FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ústav teoretické a experimentální elektrotechniky

Seminář C++

5. přednáška

Autor: doc. Ing. Jan Mikulka, Ph.D.

2023

Obsah přednášky

- Dědičnost
  - Jednoduchá
  - Vícenásobná
  - Řízení přístupu k členům zděděné třídy
- Virtuální funkce, čistě virtuální funkce
- Abstraktní třídy
- Konstruktor a destruktor při dědění

Dědičnost

- Jedná se o jednu ze základních vlastností objektově orientovaného programování.
- Slouží k tvorbě unifikovaného a předem známého předepisování API tříd.
- Princip dědičnosti spočívá v hierarchické tvorbě tříd zapouzdřujících vždy pouze společné prvky tříd odvozených.
- Definujeme dva typy dědičnosti:
  - Jednoduchá – odvozená třída dědí pouze z třídy základní (je zde pouze jedna úroveň dědičnosti).
  - Vícenásobná – odvozená třída dědí z třídy, která je také třídou odvozenou (několik úrovní dědičnosti).

Syntaxe dědičnosti

```
class bazova_trida
{
private:
    int a;
public:
    int funkce_bazova(int a, int b);
};



class odvozena_trida : public bazova_trida
{
private:
    int b;
public:
    int funkce_odvozena(int a, int b);
};
```

Bázová třída

Třída odvozená z bázové třídy

Řízení přístupu

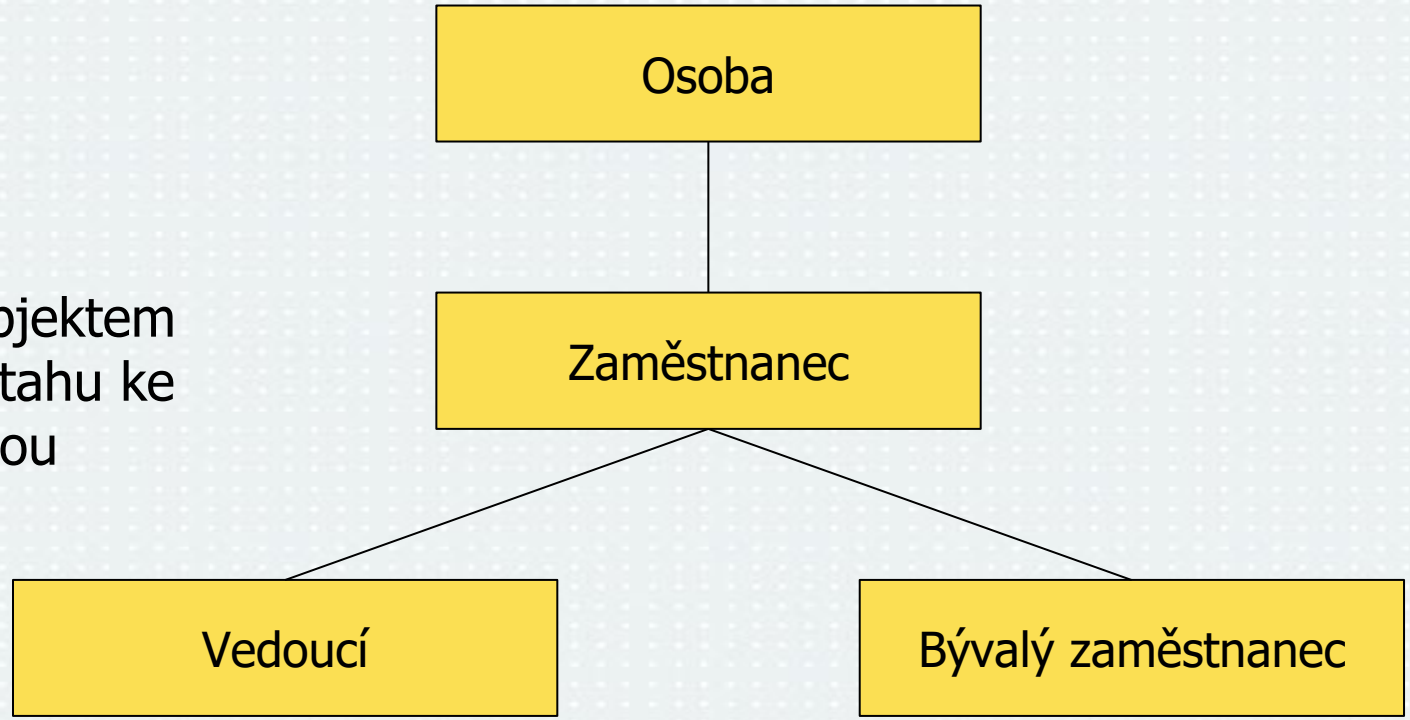
Řízení přístupu při dědění

Modifikátor v deklaraci	Modifikátor v bázové třídě	**Přístupová práva v odvozené třídě**
`public`	`public`	`public`
`public`	`protected`	`protected`
`public`	`private`	`private`
`protected`	`protected`	`protected`
`protected`	`public`	`protected`
`protected`	`private`	`private`
`private`	`private`	`private`
`private`	`protected`	`private`
`private`	`public`	`private`

Příklad použití dědičnosti

- Máme za úkol vytvořit databázi osob. Tyto osoby mohou mít různé vztahy ke společnosti:
  - zaměstnanec
  - bývalý zaměstnanec
  - vedoucí
  - osoba bez vztahu ke společnosti

- Každá osoba je reprezentovaná jiným objektem (zapouzdřuje jiné funkce odpovídající vztahu ke společnosti), ale jsou vlastnosti, které jsou společné pro všechny osoby:
  - ID
  - jméno
  - příjmení



Příklad použití dědičnosti

```
class Osoba
{
protected:
    int ID;
    string jmeno, prijmeni;

public:
    Osoba(string j, string p) {
        jmeno = j;
        prijmeni = p;
    }

    void setID(int id) {
        ID = id;
    }

    virtual string console_string(void) {
        return string("ID: " + std::to_string((long double)ID) + " - " + jmeno + " " + prijmeni);
    }

    void tisk(void) {
        cout << console_string();
    }
};
```

Bázová třída

Společné proměnné
ID, jméno, příjmení

Konstruktor pro nastavení
jména a příjmení

Funkce pro nastavení ID

Virtuální funkce sestavující
řetězec pro výpis osoby

Funkce pro výpis osoby

Příklad použití dědičnosti

```
class Zamestnanec : public Osoba
{
protected:
    string oddeleni;
public:
    Zamestnanec(string j, string p, string o) : Osoba(j, p)
    {
        oddeleni = o;
    }

    virtual string console_string(void)
    {
        return string("ID: " + std::to_string((long double)ID) + " - " + jmeno +
                      " " + prijmeni + ", oddeleni: " + oddeleni);
    }
};
```

Odvozená třída…

… z bázové třídy Osoba

Zaměstnanec dědí vše z objektu Osoba a přidává proměnnou oddeleni

Konstruktor

Do řetězce pro výpis musíme přidat i oddělení zaměstnance

Příklad použití dědičnosti

Odvozená třída...

```
class Vedouci : public Zamestnanec
{
public:
    Vedouci(string j, string p, string o) : Zamestnanec(j, p, o)
    {
    }
};
```

Odvozená třída...

```
class Byvalyzam : public Zamestnanec
{
protected:
    int rok_ukonceni;
public:
    Byvalyzam(string j, string p, string o, int ru) : Zamestnanec(j, p, o)
    {
        rok_ukonceni = ru;
    }

    virtual string console_string(void)
    {
        return string("ID: " + std::to_string((long double)ID) + " - " + jmeno +
                      " " + prijmeni + ", oddeleni: " + oddeleni + ", rok ukonceni: " +
                      std::to_string((long double)rok_ukonceni));
    }
};
```

Příklad použití dědičnosti

```
int _tmain(int argc, _TCHAR* argv[])
{
    Osoba *O[4];

    O[0] = new Osoba("Pavel", "Novak");
    O[1] = new Zamestnanec("Jan", "Mikulka", "UTEE");
    O[2] = new Byvalyzam("Pavel", "Dvorak", "UTKO", 2003);
    O[3] = new Vedouci("Pavel", "Fiala", "UTEE");

    for (int i = 0; i < 4; i++) O[i]->setID(i+1);

    cout << "Vypiseme vsechny vedouci: " << endl;
    for (int i = 0; i < 4; i++)
        if (typeid(*O[i]) == typeid(Vedouci)) O[i]->tisk();

    cout << endl << endl;

    cout << "Vypiseme vsechny zamestnance: " << endl;
    for (int i = 0; i < 4; i++)
        if (typeid(*O[i]) == typeid(Zamestnanec)) O[i]->tisk();

    cout << endl << endl;

    system("pause");
    return 0;
}
```

Virtuální funkce

- Virtuální funkce je členská funkce, která je deklarovaná (definovaná) v základní třídě a je redefinovaná v odvozené třídě.
- Aby byla metoda virtuální, je třeba před její hlavičku uvést klíčové slovo **virtual**.
- Virtuální funkce může být volána jako jakákoliv jiná členská funkce. Když ukazatel základní třídy ukazuje na odvozený objekt, který obsahuje virtuální funkci, a tato virtuální funkce je volána prostřednictvím ukazatele, pak se na základě typu objektu odkazovaného ukazatelem určí a to za běhu programu, která verze virtuální funkce bude spuštěna.
- Tímto způsobem se dociluje polymorfismu za běhu programu.
- V našem příkladu je jedna virtuální funkce – console_string.
- Pro objekty O[0] – O[3] se vždy vybere při volání této funkce ta, která je „nejblíže" typu, na který objekt ukazuje.

Čistě virtuální funkce

- Čistě virtuální funkce má v bázové třídě pouze deklaraci – je tedy bez definice. Takovou funkci nelze volat a v odvozené třídy musí čistě virtuální funkce definovat, jinak dojde v chybě v překladu.

- Čistě virtuální funkci deklarujeme následovně

```
class bazova_trida
{
public:
    virtual void ciste_virtualni_fce(void) = 0;
};
```

Abstraktní třídy

- Abstraktní třída je taková třída, která zapouzdřuje alespoň jednu deklaraci virtuální funkce.
- Jsou obecným konceptem pro odvozené třídy.
- Nelze vytvořit instanci typu abstraktní třída.

Konstruktor a destruktor při dědění

- Při definici instance odvozené třídy se volá standardně konstruktor této třídy.
- Před tím se ale volá konstruktor třídy, ze které je třída odvozená.
- Postupně se tedy volají všechny konstruktory v dané hierarchii od nejnižší úrovně až po nejvyšší.

Konstruktor a destruktor při dědění

- U konstruktoru v odvozené třídě si můžeme vybrat, který konstruktor se má volat v třídě bázové včetně předání parametrů a to následovně:

```
class Byvalyzam : public Zamestnanec
{
protected:
    int rok_ukonceni;
public:
    Byvalyzam(string j, string p, string o, int ru) : Zamestnanec(j, p, o)
    {
        rok_ukonceni = ru;
    }

    virtual string console_string(void)
    {
        return string("ID: " + std::to_string((long double)ID) + " - " + jmeno +
                      " " + prijmeni + ", oddeleni: " + oddeleni + ", rok ukonceni: " +
                      std::to_string((long double)rok_ukonceni));
    }
};
```

Výběr konstruktoru bázové třídy

Děkuji Vám
za pozornost.

mikulka@vut.cz
www.utee.fekt.vut.cz