

Języki i Paradygmaty Programowania

dr inż. Michał Ciesielczyk
michal.ciesielczyk@put.poznan.pl

Konsultacje:
środa, 08:30-09:30, pokój BM-319
środa, 11:15-11:45, sala M-216

Hermetyzacja

...

Laboratorium 02

Demo

• • •

nieprawidłowa modyfikacja struktury ***Date***

Specyfikatory dostępu

- private
 - domyślny dla klas
 - dostępność tylko z wnętrza danej klasy
 - (oraz klas/funkcji zaprzyjaźnionych)
- public
 - publicznie dostępne

Demo

• • •

kontrola dostępu w strukturze ***Date***

struct vs. class

- Struktura jest klasą, której wszystkie elementy są publiczne:

```
struct X {  
    int m;  
    // ...  
};
```

jest równoważne:

```
class X {  
public:  
    int m;  
    // ...  
};
```

struct vs. class

- Klasa jest strukturą, której wszystkie elementy są prywatne:

```
struct X {  
    private:  
        int m;  
        // ...  
};
```

jest równoważne:

```
class X {  
    int m;  
    // ...  
};
```

struct vs. class

- Elementy klasy są domyślnie prywatne:

```
class X {  
    int m;  
};
```

jest równoważne

```
class X {  
private:  
    int m;  
};
```

- W rezultacie:

```
X x;           // zmienna x typu X  
x.m = 7;       // błąd: m jest prywatne (tzn. niedostępne)
```


Modyfikatory dostępu

- Dlaczego wszystko nie mogłoby być publicznie dostępne?
 - Czytelny interfejs
 - Dane oraz funkcje wewnętrzne mogą być prywatne
 - Zapewnienie poprawności danych
 - Tylko niektóre funkcje mają dostęp do danych
 - Prostsze debugowanie
 - Tylko niektóre funkcje mają dostęp do danych
 - Możliwość zmiany reprezentacji
 - Wystarczy zmodyfikować jedynie ograniczoną liczbę funkcji
 - Pozostała część kodu pozostaje bez zmian

Demo

...

Modyfikatory dostępu

Jak projektować klasy?

- Jak powinien być interfejs klasy?
 - Minimalistyczny
 - Najprostszy jak się da
 - Kompletny
 - I nie prostszy
 - Bezpieczny (type safe)
 - Uwaga na myłącą kolejność argumentów
 - Uwaga na zbyt ogólne typy danych (np. int do reprezentacji miesiąca)
 - Poprawny z uwagi na stałe

Jak projektować funkcje?

- Nazwa funkcji powinna określać co ona wykonuje
 - Najczęściej rozpoczyna się od czasownika
 - Np.: `add()`, `print()`, `count()`
 - ŹLE: ~~`tablica()`~~, ~~`wys_k()`~~, ~~`dane()`~~
- Funkcja powinna być jak najkrótsza
 - Dzielić problemy na mniejsze!
 - Generalnie: jeśli funkcja ma ponad 100 linii i nie potrafisz jej skrócić – prawdopodobnie robisz coś źle.

Zadania

...

Przykład: BST

```
class BSTree {
```

```
private:
```

```
    struct Node {  
        int value;  
        Node* left;  
        Node* right;  
        Node(int);  
        void printInOrder();  
    };
```

```
    void clear(Node*&);
```

```
    Node* root;
```

```
public:
```

```
    BSTree();  
    ~BSTree();  
    bool isEmpty();  
    void insert(int);  
    void printInOrder();
```

```
};
```

struktura prywatna

funkcja prywatna

pole prywatne

interfejs publiczny

Przykład: BST

konstruktor Node

```
BSTree::Node::Node(int value) :  
    value(value), left(nullptr), right(nullptr) {}
```

inicjalizacja składowych klasy
(member initialization)

więcej informacji można znaleźć tutaj:

[http://www.cplusplus.com/doc/tutorial/classes/
#member_initialization](http://www.cplusplus.com/doc/tutorial/classes/#member_initialization)

Przykład: BST

Node::printInOrder()

```
void BSTree::Node::printInOrder() {  
    if (left)  
        left->printInOrder();  
    cout << value << " ";  
    if (right)  
        right->printInOrder();  
}
```

wyświetlamy lewe
podrzewo (jeśli istnieje)

wyświetlamy wartość
węzła

wyświetlamy prawe
podrzewo (jeśli istnieje)

Przykład: BST

BSTree::clear(Node*&)

```
void BSTree::clear(Node*& node) {
```

```
    if (node) {
```

```
        clear(node->left);
```

```
        clear(node->right);
```

```
        // cout<<"Usuwamy: "<<node->value<<endl;
```

```
        delete node;
```

```
        node = nullptr;
```

```
    }
```

```
}
```

przekazujemy wskaźnik
przez referencję

usuwamy lewe i prawe
podrzewa

(możesz prześledzić
usuwanie elementów)

usuwamy węzeł
(wywołujemy jego
destruktor – tutaj pusty!)

Przykład: BST konstruktor / destruktor

```
BSTree::BSTree() : root(nullptr) {}
```

konstruktor bezargumentowy
BSTree (inicjalizujemy korzeń
drzewa)

```
BSTree::~~BSTree() {  
    clear(root);  
}
```


destruktor BSTree (czyścimy
korzeń drzewa)

zwróć uwagę na
implementację funkcji clear()
na poprzednim slajdzie

Przykład: BST

BSTree::isEmpty()

```
bool BSTree::isEmpty() {  
    return root == nullptr;  
}
```



sprawdzamy czy korzeń
jest pusty

Przykład: BST

BSTree::printInOrder()

```
void BSTree::printInOrder() {  
    cout << "[";  
    root->printInOrder();  
    cout << "];"  
}
```

wykorzystujemy funkcję drukowania węzła (w tym przypadku zaczynamy od korzenia)

- Alternatywnie można np. przeciążyć operator << dla std::ostream z wykorzystaniem funkcji zaprzyjaźnionych.

Przykład: BST

BSTree::insert(int)

```
void BSTree::insert(int value) {  
    if (isEmpty()) root = new Node(value);  
    else {  
        Node* current = root;  
        Node* prev = nullptr;  
        while (current) {  
            prev = current;  
            if (current->value == value) return;  
            else if (current->value > value)  
                current = current->left;  
            else  
                current = current->right;  
        }  
        if (prev->value > value)  
            prev->left = new Node(value);  
        else  
            prev->right = new Node(value);  
    }  
}
```

drzewo jest puste –
inicjalizujemy korzeń

tymczasowe zmienne do
przeglądania drzewa

pamiętamy poprzednik

element już istnieje
(kończymy)

przechodzimy do
odpowiedniego poddrzewa

tworzymy nowy liść w
odpowiednim miejscu

Przykład: BST

- Pozostały do zaimplementowania:

- `bool BSTree::contains(int)`

zauważ analogię z funkcją
insert (przeglądanie
drzewa)

- `void BSTree::clear()`

zauważ analogię z
destruktor

- UWAGA! Pamiętaj, że podana tutaj implementacja to jedynie przykład. Twoja może wyglądać zupełnie inaczej!
 - Ograniczeniem jest jedynie interfejs publiczny, który jest stały.