

Dědičnost

- princip dědičnosti:
 - odvození vlastností deklarované třídy od jiné třídy
- odvozená třída dědí (automaticky přebírá) **atributy a metody**
 - **metody lze** v odvozené třídě **předefinovat**
 - **atributy nelze** v odvozené třídě **předefinovat** (nelze změnit typ atributu)
 - **lze dodefinovat nové atributy a metody**

- zápis:

```
class odvoz_trida: access zakl_trida  
{  
    deklarace nových atributů a metod;  
}
```

- nově deklarovaná třída `odvoz_trida` dědí od třídy `zakl_trida`
- místo **access** se dosazuje jedno ze tří klíčových slov: **public**, **private**, **protected**
 - řídí se jimi přístup ke zděděným položkám

- příklad
 - mějme definovanu třídu TAuto
 - třídu T0sobAuto, která má dědit od třídy TAuto deklarujeme např. takto:

```
class T0sobAuto: public TAuto
{
    deklarace;
}
```

Terminologie:

- třída `TOsobAuto` dědí položky a metody třídy `TAuto`
- třída `TOsobAuto` je potomkem (*descendant*) třídy `TAuto`
- třída `TOsobAuto` je podtřídou (*subclass*) třídy `TAuto`
- třída `TAuto` je předkem (*ancestor*) třídy `TOsobAuto`
- třída `TAuto` je nadtřídou (*superclass*) třídy `TOsobAuto`

Řízení přístupu k atributům a metodám nadtřídy

- přístup k atributům a metodám nadtřídy v odvozené třídě se určuje
 1. v základní třídě (nadtřídě)
 2. při deklaraci potomka

Pravidla:

- **veřejné (public)** položky jsou vždy přístupné (i v odvozené třídě)
- jsou-li atributy a metody v nadtřídě deklarovány jako **soukromé (private)**, pak **nejsou** v žádném případě **v odvozené třídě přístupné**
 - chceme-li podtřídě zpřístupnit položky nadtřídy, ale pro „okolí“ je ponechat nepřístupné, musíme je v nadtřídě deklarovat jako **chráněné (protected)**

Dále se způsob dědění určuje
specifikátorem přístupu `access` při
deklaraci potomka

Řízení přístupu při deklaraci potomka:

- **private:**
 - veřejné členy předka se stanou soukromými členy potomka
- **public:**
 - přístup zůstává stejný (veřejné členy předka zůstávají veřejné v potomkovi)
- **protected:**
 - chráněné a veřejné členy předka se stanou chráněnými v potomkovi

```
class TVekt2 {  
    float x,y;  
    public:  
        int vrat_dimenzi() { return 2; }  
        float velikost(){return sqrt(x*x+y*y);}  
        float vrat_x() { return x;}  
        void nastav(float px,float py){x=px; y=py;}  
};  
class TVekt3: public TVekt2 {  
    float z;  
    public:  
        int vrat_dimenzi() { return 3; }  
        float velikost();  
        void nastav(float px, float py, float pz);  
};
```

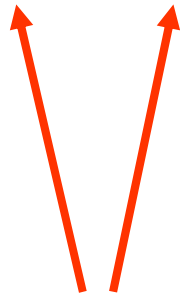
- třída `TVekt3` zdědila od třídy `TVekt2` atributy `x`, `y` a metodu `vrat_x()`; má tedy tři atributy a čtyři metody; tři metody jsou přetížené
- protože je předepsáno dědění typu **public**, je zděděná metoda veřejná

```
void TVekt3::nastav(float px, float py,  
    float pz)  
{  
    TVekt2::nastav(px, py);  
    z = pz;  
}
```

Volání metody
předka

- poznámka: nad objekty třídy TVekt3 již nelze vyvolat metodu nastav(px, py)

```
float TVekt3::velikost()  
{  
    return sqrt(x*x+y*y+z*z);  
}
```



**Chyba – zděděné atributy x,y nejsou ve třídě TVekt3
přístupné! (Proč?)**

Jak problém vyřešit?

2 možné cesty:

1. deklarovat ve třídě `TVekt2` ještě veřejnou metodu `vrat_y()` a použít metody `vrat_x()` a `vrat_y()` ve funkci `velikost()`
2. deklarovat ve třídě `TVekt2` atributy `x`, `y` jako chráněné (**protected**)

```
class TVekt3: private TVekt2 {  
    protected:  
        float z;  
    public:  
        TVekt2 prumet();  
};
```

```
void main(void)  
{  
    TVekt3 v;  
    v.vrat_dimenzi();  
    // je to správně nebo není?  
}
```

```
TVekt2 TVekt3::prumet()  
{  
    TVekt2 v;  
    v.nastav(x,y);  
    return v;  
};
```

Kompatibilita vzhledem k přiřazení

Pravidlo:

- potomka je možné přiřadit předkovi, ale ne naopak
- opačně lze provést přiřazení pouze s pomocí přetypování, což se ale příliš nedoporučuje

Proč ?

Potomek má všechny atributy a metody jako předek, ale předek nemá atributy (metody), které jsou v potomkovi deklarovány navíc. Při volání by mohlo dojít k chybě za běhu programu.

```
void main(void)
{
    TVekt2 v1,v2;
    TVekt3 v3,v4;
    v3.nastav(1,1,1);
    v1.nastav(0,0); v2.nastav(5,5);
    v1.vrat_x();
    v1=v3; //v pořádku – potomek přiřazuji předkovi
    v1.prumet(); //chyba při překladu – nelze vyvolat
    v4=v2; //chyba při překladu – nelze přiřadit
    v4 = (TVekt3)v2; // přetypováním lze obejít překladač
    v4.prumet(); // vyvolá chybu při běhu programu nebo
    // vrátí nesmyslný výsledek
}
```

Konstruktory, destruktory a dědičnost

- je možné mít konstruktory a destruktory u předků i potomků, event. jen u předka apod.
- při volání konstruktorů je spouštěn nejprve (automaticky) konstruktor předka, pak potomka
- při volání destruktůrů je volání provedeno v opačném pořadí (nejprve destruktore potomka)

- jestliže potřebujeme konstruktoru předka předat parametry (resp. volat jeden z přetížených konstruktorů předka), zapíšeme volání při implementaci konstrukturu potomka takto:

```
konstr_potomka(arg):konstr_predka(arg)
{
    tělo konstrukturu
}
```

```
class TVekt2 {  
    private:  
        float x,y;  
    public:  
        TVekt2();  
        TVekt2(float px, float py);  
};  
class TVekt3: public TVekt2 {  
    float z;  
    public:  
        TVekt3();  
        TVekt3(float px, float py, float pz);  
};
```

```
TVekt2::TVekt2 ()
```

```
{
```

```
    x=0; y=0;
```

```
}
```

```
TVekt2::TVekt2(float px, float py)
```

```
{
```

```
    x=px; y=py;
```

```
}
```

volání konstruktoru předka



```
TVekt3:: TVekt3():TVekt2()
```

```
{
```

```
    z = 0;
```

```
};
```

```
TVekt3:: TVekt3 (float px, float py, float
    pz) : TVekt2(px, py)
{
    z = pz;
};
```

Otázka:

Proč nemohu volat konstruktor předka takto?

```
TVekt3:: TVekt3 (float px, float py, float pz)
{
    TVekt2(px, py);
    z = pz;
};
```

Vícenásobná dědičnost

- třída může být potomkem *několika* rodičů
- deklarace:

```
class C: public A, public B  
{  
  
};
```

- třída C dědí atributy a metody od tříd A i B
- problém může nastat, obsahují-li třídy A a B položky (metody, atributy) označené stejným identifikátorem
 - pak je třeba použít kvalifikované jméno pomocí názvu třídy

```
class A { public int a; ... };  
class B { public char a; ... };  
class C: public A, public B {...};  
C c; c.A::a= 10; c.B::a = 'x';
```

Úkol

Implementujte třídu TCitac, která realizuje softwarový čítač s metodami Increment(), Decrement(), Reset() (nulování čítače), PreLoad(int x) (přednastavení hodnoty čítače na x).

Na základě třídy TCitac odvodte potomka TCitacMod – čítač mod n, který čítá modulo.