

Objekty

Programovací techniky

doc. Ing. Jiří Rybička, Dr.
ústav informatiky
PEF MENDELU v Brně
rybicka@mendelu.cz

Objekty v jazyce Pascal

- Datový typ `object` implementován jako záznam

- Datový typ `object` implementován jako záznam
- Datové složky napřed, pak metody

- Datový typ **object** implementován jako záznam
- Datové složky napřed, pak metody
- Příklad:

```
type Neco = object
  {datové složky:}
  D: TypData;
  P: word;
  {metody}
  procedure A;
  function B: longint;
  constructor X(Y: byte);
  destructor Z;
end;
```

Objekty v jazyce Pascal

- Metody mají čtyři typy – procedura, funkce, konstruktor a destruktork

- Metody mají čtyři typy – procedura, funkce, konstruktor a destruktork
- Těla metod jsou definována dále, hlavička se opakuje

- Metody mají čtyři typy – procedura, funkce, konstruktor a destruktork
- Těla metod jsou definována dále, hlavička se opakuje
- Identifikátor metody je doplněn o identifikátor typu objektu

- Metody mají čtyři typy – procedura, funkce, konstruktor a destruktork
- Těla metod jsou definována dále, hlavička se opakuje
- Identifikátor metody je doplněn o identifikátor typu objektu
- V tělech metod jsou přímo dostupné datové složky objektu

- Metody mají čtyři typy – procedura, funkce, konstruktor a destruktork
- Těla metod jsou definována dále, hlavička se opakuje
- Identifikátor metody je doplněn o identifikátor typu objektu
- V tělech metod jsou přímo dostupné datové složky objektu
- V případě, že objekty jsou definovány v rozhraní modulu, jsou těla metod umístěna do implementační části

Objekty v jazyce Pascal

- Příklad implementace metod:

```
procedure Neco.A;  
begin  
end;  
function Neco.B: longint;  
begin  
end;
```


- Užití objektu – objekt je běžná proměnná (statická i dynamická)

- Užití objektu – objekt je běžná proměnná (statická i dynamická)
- Deklarace a užití mají shodnou syntax jako u záznamu, včetně možnosti použití příkazu `with`:

```
var N: Neco;  
begin N.X(17);  
      with N do begin  
          A;  
          writeln(B)  
      end;  
end.
```


- 1 Zapouzdřenost (datové složky + metody v jedné struktuře, každá datová složka je ovladatelná POUZE vhodnou metodou – syntaktické pomůcky)

- 1 Zapouzdřenost (datové složky + metody v jedné struktuře, každá datová složka je ovladatelná POUZE vhodnou metodou – syntaktické pomůcky)
- 2 Dědičnost (schopnost převzít datové složky a metody z jiného objektu). Datové složky se kopírují a nemohou být v následníkovi změněny, metody mohou měnit svá těla. Syntax:

```
type Jiny = object (Neco)
    E: boolean;
    procedure A;
end;
```

- 1 Zapouzdřenost (datové složky + metody v jedné struktuře, každá datová složka je ovladatelná POUZE vhodnou metodou – syntaktické pomůcky)
- 2 Dědičnost (schopnost převzít datové složky a metody z jiného objektu). Datové složky se kopírují a nemohou být v následníkovi změněny, metody mohou měnit svá těla. Syntax:

```
type Jiny = object (Neco)
    E: boolean;
    procedure A;
end;
```

- 3 Mnohotvarost (schopnost sady objektů být ovládána stejným způsobem – stejnými metodami). Je uplatnitelná u objektů svázaných dědičnými vazbami

- **Kompatibilita objektů**

- **Kompatibilita objektů**
- Přiřazení: předchůdce:=následník

- **Kompatibilita objektů**
- Přiřazení: předchůdce:=následník
- Všechny složky předchůdce jsou přiřazením naplněny, v opačném případě by to nebylo zaručeno

- **Kompatibilita objektů**
- Přiřazení: předchůdce:=následník
- Všechny složky předchůdce jsou přiřazením naplněny, v opačném případě by to nebylo zaručeno
- **Brzká vazba**

- **Kompatibilita objektů**
- Přiřazení: předchůdce:=následník
- Všechny složky předchůdce jsou přiřazením naplněny, v opačném případě by to nebylo zaručeno
- **Brzká vazba**
- U tzv. **statických metod** – volání metod objektů je zařízeno pevnou adresou vzniklou při překladu

- **Kompatibilita objektů**
- Přiřazení: předchůdce:=následník
- Všechny složky předchůdce jsou přiřazením naplněny, v opačném případě by to nebylo zaručeno
- **Brzká vazba**
- U tzv. **statických metod** – volání metod objektů je zařízeno pevnou adresou vzniklou při překladu
- **Pozdní vazba**

- **Kompatibilita objektů**
- Přirazení: předchůdce:=následník
- Všechny složky předchůdce jsou přiřazením naplněny, v opačném případě by to nebylo zaručeno
- **Brzká vazba**
- U tzv. **statických metod** – volání metod objektů je zařízeno pevnou adresou vzniklou při překladu
- **Pozdní vazba**
- V místě volání metody se nesmí použít pevná adresa objektu, musí tam být pouze symbolický odkaz. Ten se naplní v okamžiku přiřazení konkrétního objektu, tedy až **v době běhu**

- **Kompatibilita objektů**
- Přirazení: předchůdce:=následník
- Všechny složky předchůdce jsou přiřazením naplněny, v opačném případě by to nebylo zaručeno
- **Brzká vazba**
- U tzv. **statických metod** – volání metod objektů je zařízeno pevnou adresou vzniklou při překladu
- **Pozdní vazba**
- V místě volání metody se nesmí použít pevná adresa objektu, musí tam být pouze symbolický odkaz. Ten se naplní v okamžiku přiřazení konkrétního objektu, tedy až **v době běhu**
- Metoda, u které je potřeba použít pozdní vazbu, musí být speciálně označena: klíčové slovo **virtual**

- **Tabulka virtuálních metod (VMT)** – pole ukazatelů na metody, je implicitní součástí datových složek objektu

- **Tabulka virtuálních metod** (VMT) – pole ukazatelů na metody, je implicitní součástí datových složek objektu
- Pro naplnění VMT slouží speciální metoda deklarovaná klíčovým slovem `constructor`

- **Tabulka virtuálních metod** (VMT) – pole ukazatelů na metody, je implicitní součástí datových složek objektu
- Pro naplnění VMT slouží speciální metoda deklarovaná klíčovým slovem `constructor`
- Implementace polymorfismu:

- **Tabulka virtuálních metod** (VMT) – pole ukazatelů na metody, je implicitní součástí datových složek objektu
- Pro naplnění VMT slouží speciální metoda deklarovaná klíčovým slovem `constructor`
- Implementace polymorfismu:
- Sada objektů musí být řetězcem dědiců

- **Tabulka virtuálních metod** (VMT) – pole ukazatelů na metody, je implicitní součástí datových složek objektu
- Pro naplnění VMT slouží speciální metoda deklarovaná klíčovým slovem `constructor`
- Implementace polymorfismu:
- Sada objektů musí být řetězcem dědiců
- Musí mít stejné metody (stejná jména, parametry)

- **Tabulka virtuálních metod (VMT)** – pole ukazatelů na metody, je implicitní součástí datových složek objektu
- Pro naplnění VMT slouží speciální metoda deklarovaná klíčovým slovem `constructor`
- Implementace polymorfismu:
- Sada objektů musí být řetězcem dědiců
- Musí mít stejné metody (stejná jména, parametry)
- Musí mít tyto metody jako virtuální

- **Tabulka virtuálních metod (VMT)** – pole ukazatelů na metody, je implicitní součástí datových složek objektu
- Pro naplnění VMT slouží speciální metoda deklarovaná klíčovým slovem `constructor`
- Implementace polymorfismu:
- Sada objektů musí být řetězcem dědiců
- Musí mít stejné metody (stejná jména, parametry)
- Musí mít tyto metody jako virtuální
- Objekty musí mít konstruktory a ty musí být volány na začátku před prvním použitím objektu.

- Ukazatel na objekt – stejně jako na každý jiný bázový typ, kompatibilita ukazatelů má stejný princip jako kompatibilita bázových typů

- Ukazatel na objekt – stejně jako na každý jiný bázevý typ, kompatibilita ukazatelů má stejný princip jako kompatibilita bázevých typů
- Příklad:

```
type ukobjekt = ^Neco;  
    Neco = object(predchudce)  
        ...  
        constructor Start;  
        destructor Done;  
    end;  
var U: UkObjekt;  
begin new(U);  
    U^.Start;  
    ...
```


- Rozšíření procedury `new` – automatické volání konstruktoru:

```
| new(U, Start);
```

- Rozšíření procedury `new` – automatické volání konstruktoru:

```
| new(U, Start);
```

- Druhé rozšíření `new`: použito jako funkce:

```
| U:=new(Neco, Start);
```

- Rozšíření procedury `new` – automatické volání konstruktoru:

```
| new(U, Start);
```

- Druhé rozšíření `new`: použito jako funkce:

```
| U:=new(Neco, Start);
```

- Prvním parametrem je **datový typ**, proměnná `U` může být **libovolným předchůdcem** objektu typu `Neco`

- Rozšíření procedury `new` – automatické volání konstruktoru:

```
| new(U, Start);
```

- Druhé rozšíření `new`: použito jako funkce:

```
| U:=new(Neco, Start);
```

- Prvním parametrem je **datový typ**, proměnná `U` může být **libovolným předchůdcem** objektu typu `Neco`

- Rozšíření procedury `Dispose` – automatické volání destrukturu:

```
| Dispose(U, Done)
```

Objektová implementace ADT

- Pojetí abstraktního typu odpovídá pojetí objektu: hodnoty typu jsou reprezentovány datovými složkami, povolené operace pak metodami objektu

Objektová implementace ADT

- Pojetí abstraktního typu odpovídá pojetí objektu: hodnoty typu jsou reprezentovány datovými složkami, povolené operace pak metodami objektu
- Příklad – implementace seznamu, první typ obyčejný, druhý typ uspořádaný:

```
type TypData = string;  
  UkClen = ^Clen;  
  Clen = record  
    D: TypData;  
    dalsi: UkClen;  
  end;
```



```
seznam = object
  UkPrvni: UkClen;
  Pocet: word;
  constructor Start;
  procedure init;
  procedure add(Prvek: TypData); virtual;
  procedure remove(var Prvek: TypData);
  function Empty: boolean;
end;
```

```
constructor Seznam.Start;  
  begin  
    end;  
procedure Seznam.init;  
  begin UkPrvni:=nil;  
        Pocet:=0;  
  end;  
{... ostatní metody }  
type UspSeznam = object(Seznam)  
  constructor Start;  
  procedure add(Prvek: Typdata); virtual;  
  end;
```

```
constructor UspSeznam.Start;  
  begin  
  end;
```

```
procedure UspSeznam.add(Prvek: Typdata);  
  begin {... zařadí řetězec na správné místo}  
  end;
```

```
var S: Seznam;  
      R: string;  
      T: UspSeznam;
```

Polymorfní podprogramy

- Jsou to podprogramy, jejichž parametrem jsou objekty

- Jsou to podprogramy, jejichž parametrem jsou objekty
- Uvnitř podprogramu se používají výhradně virtuální metody objektů

- Jsou to podprogramy, jejichž parametrem jsou objekty
- Uvnitř podprogramu se používají výhradně virtuální metody objektů
- Podle dosažení skutečného parametru dělá podprogram pokaždé jiné akce příslušející danému objektu

Implementace – pokračování

- Příklad použití seznamu pro uložení a výpis vstupních řetězců:

```
procedure Zpracuj (Obecny: Seznam);  
  begin Obecny.Init;  
    while not eof do begin  
      readln(R);  
      Obecny.add(R);  
    END;  
    while not Obecny.empty do begin  
      Obecny.remove(r);  
      writeln(r)  
    end;  
end;
```

Využití polymorfní procedury

- Využití polymorfní procedury pro různá zpracování vstupu (obyčejný opis, seřazení)

```
begin
    S.Start;          Zpracuj (S) ;
    T.Start;
    Zpracuj (T) ;
end.
```

Příklad polymorfních objektů

Příklad polymorfních objektů

- Příklad polymorfních objektů zásobník/fronta:

```
type neco = object
    constructor Start;
    procedure Init; virtual;
    procedure Vloz(A: string); virtual;
    function Empty: boolean; virtual;
    function Vyjmi: string; virtual;
end;

dalsi = object(neco)
    constructor Start;
    procedure Init; virtual;
    procedure Vloz(A: string); virtual;
    function Empty: boolean; virtual;
    function Vyjmi: string; virtual;
end;
```

Příklad polymorfních objektů

```
procedure neco.Vloz(A: string);  
begin ... vložení do zásobníku  
end;  
  
procedure dalsi.Vloz(A: string);  
begin ... vložení do fronty  
end;  
  
var A:neco; B:dalsi;
```

Příklad polymorfních objektů

```
procedure Delej (var X:neco);  
  var R:string;  
begin X.Init;  
  while not eof do begin  
    readln(R);  
    X.Vloz(R);  
  end;  
  while not X.Empty do begin  
    R:=X.Vyjmi;  
    writeln('--', R)  
  end  
end;  
begin A.Start; Delej(A);  
  {alternativně: B.Start; Delej(B);}  
end.
```