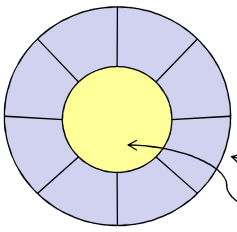
Techniki programowania

Wprowadzenie do zadania 2

1. Podstawy programowania obiektowego
2. Standardowa biblioteka wzorców STL

marzec 2014

Klasa i jej składowe



Obiekt każdej klasy:
- posiada część publiczną (*ang. public*),
- może posiadać część prywatną (*ang. private*).

Deklarowanie klasy

Klasa to idea

```
class PewnaKlasa
{
  public:
  private:
}
```

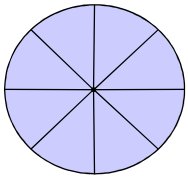
Definiowanie obiektów klasy

```
PewnaKlasa Obiekt1;
PewnaKlasa Obiekt2;
PewnaKlasa* wsk = new PewnaKlasa;
    ...
```

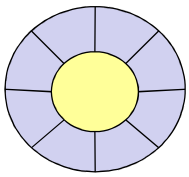
Obiekty to realizacja idei - jej konkretyzacja

Klasa a struktura

W pewnym uproszczeniu struktura może być postrzegana jako klasa, w której wszystkie składowe zadeklarowano jako publiczne.



Zmienna typu strukturalnego



Obiekt pewnej klasy

Składowymi struktury i klasy mogą być dane różnych typów oraz funkcje.

W odróżnieniu od struktur, każda klasa posiada dwie szczególne funkcje nazywane konstruktorem i destruktorem

3

Konstruktor i destruktor

Konstruktor i destruktor definiowany jest w części publicznej. Definiowany jest automatycznie jako konstruktor/destruktor domyślny albo jawnie przez programistę.

Wybrane zasady:

- Nazwa konstruktora i destruktora musi być taka jak nazwa klasy, przy czym nazwa destruktora poprzedzona jest znakiem tylda.
- Konstruktor domyślny nie zawiera parametrów.
- Konstruktor wywoływany jest podczas definiowania obiektu.
- Można zdefiniować wiele konstruktorów, które będą funkcjami przeciążonymi (patrz wykład POP o podprogramach).

4

Odwołania do składowych obiektu

Ponieważ struktura jest szczególnym przypadkiem klasy można spodziewać się, że syntaktyka odwołań do składowych obiektów jest podobna do odwołań do składowych struktur.

```
nazwa_obiektu.nazwa_składowej
```

Pamiętajmy, że składowe mogą być funkcjami. Wówczas odwołanie do takiej składowej równoważne jest wywołaniu odpowiedniej funkcji.

Jeżeli operujemy wskaźnikiem do obiektu, to można stosować notację z operatorem * albo notację skróconą ->

```
wskaźnik_na_obiekt -> nazwa_składowej

( *wskaźnik_na_obiekt ).nazwa_składowej
```

5

Przykłady definicji klasy (1)

Przykład: `Kot2.cpp`

6

Tablica obiektów

```
...
class Osoba {
  public:
     Osoba();
     ~Osoba();
     int PobierzWiek() { return jejWiek; }     // implementacja funkcji w def. klasy
     void UstawWiek(int wiek) { jejWiek = wiek;}
...
  private:
        int jejWiek;
      ...
}

Osoba TablicaOsob[MAX_L_OS];         // tablica obiektów klasy Osoba
...
for( int i = 0; i < ile_osob; i++ ) {
   cout << "Podaj wiek osoby: ";
   cin >> wiek;
   TablicaOsob[i].UstawWiek(wiek);    // przypisanie wartości składowej obiektu
     ...                               // przez wywołanie funkcji składowej UstawWiek
}
...
for( int i = 0; i < ile_osob; i++ ) {
   cout << "Osoba " << i << " ma "
        << TablicaOsob[i].PobierzWiek() << " lat ";
    ...
}
```

7

Przykład definicji prostej klasy (2)

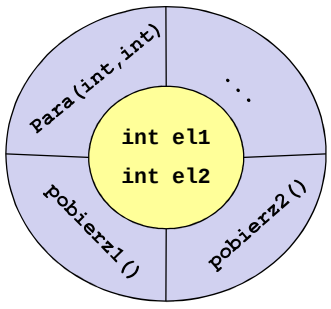
```
class Para {
   public:
       Para();              // konstruktor domyślny
       Para( int, int );    // drugi konstruktor
       ~Para();             // destruktor
       int pobierz1();
       int pobierz2();
       // ...
   private:
       int el1;
       int el2;
};

Para::Para( int a, int b ){
  el1 = a;  el2 = b;
}

int Para::pobierz1() {
   return(el1);
}

int Para::pobierz2() {
   return(el2);
}

int main() {
    Para P(2,3);   // wywołanie konstruktora obiektu P klasy para
    cout << "element 1 = " << P.pobierz1() << endl
         << "element 2 = " << P.pobierz2() << endl;
}
```



8

Przykład definicji prostej klasy (3)

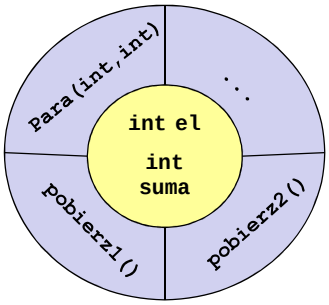
```
class Para {
   public:
       Para();
       Para( int, int );
       ~Para();
       int pobierz1();
       int pobierz2();
       // ...
   private:
       int suma;
       int el;
};

Para::Para( int a, int b ){
  suma = a + b;
  el = b;
}

int Para::pobierz1() {
   return(suma - el);
}

int Para::pobierz2() {
   return(el);
}

int main() {
    Para P(2,3);
    cout << "element 1 = " << P.pobierz1() << endl
         << "element 2 = " << P.pobierz2() << endl;
}
```



Zmieniły się składowe obiektu, w tym funkcje zapewniające dostęp do składowych prywatnych ale użycie obiektu pozostało takie samo.
Zdolność ukrywania implementacji wnętrza obiektu nazywana jest hermetyzacją.

9

Przykłady definicji klasy (4)

Przykład: **Kot_Dynam.cpp**

10

Przykłady definicji klasy (5)

Przykład: **Kot_Dynam2.cpp**

Tablica dynamicznych obiektów

Tablicę taką tworzymy analogicznie do tablicy dynamicznych struktur

```
...
class Osoba {
  public:
     Osoba();
     ~Osoba();
     int PobierzWiek(){ return jejWiek; }
     void UstawWiek(int wiek) { jejWiek = wiek;}
        ...
  private:
        int jejWiek;
      ...
}
Osoba* TablicaOsob[MAX_L_OS];        // tablica wskaźników na obiekt klasy Osoba
...
for( int i = 0; i < ile_osob; i++ ) {
   TablicaOsob[i] = new Osoba;
   cout << "Podaj wiek osoby: ";
   cin >> wiek;
   TablicaOsob[i]->UstawWiek(wiek);     // przypisanie wartości składowej obiektu
     ...
}
...
for( int i = 0; i < ile_osob; i++ ) {
   cout << "Osoba " << i << " ma "
        << TablicaOsob[i]->PobierzWiek() << " lat ";
    ...
}
```

Standardowa biblioteka wzorców STL
(ang. Standard Template Library)

Biblioteka zawierająca kilkaset szablonów klas i funkcji.

Najważniejsze z nich to tzw. kontenery, które są szablonami klas pozwalającymi tworzyć obiekty służące do przechowywania zbiorów innych obiektów oraz do zarządzania nimi.

Kontenery posiadają wewnętrzną strukturę oraz zestaw metod, które pozwalają zarządzać obiektami umieszczonymi wewnątrz kontenera.

13

Standardowa biblioteka wzorców STL

Podstawowe kontenery STL:

- vector
- queue
- dequeue
- list
- set
- multiset
- map
- multimap
- stack
- priority_queue

Dołączanie kontenerów:

```
#include <nazwakontenera>
```

Przykład:

```
#include <multimap>
#include <priority_queue>
```

14

Kontener vector

1. Jednowymiarowa tablica dynamiczna z automatyczną alokacją pamięci:

   = pojemność (**`capacity`**)
   = rozmiar (**`size`**)

   = **`resize`** zmiana aktualnej liczby elementów (rozmiaru),
   = **`reserve`** alokacja/dealokacja pamięci (pojemności).

2. Odwołania do elementów:
   = operator **`[]`**
   = metoda **`pushback`**

3. Przykład: **`vector1.cpp`**

15

Iteratory

1. Iteratory to obiekty stosowane do wskazywania elementu kontenera(nie mylić ze wskaźnikami).
2. Iterator "wie" gdzie jest obiekt.
3. Wskazując iteratorem kolejne elementy można "wędrować" po kontenerze.
4. Iteratory zaimplementowane są tak aby były "bezpieczne", np. nie pozwalają wyjść poza zakres wektora.
5. Definiowanie:

```
nazwa_kontenera<Typ>::iterator nazwa_iteratora
```

   Przykład definicji:

```
vector<int>::iterator iter1;
```

6. Przykład: **`vector1_iterator.cpp`**

`it` - iterator
`rit` - iterator wsteczny (reverse iterator)
`cit` - stały iterator
`crit` - stały iterator wsteczny (const reverse iterator)

16

Kontener stack

1. Realizuje stos

2. Metody kontenera stack

```
= push
= pop
= top
= size
= empty
= ...
```

3. Przykłady:

```
stack.cpp
stack_punkt.cpp
weighted_stack.cpp
```

17

Kontener queue

1. Realizuje kolejkę

2. Metody kontenera queue

```
= push
= pop
= front
= back
= size
= empty
= ...
```

18

Literatura

[1] Jesse Liberty, C++ dla każdego, Helion 2002

[2] Artykuły dotyczące STL w czasopiśmie elektronicznym Warp 2.0 digital

= część I, luty 2006
= część II, marzec 2006
= część III, czerwiec 2006

19