Objektově orientované programování

Objektová dekompozice a třída jako objekt

2023/24

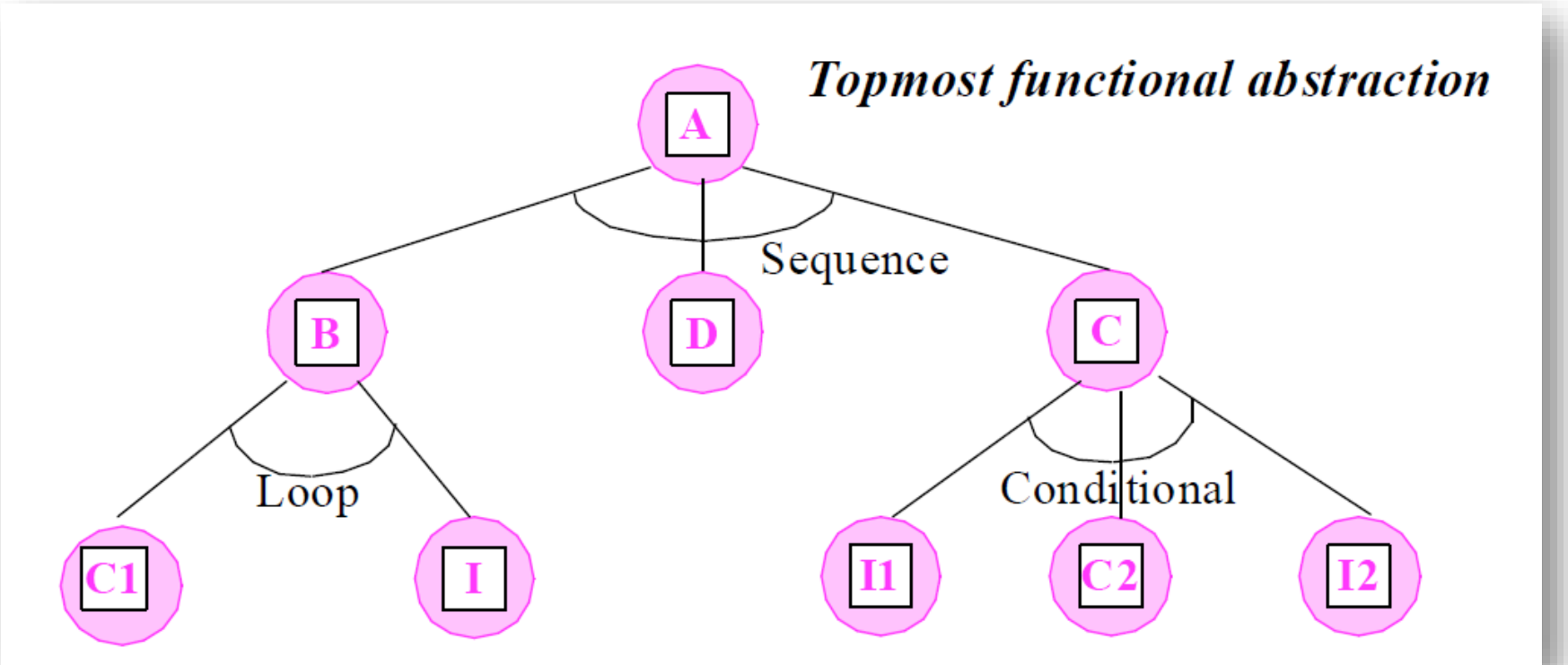
Osnova přednášky

- Co je lepší? Funkce nebo objekty?
- Může být třída zároveň objektem?
- Příklad.

Funkce nebo objekty?

Funkce x objekty

- Je lepší založit strukturu programu na funkcích nebo datech?
- Na návrh systému můžeme nahlížet dvěma způsoby:
  - Jako na sadu funkci (odpovídá na otázku, **CO** systém bude dělat).
  - Jako na sadu objektů, které spolupracují (odpovídá na otázku, **KDO** bude funkčnost zajišťovat).


Topmost functional abstraction
A
Sequence
B
D
C
Loop
Conditional
C1
I
I1
C2
I2

Problémy

- Rozšiřitelnost
- Opakovaná použitelnost
- Kombinovatelnost

Object-oriented software construction (definition 1)

Object-oriented software construction is the software development method which bases the architecture of any software system on modules deduced from the types of objects it manipulates (rather than the function or functions that the system is intended to ensure).

OBJECT MOTTO

Ask not first what the system does:

Ask what it does it to!

Příklad zadání

- Mějme malou **banku** s omezeným počtem **klientů** a **účtů**. V bance mohou klienti a účty přibývat.
- Každý účet má jednoho vlastníka a může mít jednoho partnera, oba jsou klienti banky a mají jméno a kód. Na účty lze vkládat a vybírat z nich, lze zjistit stav na účtu. Pokud není na účtu dostatek peněz, nelze vybrat.
- Vklady na účtech jsou úročeny, a to buď základní nebo speciální úrokovou sazbou. Jednou za čas banka všem účtům připíše úrok odpovídající úrokové sazbě.
- Účet resp. klienta je možno v bance vyhledat podle čísla resp. kódu.

Funkce nebo objekty?

- Computation involves three kinds of ingredient: processors (or threads of control), actions (or functions), and data (or objects).
- A system's architecture may be obtained from the functions or from the object types.
- A description based on object types tends to provide better stability over time and better reusability than one based on an analysis of the system's functions.
- It is usually artificial to view a system as consisting of just one function. A realistic system usually has more than one "top" and is better described as providing a set of services.

Shora dolů nebo naopak?

- It is preferable not to pay too much attention to ordering constraints during the early stages of system analysis and design. Many temporal constraints can be described more abstractly as logical constraints.
- Top-down functional design is not appropriate for the long-term view of software systems, which involves change and reuse.

Proč objekty?

- Object-oriented software construction bases the structure of systems on the types of objects they manipulate.
- In object-oriented design, the primary design issue is not what the system does, but what types of objects it does it to. The design process defers to the last steps the decision as to what is the topmost function, if any, of the system.
- To satisfy the requirements of extendibility and reusability, object-oriented software construction needs to deduce the architecture from sufficiently abstract descriptions of objects.
- Two kinds of relation may exist between object types: client and inheritance.

Třídy jako objekty? Proč?

Třída jako objekt

- Objektově orientovaný přístup obecně vychází z předpokladu, že „všechno je objekt“ (tedy i typy jsou objekty).
- Může být i třída objektem? A za jakých podmínek?
- *Objekty mají svůj stav a chování…*

Stav a chování třídy

- Stav je reprezentován daty.
- Chování je reprezentováno metodami.
- Musí být splněno zapouzdření a skrývání informace.
- Třídě se musí dát zaslat zpráva (zavolat její metodu).

Příklad

Deklarace a definice

```
class StaticValue
{
private:
    static int value;

public:
    static void IncValue();
    static int GetValue();
};
```

```
int StaticValue::value = 0;

void StaticValue::IncValue()
{
    StaticValue::value += 1;
}

int StaticValue::GetValue()
{
    return StaticValue::value;
}
```

Použití

```
int main()
{
    cout << StaticValue::GetValue() << endl;
    StaticValue::IncValue();
    cout << StaticValue::GetValue() << endl;

    StaticValue *sv = new StaticValue();
    cout << sv->GetValue() << endl;

    getchar();
    return 0;
}
```

Jak to je…

- Data a metody deklarované jako „static“ patří třídě.
- Přístup k nim ale mají i objekty (instance) třídy.
- Je potřeba rozlišovat mezi **třídními** a **instančními** proměnnými a metodami.

Vhodné konvence

- V kontextu třídy se při přístupu k datům nebo metodám nemusí uvádět adresát zprávy.
- Pro zabránění nedorozumění je dobré:
  - pro přístup k instančním datům/metodám používat formu **OBJECT_NAME->METHOD_NAME**
  - pro přístup k třídním datům/metodám používat formu **CLASS_NAME::METHOD_NAME**

Adresát zprávy

- Adresátem zprávy je tedy buď
  - objekt (instance) této třídy v případě instanční proměnné nebo metody
  - sama třída v případě třídní, a tedy **static**, proměnné nebo metody

Kde je rozdíl?

```
class StaticValue
{
private:
    static int value;
    StaticValue();

public:
    static void IncValue();
    static int GetValue();
};
```

Třída bez objektů

```
int main()
{
    cout << StaticValue::GetValue() << endl;
    StaticValue::IncValue();
    cout << StaticValue::GetValue() << endl;

    StaticValue *sv = new StaticValue();
    cout << sv->GetValue() << endl;

    getchar();
    return 0;
}
```

Konstruktor? Destruktor?

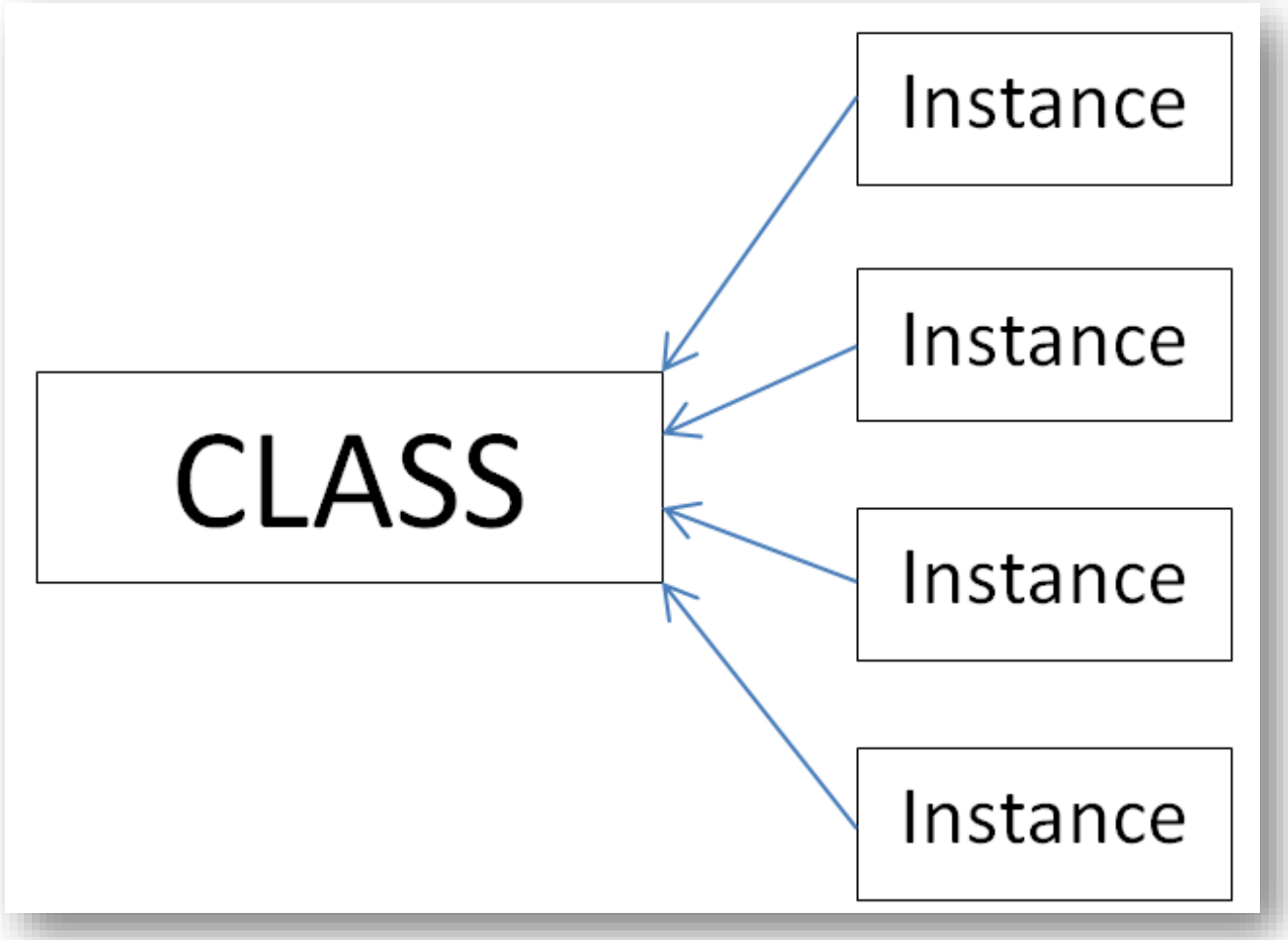
- Třída existuje po celou dobu běhu programu.
- Pokud má třída třídní (static) proměnné, pak je musíme inicializovat zvlášť

```
int StaticValue::value = 0;
```

- Konstruktor ani destruktor pro třídu jako objekt neexistuje.

Kdo o kom ví?

- Prostřednictvím konstruktoru třídy objekty (instance) vytvoříme, ale třída o nich nic neví.
- Z třídní metody nelze přistupovat k členským položkám objektu.
- Objekty (instance) třídy mají přístup k členským (static) položkám třídy (při použití nemusí být rozpoznatelné, s jakou položkou pracujeme).


CLASS
Instance
Instance
Instance
Instance

Co je správné volání?

```
int main()
{
    cout << StaticValue::GetValue() << endl;
    StaticValue::IncValue();
    cout << StaticValue::GetValue() << endl;

    StaticValue *sv = new StaticValue();
    cout << sv->GetValue() << endl;

    getchar();
    return 0;
}
```

Kdy použít třídu jako objekt?

- Vytvoření knihovny funkcí (např. matematika).
- Potřebujeme, aby objekty (instance) sdílely společná data.
- Např. evidence počtu objektů (instancí) třídy.

Upravte třídu pro počítání objektů

```
class Client
{
private:
    int code;
    string name;

public:
    Client(int c, string n);

    int GetCode();
    string GetName();
};
```

Deklarace a definice

```
class Client
{
private:
    static int objectsCount;
    int code;
    string name;

public:
    static int GetObjectsCount();

    Client(int c, string n);
    ~Client();

    int GetCode();
    string GetName();
};
```

```
int Client::objectsCount = 0;

Client::Client(int c, string n)
{
    this->code = c;
    this->name = n;
    Client::objectsCount += 1;
}

Client::~Client()
{
    Client::objectsCount -= 1;
}

int Client::GetObjectsCount()
{
    return Client::objectsCount;
}
```

Úkoly na cvičení

- Implementujte příklady z přednášky a doplňte do tříd *Client* a *Account* počítání existujících objektů.

- Navrhněte a implementujte další příklady členských položek tříd. Například stejnou úrokovou sazbu pro všechny účty, kterým nebyla sazba zadána v konstruktoru a kterou lze prostřednictvím metody třídy změnit.

Otázky

- Jaký je rozdíl mezi funkční a objektovou dekompozicí programu?
- Proč preferujeme objektovou dekompozici a jaké jsou hlavní problémy funkční dekompozice?
- Za jakých podmínek můžeme považovat třídu za objekt a jak to implementovat v C++?
- Vysvětlete rozdíl mezi členskými položkami třídy a instance a popište jejich dostupnost.
- Jak můžeme v C++ důsledně odlišovat práci s členskými položkami tříd a instancí?
- Potřebuje třída v roli objektu konstruktor resp. destruktor a proč?

Ke studiu

- Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall 1997. [101-120]