Anotace

- Středník se odehraje 8. 5. 2015,
- objektové programování II,
- programování her.

Zapouzdření, polymorfismus, dědičnost

- Zapouzdření (encapsulation) odkazuje k tomu, že atributy "žijí" v dotyčném objektu a metody pracují nad daty příslušného objektu. Někdy se tak mluví i o tom, že objekt je zvenku určitým způsobem chráněný.
- Polymorfismus – synovské třídy se mohou chovat určitým způsobem odlišně, než rodičovské třídy. Můžeme metody předefinovávat a překrývat.
- Dědičnost nám umožňuje definovat třídu jako šablonu určující, jak mají být definovány společné rysy synovských tříd.
  To ovšem není všechno.
- Pointerem na rodičovský typ je možno ukázat na syna.
  A nyní, co z toho všechno plyne?

Rodič ukazuje na syna

```
type  zvire=^zv;
      zv=object
            vek:integer;
            procedure kdotam;
            next:zvire;
      end;
      pes=^p;
      p=object(zv)
            procedure hlidej;
      end;
      kocka=^k;
      k=object(zv)
            procedure chyt_drzou_mys;
      end;
```

```
procedure zv.kdotam;
begin
      writeln('Ja ''sem ale zvire!');
end;

procedure p.hlidej;
begin
      writeln('Haf, baf!');
end;

procedure k.sezer_drzou_mys;
begin
      writeln('Kocka kolotocka dela ham!');
end;
```

Hlavní program

```
var zverinec:zvire;
begin
      zverinec:=new(zvire);
      zverinec^.next:=new(kocka);
      zverinec^.kdotam;
      zverinec^.next^.kdotam;
end.
```

Využití dědičnosti

- Dědičnost můžeme použít jako šablonu na jednotlivé (synovské) třídy.
- Jde tedy o něco jako variantní record z minuta!
- Jenže k tomu místo definování nových tříd budeme chtít předefinovávat staré metody,...
- ... což kupodivu lze.

Předefinovávání metod

Minulý příklad

```
type  zvire=^zv;
      zv=object
            vek:integer;
            procedure kdotam;
            next:zvire;
      end;
      pes=^p;
      p=object(zv)
            procedure hlidej;
      end;
      kocka=^k;
      k=object(zv)
            procedure chyt_drzou_mys;
      end;
```

Předefinovávání metod

A hle, metody se jmenují stejně!

```
type  zvire=^zv;
      zv=object
            vek:integer;
            procedure kdotam;
            next:zvire;
      end;
      pes=^p;
      p=object(zv)
            procedure kdotam;
      end;
      kocka=^k;
      k=object(zv)
            procedure kdotam;
      end;
```

Ale co udělá program?

- `var zv:zvire; zv:=new(zvire);`
  Vytvoříme nové zvíře.
- `zv^.kdotam;`
  Já 'sem ale zvíře!
- `zv:=new(kocka);`
  v pořádku, přiřazujeme syna do rodiče.
- `zv^.kdotam;`
  Ouha:
  Já 'sem ale zvíře!
  Proč?
- Protože jednotlivé metody se hledají v prototypu (tedy v popisu příslušné třídy, v tomto případě `zvire`).
  Jak z toho?

Virtuální funkce

- Virtuální metody se nehledají v prototypu, nýbrž každý objekt má **tabulku virtuálních metod** (alias VMT), ve které jsou pointery na jednotlivé virtuální funkce.
- Funkci `kdotam` tedy uděláme virtuální (zejména ve třídě `zvire`, pak už ale bude virtuální i v synovských třídách).

Předefinovávání metod

A hle, metody jsou virtuální!

```
type  zvire=^zv;
      zv=object
            vek:integer;
            procedure kdotam; virtual;
            next:zvire;
      end;
      pes=^p;
      p=object(zv)
            procedure kdotam; virtual;
      end;
      kocka=^k;
      k=object(zv)
            procedure kdotam; virtual;
      end;
```

Volání virtuálních metod

- `var zv:zvire; zv:=new(zvire);`
  Vytvoříme nové zvíře.
- `zv^.kdotam;`
  Já 'sem ale zvíře!
- `zv:=new(kocka);`
  v pořádku, přiřazujeme syna do rodiče.
- `zv^.kdotam;`
  'Kocka kolotocka'
- Virtuální metoda se hledá ve VMT, kde je adresa `kdotam` od typu `kocka`.

Poznámky

- Pokud neřekneme, že metoda má být v synovských metodách virtuální, překladač to buďto sám pochopí, nebo nepřeloží (GPC vydedukuje). Při divočejším předefinovávání virtuálních metod nevirtuálními můžeme mít problémy.
- Pokud má synovská třída (na kterou si ukazujeme pomocí rodičovského typu) destruktor, je potřeba, aby tento byl virtuální!

Funkce `typeof`

- Tím, že můžeme přiřadit syna do rodiče nám vzniká nepořádek. Občas chceme vědět, na jaký typ si ukazujeme.
- K tomu je funkce `typeof`, které předáme strukturu nebo jméno třídy:
- `if typeof(zv^)=typeof(k) then writeln('Je to kocka!');`
- Takto můžeme například implementovat spojový seznam s hlavou pomocí objektů, kde hlavu si označíme speciálním typem.
- Příklad lze najít na stránkách kolegy Kryla.

Čistě virtuální funkce, abstraktní třídy

- Jde o metody, které nechceme definovat (společný předek slouží jen jako šablona).
- Čistě virtuální funkce je virtuální funkce, kterou nechceme definovat v rodičovském typu, ale chceme, aby byla ve všech synovských typech překryta místní virtuální funkcí.
- V Pascalu stojí na dobrovolnosti (funkce definujeme jako RunError).
- Zpravidla (v moderních objektových jazycích) je syntaktická podpora a například v C++ nelze vůbec definovat proměnnou takového typu.
- Třída obsahující aspoň jednu čistě virtuální funkci se nazývá *abstraktní.*

Příklad čistě virt. metody

- Například každá živá hmota dýchá. Je ale otázka, jestli dýchá kyslík nebo oxid uhličitý a jestli dýchá žábrami nebo ústy (kyslík ze vzduchu nebo ten rozpuštěný ve vodě).
- Typu `ziva_hmota` tak definujeme metodu `dychej`, která bude čistě virtuální a bude muset být ve všech synovských třídách překryta.

Objekt `self`, operátor vzetí pointeru

- Říkali jsme si o konstruktorech a destruktorech. Konstruktor často udělal plno užitečné práce, například...
- přidal prvek do spojového seznamu (obousměrného). Jenže jak to udělal?
- ```
  type spojak=^sp; sp=object
       hod:longint; prev,next:spojak;
  ...
       constructor init(h:longint;p,n:spojak);
       begin  hod:=h; next:=n; prev:=p;
       end;
  ```
  Stačí to takhle?
- Správně, následník a předchůdce neukazuje na nás. Jak to zalepit?
  ```

- Pomocí objektu `self`, který odkazuje k současnému objektu.
- A pomocí operátoru vzetí pointeru `@`.
- Takhle: `next^.prev:=@self;`
  `prev^.next:=@self;`
- Takto tedy můžeme v případě potřeby odkázat k současnému objektu.
- Samozřejmě operátorem vzetí pointeru můžeme vytvořit pointer na cokoliv (podobně jako operátorem stříšky můžeme jakýkoliv (negenerický) pointer dereferencovat).

Teorie her

- Kombinatorická hra je hrou dvou hráčů. Stav hry je určen pozicí nějakých předmětů. Všechny zúčastněné předměty jsou viditelné. Jde o tzv. hru s úplnou informací.
- Příklad: Nimm, Podivná hra, Dáma, Šachy, Halma, Mlýn, Otrávená čokoláda...
- Kombinatorickými hrami nejsou: Poker, Prší, Mariáš, Black Jack, závody formulí...
- Zaměříme se na hrací část, ne na vstup a výstup.
- U her předpokládáme, že hrají rozumně se chovající jedinci (s motivací vyhrát).

Shannonova věta

Theorem (Shannon)

Každá kombinatorická hra má pro některého z hráčů neprohrávající strategii.

Důkaz.

Náznak: Buďto platí, že si jeden z hráčů může vynutit zacyklení hry (a tak neprohrát), nebo budeme zkoumat predikáty:
Existuje náš tah, že pro každý tah protihráče existuje náš tah, že pro každý tah protihráče... protihráč prohraje.
Pro každý náš tah existuje tah protihráče, že pro každý náš tah... my prohrajeme.
Formule jsou konečné, počty tahů jsou také konečné, jsou to vzájemně negace a lze je algoritmicky rozhodnout. □

Konec

...děkuji za pozornost...

Otázky?