Objektově orientované programování

Modularita

2023/24

Modul?

```
class KeyValue
{
    private:
        int key;
        double value;

    public:
        KeyValue(int k, double v);
        int GetKey();
        double GetValue();
};
```

Osnova přednášky

- Vývoj programování
- Modularita
- Příklad

Vývoj programování

Paradigmata programování

- Jak a proč se jazyky vyvíjejí?
- V čem se OOP liší od předchozích paradigmat?
- Aspekty kvality software.

Velké projekty

- Programování velkých projektů je kvalitativně odlišné od vytváření malých programů.
- Důvodem k úvahám o principech programování byla a je **rostoucí komplexnost počítačových programů**.

Paradigmata

- Imperativní programování (popisujeme JAK se řeší)
- Deklarativní programování (popisujeme CO se řeší)
  - Funkcionální programování
  - Logické programování
- Modulární programování
- Objektově orientované programování

Imperativní programování

- Procedurální programování
  - Posloupnost kroků, kterými měníme stav proměnných programu.
- Strukturované programování
  - Sekvence, iterace, větvení, skoky, abstrakce.
- Tak, jak to dnes známe z běžně používaných jazyků.

Modulární programování

- Návrh shora-dolů.
- Rozdělení programu na nezávislé, zaměnitelné moduly, které zajišťují dílčí funkčnost.
- Modul obsahuje vše nezbytné pro zajištění funkčnosti (data a algoritmy).

Objektově orientované programování

Faktory kvality software

- Vnitřní a vnější faktory.
- Vnitřní jsou skryty uživateli. (JAK)
- Vnější popisují chování navenek. (CO)
- Musíme umět měřit kvalitu.

Vnější faktory

- Správnost
- Robustnost
- Rychlost, rozšiřitelnost, použitelnost, kompatibilita,. . .

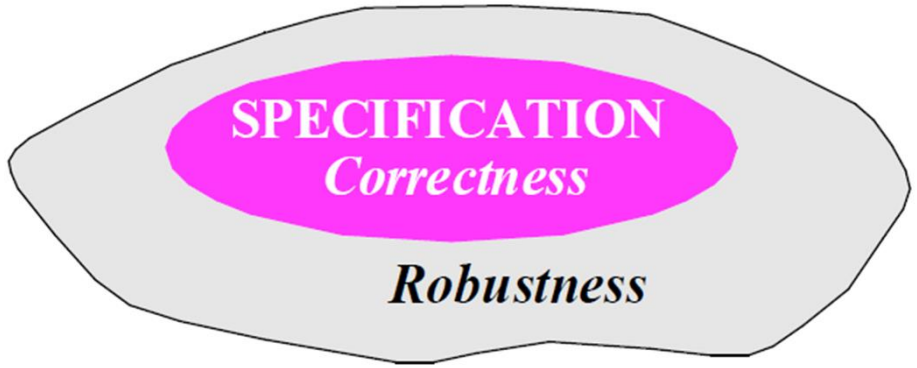
Klíčové faktory

Definition: correctness

Correctness is the ability of software products to perform their exact tasks, as defined by their specification.

Definition: robustness

Robustness is the ability of software systems to react appropriately to abnormal conditions.



Robustnost?

Náklady na robustnost mohou být vyšší než náklady vynaložené na správnost.

Robustnost není cílem předmětu OOP, proto ji nebudeme vyžadovat…

…a budeme tedy vždy předpokládat správné vstupy.

Další faktory

Definition: extendibility

Extendibility is the ease of adapting software products to changes of specification.

Definition: reusability

Reusability is the ability of software elements to serve for the construction of many different applications.

Definition: compatibility

Compatibility is the ease of combining software elements with others.

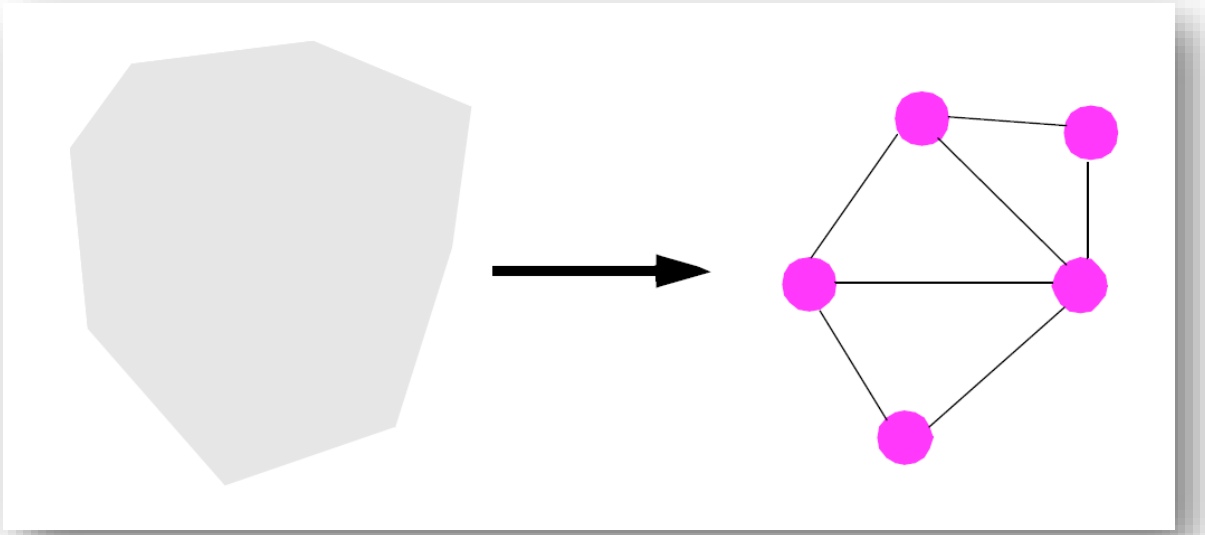
Definition: functionality

Functionality is the extent of possibilities provided by a system.

Modularita

Modul

- Metoda konstrukce software je modulární, pokud je založena na dekompozici, která pomáhá návrhářům decentralizovat architekturu a tvořit softwarové systémy složené z autonomních prvků, *modulů*, propojených do jednoduchých a srozumitelných struktur.



Modularita programu

- Pochopitelnost
- Samostatnost
- Kombinovatelnost
- Zapouzdření
- Explicitní rozhraní
- Syntaktická podpora

Pochopitelnost

- Modul by měl vykonávat jednu jasně definovanou a pochopitelnou úlohu, popřípadě několik málo jasně definovaných úloh.

Samostatnost

- Každý modul musí být relativně samostatný, a měl by mít co možná nejmenší počet vazeb na ostatní moduly.
- Nebylo by vhodné, aby byly všechny moduly programu navzájem propojené a na sobě závislé.
- Moduly bychom tak nemohli individuálně otestovat, pochopit ani přenést do jiného projektu.

Kombinovatelnost

- Moduly musí být navzájem kombinovatelné. Musí být možno modul vzít a použít jej v jiném kontextu nebo třeba i v jiném projektu.

Zapouzdření

- Moduly musí mít právo na jisté soukromí: je přípustné a žádoucí, aby veškeré informace, které nejsou potřebné pro klienty modulů, zůstaly skryté uvnitř modulu.

- V praxi se pak ukazuje, že většina funkcionality modulu je skryta a jen malá část je viditelná navenek.

- Skryté části říkáme **implementace** modulu a veřejné části říkáme **rozhraní** (interface) modulu.

Explicitní rozhraní

- Z deklarace modulu musí být všeobecně zřejmé, jaké předpoklady pro vykonávání své úlohy potřebuje.

Syntaktická podpora

- Moduly počítačového programu musí být jasně vymezeny syntaktickými jednotkami programu.
- Ze zápisu programovacího jazyka musí být zcela evidentní, kde začíná a kde končí zápis jednoho modulu.
- Není například možné, aby v kompaktním programu patřily modulu jen některé části, zatímco jiné části modulu nepatřily.

Pět kritérií a pravidel modularity

- Dekomponovatelnost
- Kombinovatelnost
- Pochopitelnost
- Kontinuita
- Ochrana
- Přímé mapování
- Pár rozhraní
- Malá rozhraní (weak coupling)
- Explicitní rozhraní
- Skrývání informací

Kritéria
dekomponovatelnost, kombinovatelnost, pochopitelnost

A software construction method satisfies Modular Decomposability if it helps in the task of decomposing a software problem into a small number of less complex subproblems, connected by a simple structure, and independent enough to allow further work to proceed separately on each of them

A method satisfies Modular Composability if it favors the production of software elements which may then be freely combined with each other to produce new systems, possibly in an environment quite different from the one in which they were initially developed.

A method favors Modular Understandability if it helps produce software in which a human reader can understand each module without having to know the others, or, at worst, by having to examine only a few of the others.

Kritéria
kontinuita, ochrana

A method satisfies Modular Continuity if, in the software architectures that it yields, a small change in a problem specification will trigger a change of just one module, or a small number of modules.

A method satisfies Modular Protection if it yields architectures in which the effect of an abnormal condition occurring at run time in a module will remain confined to that module, or at worst will only propagate to a few neighboring modules.

Pravidla
přímé mapování, pár rozhraní, malá rozhraní

The modular structure devised in the process of building a software system should remain compatible with any modular structure devised in the process of modeling the problem domain.

Every module should communicate with as few others as possible.

If two modules communicate, they should exchange as little information as possible

Pravidla
explicitní rozhraní, skrývání informací

> Whenever two modules *A* and *B* communicate, this must be obvious from the text of *A* or *B* or both.

> The designer of every module must select a subset of the module's properties as the official information about the module, to be made available to authors of client modules.

Příklad

Třída x Objekt

- Třída jako statický popis.
- Objekt jako run-time (běhová) reprezentace:
  - stav (data)
  - chování (algoritmy)

Terminologie

- Třída
- Členská funkce, metoda
- Členská položka, proměnná
- Objekt, instance třídy, **this** (vykonavatel metody)
- Konstruktor, destruktor

Deklarace třídy

```
#include <iostream>
using namespace std;

class KeyValue
{
private:
    int key;
    double value;
    KeyValue *next;

public:
    KeyValue(int k, double v);
    ~KeyValue();
    int GetKey();
    double GetValue();
    KeyValue* GetNext();
    KeyValue* CreateNext(int k, double v);
};
```

Implementace (definice) třídy

```
KeyValue::KeyValue(int k, double v)
{
    this->key = k;
    this->value = v;
    this->next = nullptr;
}

KeyValue::~KeyValue()
{
    if (this->next != nullptr)
    {
        delete this->next;
        this->next = nullptr;
    }
}
```

```
KeyValue* KeyValue::GetNext()
{
    return this->next;
}

KeyValue* KeyValue::CreateNext(int k, double v)
{
    this->next = new KeyValue(k, v);
    return this->next;
}
```

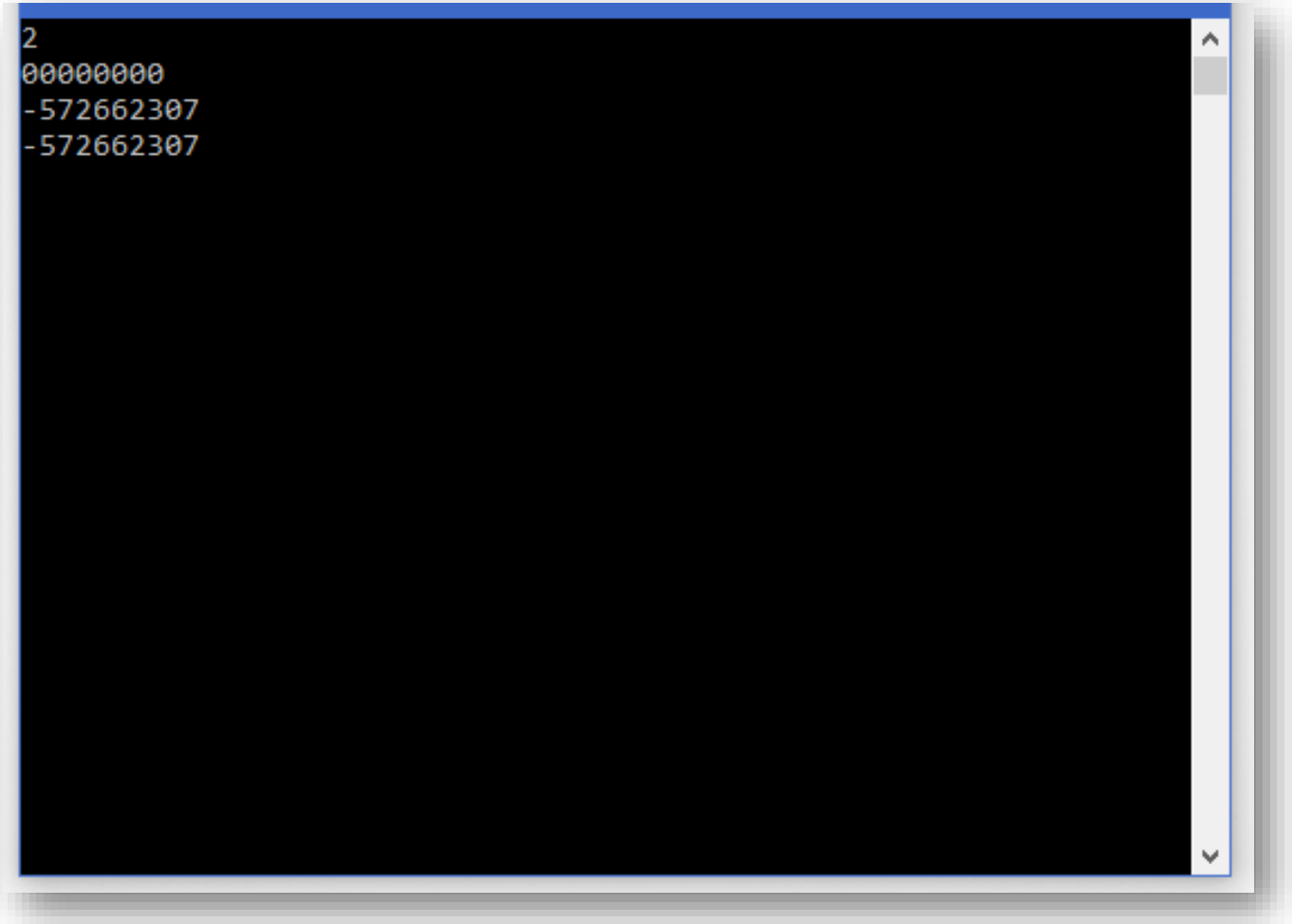
Použití třídy 1

```
int main()
{
    KeyValue *kv1 = new KeyValue(1, 1.5);
    cout << kv1->CreateNext(2, 2.5)->GetKey() << endl;

    KeyValue *kv2 = kv1->GetNext();
    cout << kv2->GetNext() << endl;

    delete kv1;
    //delete kv2;

    cout << kv1->GetKey() << endl;
    cout << kv2->GetKey() << endl;

    getchar();
    return 0;
}
```

Použití třídy 2

```
int main()
{
    KeyValue *kv1 = new KeyValue(1, 1.5);
    cout << kv1->CreateNext(2, 2.5)->GetKey() << endl;

    KeyValue *kv2 = kv1->GetNext();
    cout << kv2->GetNext() << endl;

    //delete kv2;
    delete kv1;

    cout << kv1->GetKey() << endl;
    cout << kv2->GetKey() << endl;

    getchar();
    return 0;
}
```

Výsledek



Použití třídy 3

```
int main()
{
    KeyValue *kv1 = new KeyValue(1, 1.5);
    cout << kv1->CreateNext(2, 2.5)->GetKey() << endl;

    KeyValue *kv2 = kv1->GetNext();
    cout << kv2->GetNext() << endl;

    delete kv1;
    kv1 = nullptr;
    kv2 = nullptr;

    //cout << kv1->GetKey() << endl;
    //cout << kv2->GetKey() << endl;

    getchar();
    return 0;
}
```

Konstruktor a destruktor

- Konstruktor inicializuje data objektu hodnotami parametrů v konstruktoru (naplní paměť daty).
- Destruktor odstraní z paměti data objektu (čistí paměť).
- Destruktor není potřeba, pokud jsou data objektu statická.

Úkoly na cvičení

- Implementujte třídu *KeyValue* dle přednášky a vytvořte zřetězenou lineární strukturu mnoha (např. tisíce) objektů a pracujte s ní (vypište např. všechny klíče od prvního do posledního objektu).

- Vytvořte podobnou třídu jako *KeyValue*, ale s hodnotou i klíčem typu (třídy) *string* a se dvěma sousedícími (next) objekty. Výsledkem bude tzv. strom.

- Implementuje strukturu (rozhodovací strom) na identifikaci zvířat nebo rostlin. Klíčem uzlu stromu je rozhodovací kritérium, hodnotou uzlu je název zvířete nebo rostliny, resp. druhu apod. Naplňte klíč alespoň deseti objekty a vypište celou strukturu na obrazovku.

Kontrolní otázky

- Co je hlavním motivem pro vývoj programovacího paradigmatu od imperativního k objektovému?
- Co je imperativní programování?
- Co je modulární programování?
- Jaké jsou hlavní faktory kvality software?
- Co je pochopitelnost modulu? Uveďte příklad.
- Co je samostatnost modulu? Uveďte příklad.
- Co je kombinovatelnost modulu? Uveďte příklad.
- Co je zapouzdření modulu? Uveďte příklad.
- Co je explicitní rozhraní modulu? Uveďte příklad.
- Co je syntaktická podpora modularity?
- Co je pět kritérií pro dobrou modularitu?
- Co se rozumí pěti pravidly zajišťující dobrou modularitu?
- Popište jednotlivá kritéria dobré modularity. Uveďte příklady.
- Popište jednotlivá pravidla pro dobrou modularitu. Uveďte příklady.
- K čemu je konstruktor? Uveďte příklad.
- K čemu je destruktor, kdy ho potřebujeme a kdy ne? Uveďte příklad.

Ke studiu

- Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall 1997. [3-16, 39-52]