Objektově orientované programování

Třídy a objekty
(objektová orientovanost)

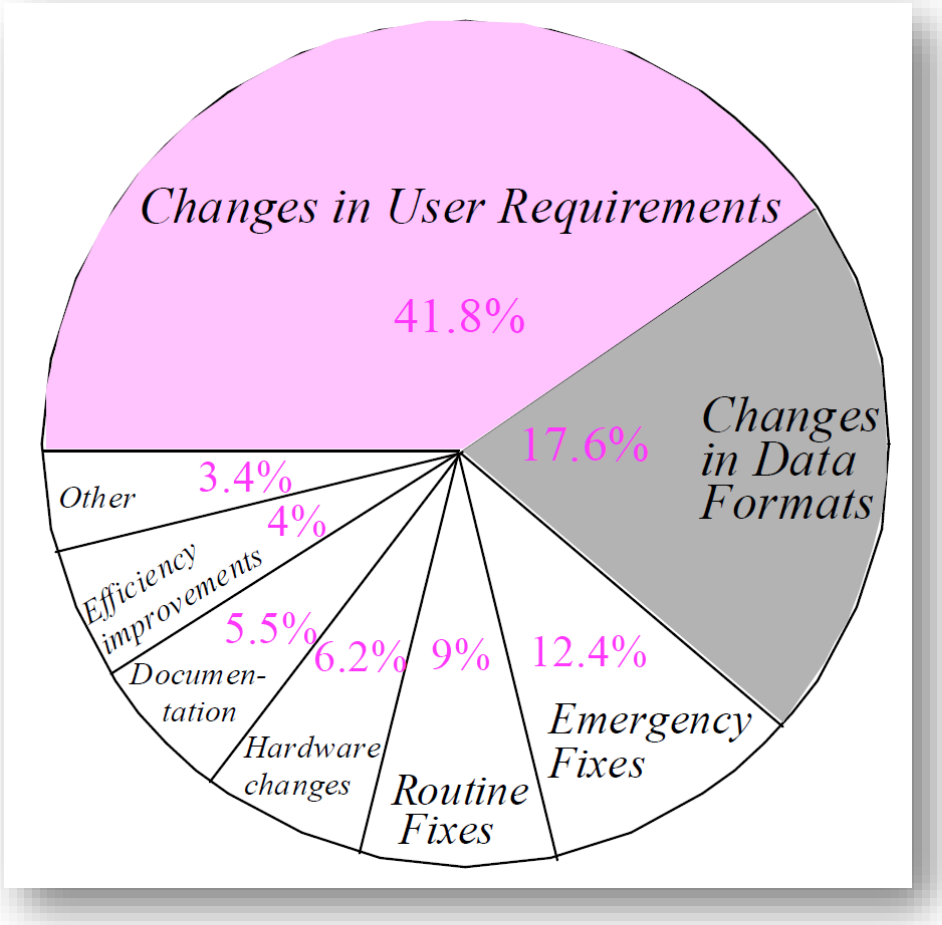
2023/24

Osnova přednášky

- Objektový přístup (proč potřebujeme objekty).
- Třídy, objekty,...
- Příklad.

Proč potřebujeme objekty?

Udržovatelnost softwaru (maintenance)



- Studie Lientz a Swanson (1980).
- 487 různých systémů.
- **Jak mnoho úsilí to vyžaduje?**

Kdy to začalo?

- Jazyk *Simula 67* (Ole-Johan Dahl a Kristen Nygaard, Norwegian Computing Center, 1960+)
  - třídy a instance (objekty)
  - automatické zanikání objektů (garbage collection)
- Jazyk *Smalltalk* (Xerox PARC, Alan Kay a další, 1970+)
  - pojem „objektově-orientované programování“
  - použití objektů a zpráv, které si objekty posílají a zpracovávají

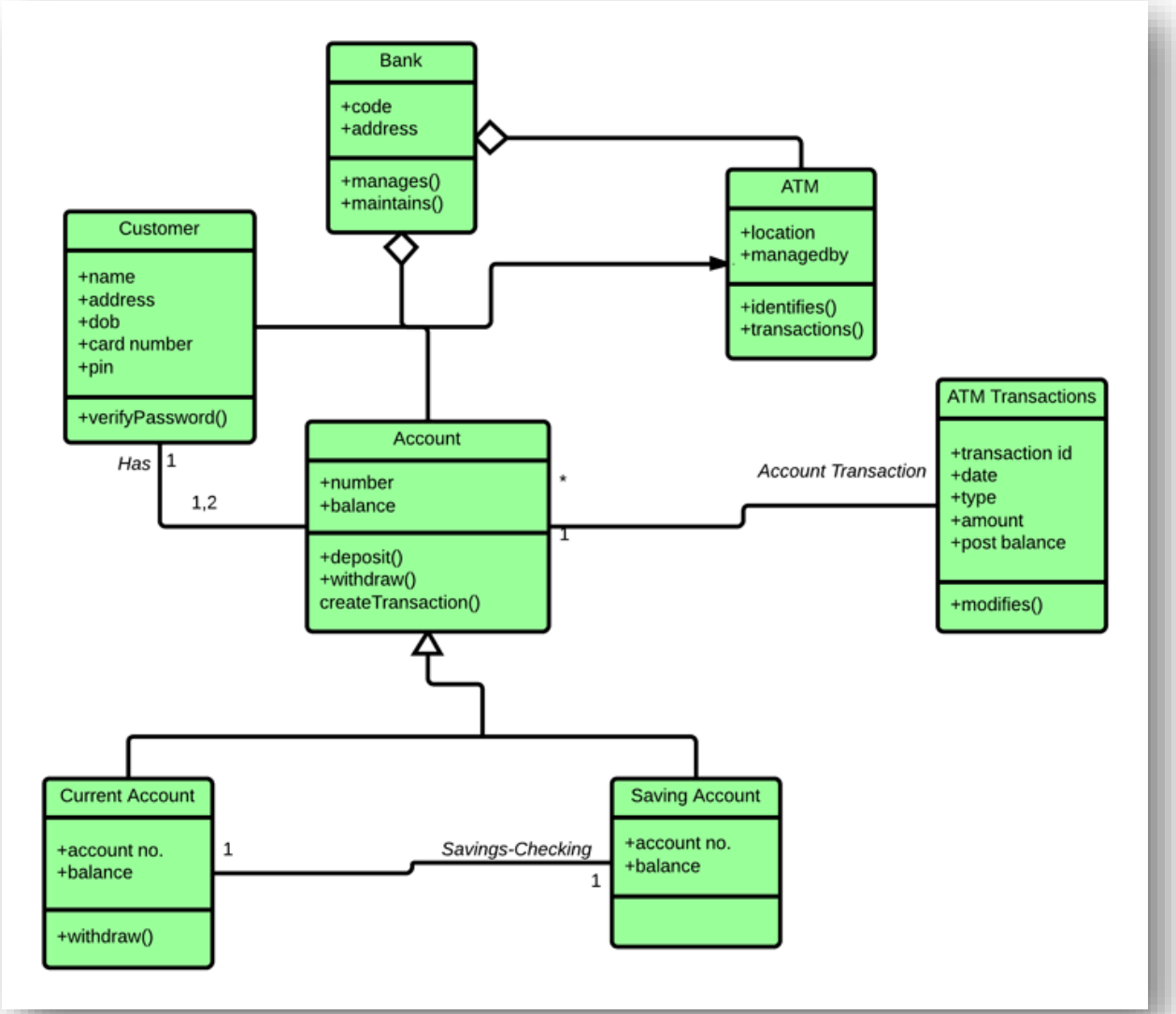
Objektová orientovanost…

- Objektové techniky pomáhají napsat lépe udržovatelný software.
  - Metoda a jazyk.
  - Implementace a prostředí.
  - Knihovny.

Metoda a jazyk

- Nejde jen o programovací jazyk a způsob jeho použití.
- Jde i o způsob uvažování a vyjadřování…
- …a také o záznamy v textové či grafické formě.

Model domény



Implementace a prostředí

- Podpora vývoje
  - Vlastnosti a efektivita nástrojů pro vývoj.
  - Nástroje pro podporu nasazení nových verzí.
  - Nástroje pro podporu dokumentování.

Knihovny

- Objektové technologie velmi spoléhají na opakovanou použitelnost.
- Jde o podporu vývoje formou využití již implementovaných řešení (knihoven).
- Jde také o podporu vytváření a správy nových vlastních knihoven.

Máme být dogmatičtí?

- Objektově orientovaný přístup se dnes chápe jako základní nástroj pro vývoj softwaru. Nicméně…
  - Existují různé programovací jazyky s různou mírou podpory technik objektového programování (OOP).
  - Ne každý potřebuje všechny vlastnosti, které OOP nabízí.
  - Objektová orientovanost může být jen jedním z faktorů úspěšného vývoje, proto je potřeba uvažovat komplexněji.

Metoda a jazyk

- Třídy
- Třídy jako moduly
- Třídy jako typy
- Zasílání zpráv (message passing, feature-based computation)
- Skrývání informací
- Statická kontrola typů
- Dědičnost, redefinice, polymorfismus a dynamická vazba
- Generičnost
- Správa paměti a garbage collection

Třída

Třídy

> The method and the language should have the notion of class as their central concept.

- Objektově orientovaný přístup je postaven na pojmu třída.
- Třídu můžeme chápat jako část softwaru, která popisuje abstraktní datový typ a jeho implementaci.
- Abstraktním datovým typem rozumíme objekty se společným chováním reprezentovaným seznamem operací, které umí objekty vykonávat.

Třídy jako moduly

> Classes should be the only modules.

- OOP je zejména o struktuře (architektuře) softwaru, kde je důležitá modularita.
- Třídy nepopisují jen typy objektů, ale musí být zároveň modulární jednotky.
- V čistě objektově orientovaných programech by neměly být jiné samostatné jednotky než třídy (např. funkce).

Třídy jako typy

> Every type should be based on a class.

- V čistě objektově orientovaných jazycích a programech by neměly být jiné typy než třídy.

- Platí to i pro systémové typy jako INT nebo DOUBLE.

Zasílání zpráv

Feature call should be the primary computational mechanism.

- Message passing (feature call), *feature-based computation* je výpočetní mechanismus.
  - Objektu jako instanci třídy je zaslána zpráva daného jména a s potřebnými parametry.
  - *aPerson->ChangeLastName("Smith")*
  - Ten, kdo zasílá zprávu (požaduje vykonání operace s určitými argumenty) je **KLIENT** třídy.

Skrývání informací

> It should be possible for the author of a class to specify that a feature is available to all clients, to no client, or to specified clients.

- Pro klienta jsou důležité pouze ty operace (metody), které popisují vnější chování objektů třídy.
- Detaily implementace by měly zůstat skryty (data + pomocné operace).
- Pokud klient potřebujeme získat informace a stavu (datech) objektu, dostane se k nim pouze prostřednictvím zaslání zprávy (voláním metody ne proměnné).

Statická vazba a kontrola typů

> A well-defined type system should, by enforcing a number of type declaration and compatibility rules, guarantee the run-time type safety of the systems it accepts.

- Každá entita v programu (např. proměnná daného jména) musí mít definovaný typ.

- Požadavek na objekt (zaslání zprávy) musí odpovídat operaci (metodě), kterou třída poskytuje.

Generičnost

> It should be possible to write classes with formal generic parameters representing arbitrary types.

- Je potřeba, aby existovaly třídy, které umí pracovat s typem, který není dopředu znám.

- Např. seznamy, do kterých lze ukládat objekty různých tříd (typů).

Dědičnost a redefinice

It should be possible to define a class as inheriting from another.

It should be possible to redefine the specification, signature and implementation of an inherited feature.

- Děděním se rozumí založení nové třídy na základě již existující třídy. Základem je rozšíření původní třídy o nové vlastnosti.

- V rámci dědění se také předpokládá možnost změnit některé vlastnosti původní třídy.

Polymorfismus a dynamická vazba

It should be possible to attach entities (names in the software texts representing run-time objects) to run-time objects of various possible types, under the control of the inheritance-based type system.

Calling a feature on an entity should always trigger the feature corresponding to the type of the attached run-time object, which is not necessarily the same in different executions of the call.

- Někdy potřebujeme, aby jeden a tentýž objekt vystupoval v různém kontextu v různých rolích.
- Rolí rozumíme různé chování, které se může měnit v čase.

Správa paměti a garbage collection

> The language should make safe automatic memory management possible, and the implementation should provide an automatic memory manager taking care of garbage collection.

- V rozsáhlých programech vzniká a zaniká mnoho objektů, a to v různých a ne zcela jednoduchých situacích.
- Je problém ručně uhlídat životní cyklus objektů.
- V moderních jazycích je zanikání objektů potřeba automatizovat.

Příklad

Deklarace

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

class KeyValue
{
private:
    int key;
    double value;

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

Konstruktor inicializuje objekt (plní paměť daty, která objekt používá)

Destruktor odstraňuje data objektu (uvolňuje paměť, kterou objekt zabírá)

```
class KeyValues
{
private:
    KeyValue** keyValues;
    int count;

public:
    KeyValues(int n);
    ~KeyValues();
    KeyValue* CreateObject(int k, double v);
    KeyValue* SearchObject(int key);
    int Count();
};
```

Použití

Použití klíčového slova *new* zajistí vznik objektu (alokuje paměť pro data („plochou“ část) objektu).

```
int main()
{
    int N = 5;
    KeyValues* myKeyValues = new KeyValues(N);

    KeyValue* myKeyValue = myKeyValues->CreateObject(0, 0.5);
    cout << myKeyValue->GetValue() << endl;

    for (int i = 1; i < N; i++)
    {
        myKeyValues->CreateObject(i, i + 0.5);
    }
    cout << myKeyValues->SearchObject(4)->GetValue() << endl;

    delete myKeyValues;

    //cout << myKeyValue->GetKey() << endl;

    getchar();
    return 0;
}
```

Výsledek

Definice (implementace)

```
KeyValues::KeyValues(int n)
{
    this->keyValues = new KeyValue*[n];
    this->count = 0;
}

KeyValues::~KeyValues()
{
    for (int i = 0; i < this->count; i++)
    {
        delete this->keyValues[i];
    }

    delete[] this->keyValues;
}
```

```
int KeyValues::Count()
{
    return this->count;
}
```

```
KeyValue* KeyValues::CreateObject(int k, double v)
{
    KeyValue *newObject = new KeyValue(k, v);

    this->keyValues[this->count] = newObject;
    this->count += 1;

    return newObject;
}

KeyValue* KeyValues::SearchObject(int k)
{
    for (int i = 0; i < this->count; i++)
    {
        if (this->keyValues[i]->GetKey() == k)
        {
            return this->keyValues[i];
        }
    }

    return nullptr;
}
```

Úkoly na cvičení

- Implementujte příklad z přednášky a do třídy *KeyValues* přidejte metodu *KeyValue* RemoveObject(int k)*, která odebere objekt s daným klíčem a vrátí ukazatel na něj.

- Implementuje třídu *Faktura*, která bude obsahovat *číslo*, objekt třídy *Osoba* (se *jménem* a *adresou*) a pole objektů *PolozkaFaktury* (s *názvem* a *cenou*). Navrhněte konstruktor a destruktor a další potřebné metody. Faktura bude mít metodu (funkci), která spočítá a vrátí celkovou cenu.

Kontrolní otázky

- Co je hlavními příčinami potřeby změn software?
- Jaké jsou hlavní faktory ovlivňující objektovou orientovanost?
- Vysvětlete, co rozumíme pojmy objektově orientovaná metoda (přístup) a jazyk.
- Vysvětlete, co rozumíme podporou objektově orientované implementace.
- Vysvětlete, co rozumíme podporou opakované použitelnosti.
- Vysvětlete pojmy třída a objekt a použijte správnou terminologii.
- Zdůrazněte vlastnosti třídy z pohledu modularity.
- Vysvětlete princip zapouzdření v OOP.
- Vysvětlete princip zasílání zpráv.
- Vysvětlete principy deklarace a definice jednoduché třídy v C++.

Ke studiu

- Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall 1997. [17-36]