Car {
    txt = id;
}
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 };

obiekt = identyfikator, [ słowo_kluczowe_extends, identyfikator ], blok;

blok = lewy_nawias_klamrowy, { atrybut | metoda | przeciążanie_operatora },
    prawy_nawias_klamrowy;

metoda = identyfikator, lewy_nawias, [ lista_argumentów ], prawy_nawias, [ dwukropek,
    identyfikator, lewy_nawias, [ lista_argumentów ], prawy_nawias ], ciało_metody;

ciało_metody = lewy_nawias_klamrowy, { wyrażenie }, prawy_nawias_klamrowy;

atrybut = identyfikator, [ lewy_nawias_klamrowy, [słowo_kluczowe_get, średnik],
    [słowo_kluczowe_set, średnik], prawy_nawias_klamrowy ], operator_przypisania,
    (  stała_znakowa | słowo_kluczowe_none | operator_indeksu |
    wyrażenie_arytmetyczne ), średnik;

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

funkcja = identyfikator, lewy_nawias, [ lista_argumentów ], prawy_nawias
    [słowo_kluczowe_extends, identyfikator ], ciało_funkcji;

ciało_funkcji = lewy_nawias_klamrowy, { wyrażenie }, prawy_nawias_klamrowy;

wywołanie_funkcji_metody = wywołanie_metody | wywołanie_funkcji;
wywołanie_metody = identyfikator, kropka, identyfikator, lewy_nawias,
    [ lista_argumentów ], prawy_nawias, średnik;
wywołanie_funkcji = identyfikator, lewy_nawias, [ lista_argumentów ],
    prawy_nawias, średnik;

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

pętla = słowo_kluczowe_loop, lewy_nawias, ( przypisanie, średnik, warunek,
    średnik, krok | identyfikator, dwukropek, identyfikator ),
```

```
    prawy_nawias, ciało_funkcji;

wyrażenie = przypisanie | wywołanie_funkcji_metody | instrukcja_warunkowa | pętla |
    obsługa_wejścia_wyjścia;

przypisanie = [ słowo_kluczowe_this, kropka ], identyfikator, operator_przypisania,
    wyrażenie_arytmetyczne;

obsługa_wejścia_wyjścia = obsługa_wejścia | obsługa_wyjścia;
obsługa_wejścia = słowo_kluczowe_in, operator_wczytywania, (wyrażenie_arytmetyczne |
    stała_znakowa), średnik;
obługa_wyjścia = słowo_kluczowe_out, operator_wypisywania, (wyrażenie_arytmetyczne |
    stała_znakowa), średnik;

krok = identyfikator, operator_przypisania, ( wyrażenie_arytmetyczne |
    wywołanie_funkcji_metody);

(*
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 = [ operator_negacji ], lewy_nawias, warunek, prawy_nawias |
    wyrażenie_arytmetyczne;

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

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

odwołanie_do_atrybutu = (słowo_kluczowe_this | identyfikator), kropka, identyfikator;
```

Język obiektowy

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

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

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

operator_przypisania = "=";
operator_negacji = "!";
operator_relacyjny = "<=" | ">=" | "==" | "!=" | "<" | ">";
operator_logiczny = "||" | "&&";
operator_matematyczny = operator_dodawania | operator_mnożenia | operator_odejmowania;
operator_odejmowania = "-";
operator_dodawania = "+";
operator_mnożenia = "*" | "/";
operator_indeksu = "[]";
operator_wypisywania = "<<";
operator_wczytywania = ">>";

słowo_kluczowe_else = "else";
słowo_kluczowe_when = "when";
słowo_kluczowe_loop = "loop";
słowo_kluczowe_operator = "operator";
słowo_kluczowe_none = "none";
słowo_kluczowe_extends = "extends";
słowo_kluczowe_in = "in";
słowo_kluczowe_out = "out";
słowo_kluczowe_this = "this";
słowo_kluczowe_get = "get";
słowo_kluczowe_set = "set";

przecinek = ",";
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, "\'" );


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

Struktura

Informacje, które muszą być przechowywane:

1. Zmienna: identyfikator, typ, wartość

2. Obiekt: identyfikator, identyfikator obiektu bazowego, atrybuty, identyfikatory metod, przeciążone operatory.
   Każda instancja obiektu musi posiadać taki 'zbiór'. Atrybuty w porównaniu do zmiennych muszą przechowywać jeszcze informację o tym, czy korzystają z wbudowanych metod get/set.

3. Funkcje: identyfikator, argumenty, zmienne lokalne.
   Dla każdego wywołanie funkcji musi powstawać nowy kontekst - wartości zmiennych mogą się różnić typami i wartościami w różnych wywołaniach funkcji. Argument w porównaniu do zmiennych musi jeszcze przechowywać informację czy jest do odczytu czy zapisu. W przypadku wywołań rekurencyjnych konteksty odkładane na stosie.

4. Pętle/instrukcja warunkowa: zmienne lokalne.
   Mogą występować zmienne widoczne tylko w danej pętli/instrukcji. Trzeba zatem stworzyć dla nich oddzielny kontekst.