A0B36PRI - PROGRAMOVÁNÏ

Procedurální programování, rekurze

Jazyk JAVA

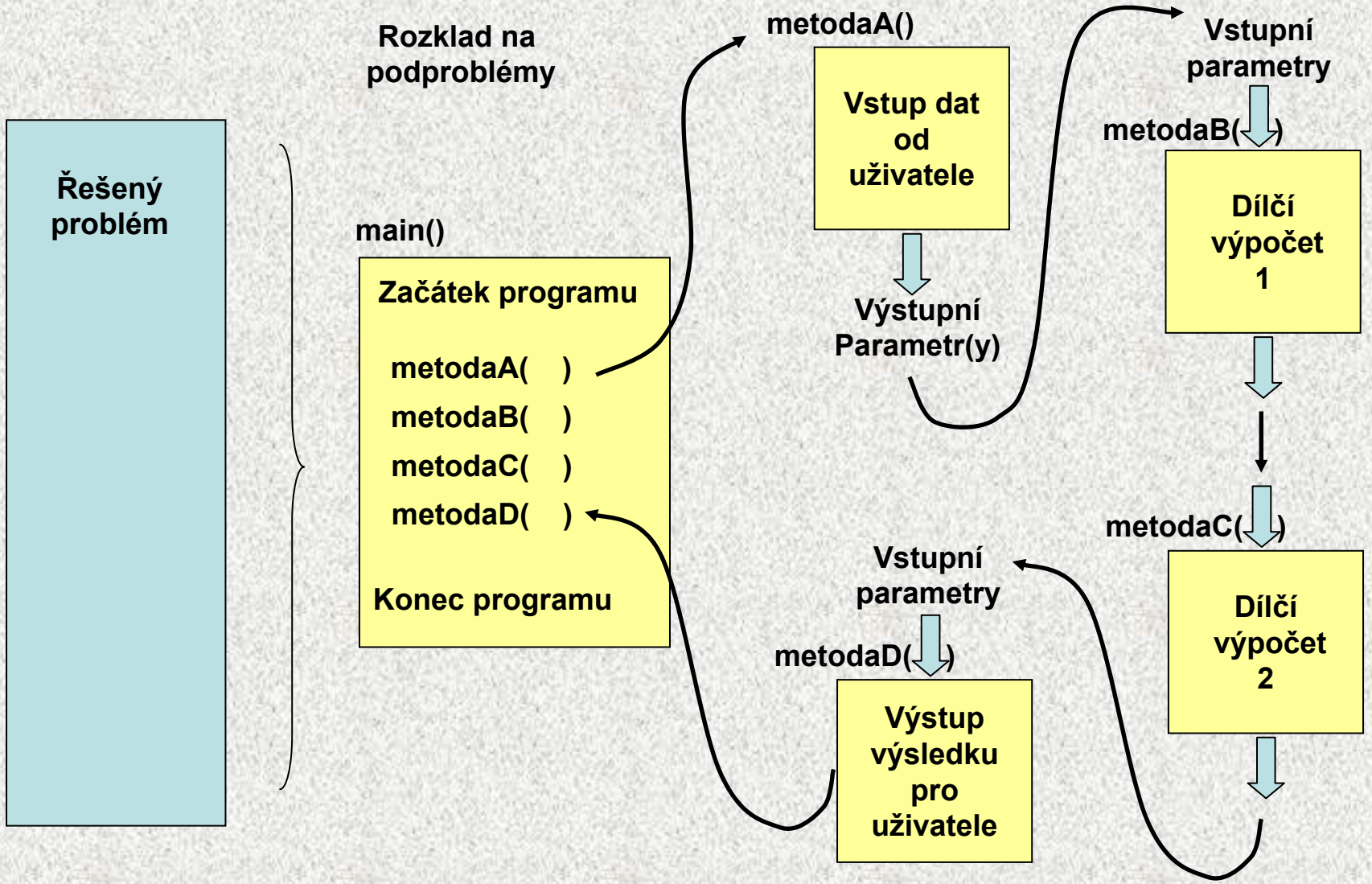
České vysoké učení technické Fakulta elektrotechnická

v1.01

Obsah

- Rozklad problému na podproblémy
- Hra NIM
- Rekurzivní algorimus
- Rekurze a rozklad problému na podproblémy
- Hanojské věže
- Obecně k rekurzivitě
- Od rekurze k iteraci
- Programovací styly
- Naivní styl
- Procedurální styl

Rozklad problému na podproblémy



Rozklad problému na podproblémy

- Postupný návrh programu rozkladem problému na podproblémy
  - zadaný problém rozložíme na podproblémy
  - pro řešení podproblémů zavedeme abstraktní příkazy
  - s pomocí abstraktních příkazů sestavíme hrubé řešení
  - abstraktní příkazy realizujeme pomocí metod (funkcí, procedur)
- Rozklad problému na podproblémy ilustrujme na příkladu hry NIM
- Pravidla:
  - hráč zadá počet zápalek (např. od 15 do 35)
  - pak se střídá se strojem v odebírání; odebrat lze 1, 2 nebo 3 zápalky,
  - prohraje ten, kdo odebere poslední zápalku.
- Dílčí podproblémy:
  - zadání počtu zápalek
  - odebrání zápalek hráčem
  - odebrání zápalek strojem

Hra NIM

- Hrubé řešení:

```
int pocet;
boolean stroj = false;

// zadání počtu zápalek
do {
  if ( stroj ) "odebrání zápalek strojem"
   else "odebrání zápalek hráčem"
  stroj = !stroj;
} while ( pocet>0 );
if ( stroj ) "vyhrál stroj"
else "vyhrál hráč"
```

- Podproblémy „zadání počtu zápalek“, „odebrání zápalek strojem“ a „odebrání zápalek hráčem“ budeme realizovat metodami, proměnné *pocet* a *stroj* pro ně budou statickými (čili nelokálními) proměnnými

Hra NIM

```
public class Nim {
  static int pocet;     // aktuální počet zápalek
  static boolean stroj; // =true znamená, že bere počítač

  public static void main(String[] args) {
    zadaniPoctu();
    stroj = false;      // zacina hrac
    do {
      if (stroj) bereStroj();
      else       bereHrac();
      stroj = !stroj;
    } while (pocet>0);
    if (stroj) Sys.pln( "vyhrál jsem" );
    else       Sys.pln( "vyhrál jste, gratuluji" );
  }

  static void zadaniPoctu() { … }
  static void bereHrac() { … }
  static void bereStroj() { … }
}
```

Hra NIM

```
static void zadaniPoctu() {
    Scanner sc = new Scanner(System.in);
    do {
      System.out.prinln("Zadejte počet zápalek (od 15 do 35)");
      pocet = sc.nextInt();
    } while (pocet<15 || pocet>30);
  }

  static void bereHrac() {
    Scanner sc = new Scanner(System.in);
    int x; boolean chyba;
    do {
      chyba = false;
      System.out.println("Počet zápalek " + pocet);
      System.out.println("Kolik odeberete");
      x = sc.nextInt();
      if (x < 1) {
        System.out.println("Prilis malo" ); chyba = true;
      }
      else
      if (x > 3 || x > pocet) {
        System.out.println("Prilis mnoho"); chyba = true;
      }
    } while (chyba);
    pocet -= x;
  }
```

Hra NIM

- Pravidla pro odebírání zápalek strojem, která vedou k vítězství (je-li to možné):
  - počet zápalek nevýhodných pro protihráče je 1, 5, 9, atd., obecně $4n+1$, kde $n > 0$,
  - stroj musí z počtu p zápalek odebrat x zápalek tak, aby platilo $p - x = 4n + 1$
  - z tohoto vztahu po úpravě a s ohledem na omezení pro x dostaneme $x = (p - 1) \bmod 4$
  - vyjde-li x=0, znamená to, že okamžitý počet zápalek je pro stroj nevýhodný a bude-li protihráč postupovat správně, stroj prohraje.

```
static void bereStroj() {
   System.out.println("Počet zápalek " +pocet);
   int x = (pocet-1) % 4;
   if (x == 0) x = 1;
   System.out.println("Odebírám " +x);
   pocet -= x;
 }
```

Hra NIM (bez globálních prom.)

```
public class Nim {

  public static void main(String[] args) {
     int pocet;                  // aktuální počet zápalek
     boolean stroj = false;      // =true => bere počítač
    pocet = zadaniPoctu();
     do {
       if (stroj) pocet = bereStroj(pocet);
       else       pocet = bereHrac(pocet);
       stroj = !stroj;
     } while (pocet>0);
     if (stroj)
           System.out.println("Vyhrál jsem");
     else   System.out.println("Vyhrál jste, gratuluji");
  }

  static int zadaniPoctu() { … }
  static int bereHrac(int pocetZapalek) { … }
  static int bereStroj(int pocetZapalek) { … }
}
```

Hra NIM (bez globálních prom.)

```
static int bereHrac(int pocetZapalek) {
   Scanner sc = new Scanner(System.in);
  int x; boolean chyba;
  do {
    chyba = false;
    System.out.println("Počet zápalek " + pocet);
    System.out.println("Kolik odeberete");
    x = sc.nextInt();
    if (x < 1) {
      System.out.println("Prilis malo" ); chyba = true;
    }
    else
    if (x > 3 || x > pocetZapalek) {
      System.out.println("Prilis mnoho"); chyba = true;
    }
  } while (chyba);
return(pocetZapalek-x);
}
```

Hra NIM (bez globálních prom.)

```
static int zadaniPoctu() {
   int pocetZapalek;
    Scanner sc = new Scanner(System.in);
     do {
       System.out.prinln("Zadejte počet zápalek (od 15 do
  35)");
       pocetZapalek = sc.nextInt();
     } while (pocetZapalek < 15 || pocetZapalek > 30);
    return(pocetZapalek);
}

static int bereStroj(int pocetZapalek) {
    System.out.println("Počet zápalek " +pocet);
    int x = (pocet-1) % 4;
    if (x == 0) x = 1;
    System.out.println("Odebírám " +x);
    return(pocetZapalek-x);
  }
```

Od rekurze k iteraci

- Řadu rekurzívních algoritmů lze nahradit iteračními, které počítají výsledek „zdola nahoru“, tj, od menších (jednodušších) dat k větším (složitějším)

  $n! = 1$ pro $n \leq 1$

  $n! = n*(n-1)!$ pro $n > 1$

```
static int fakt(int n) {
    if (n<=
     return
      }
```

```
static int fakt(int n) {
                  return 1;
                  rn n*fakt(n-1);
```

```
static int fakt(int n) {
   int f = 1;
   while (n>1){f *= n; n--;
   }
      return f;
    }
```

```
static int fakt(int n) {
  return n<=1?1:n*fakt(n-1); // ternární operátor
}
```

Rekurze

- Rekurzivní funkce (procedury) jsou přímou realizací rekurzivních algoritmů
- Rekurzivní algoritmus předepisuje výpočet „shora dolů“ v závislosti na velikosti (složitosti) vstupních dat:
    - pro nejmenší (nejjednodušší) data je výpočet předepsán přímo
    - pro obecná data je výpočet předepsán s využitím téhož algoritmu pro menší (jednodušší) data
- Výhodou rekurzivních funkcí (procedur) je jednoduchost a přehlednost
- Nevýhodou může být časová náročnost způsobená např. zbytečným opakováním výpočtu

- Řadu rekurzívních algoritmů lze nahradit iteračními, které počítají výsledek „zdola nahoru“, tj, od menších (jednodušších) dat k větším (složitějším)

- Pokud algoritmus výpočtu „zdola nahoru“ nenajdeme (např. při řešení problému Hanojských věží), lze rekurzivitu odstranit pomocí tzv. zásobníku

Rekurzivní algorimus

- Rekurzivní algoritmus v některém kroku volá sám sebe
- Rekurzivní metoda v některém příkazu volá sama sebe (i nepřímo)
- Příklad: nsd(x, y)

  je-li $x = y$, pak $nsd(x, y) = x$

  je-li $x > y$, pak $nsd(x, y) = nsd(x-y, y)$

  je-li $x < y$, pak $nsd(x, y) = nsd(x, y-x)$
- Rekurzivní funkce:

```
static int nsd(int x, int y) {
   if (x==y) return x;
   else if (x>y) return nsd(x-y, y);
        else     return nsd(x, y-x);
}
```

- Jiný příklad – faktoriál:

  $n! = 1$ pro $n \leq 1$

  $n! = n*(n-1)!$ pro $n > 1$
- Rekurzivní funkce:

```
static int fakt(int n) {
   if (n<=0) return 1;
   return n*fakt(n-1);
}
```

Rekurze a rozklad problému na podproblémy

- Program, který přečte posloupnost čísel zakončenou nulou a vypíše ji obráceně
- Rozklad problému:
  - zavedeme abstraktní příkaz *přečti a obrať posloupnost*
  - příkaz rozložíme do tří kroků:
    - *přečti číslo*
    - if (*přečtené číslo není nula*) *přečti a obrať posloupnost*
    - *vypiš číslo*
- Řešení:

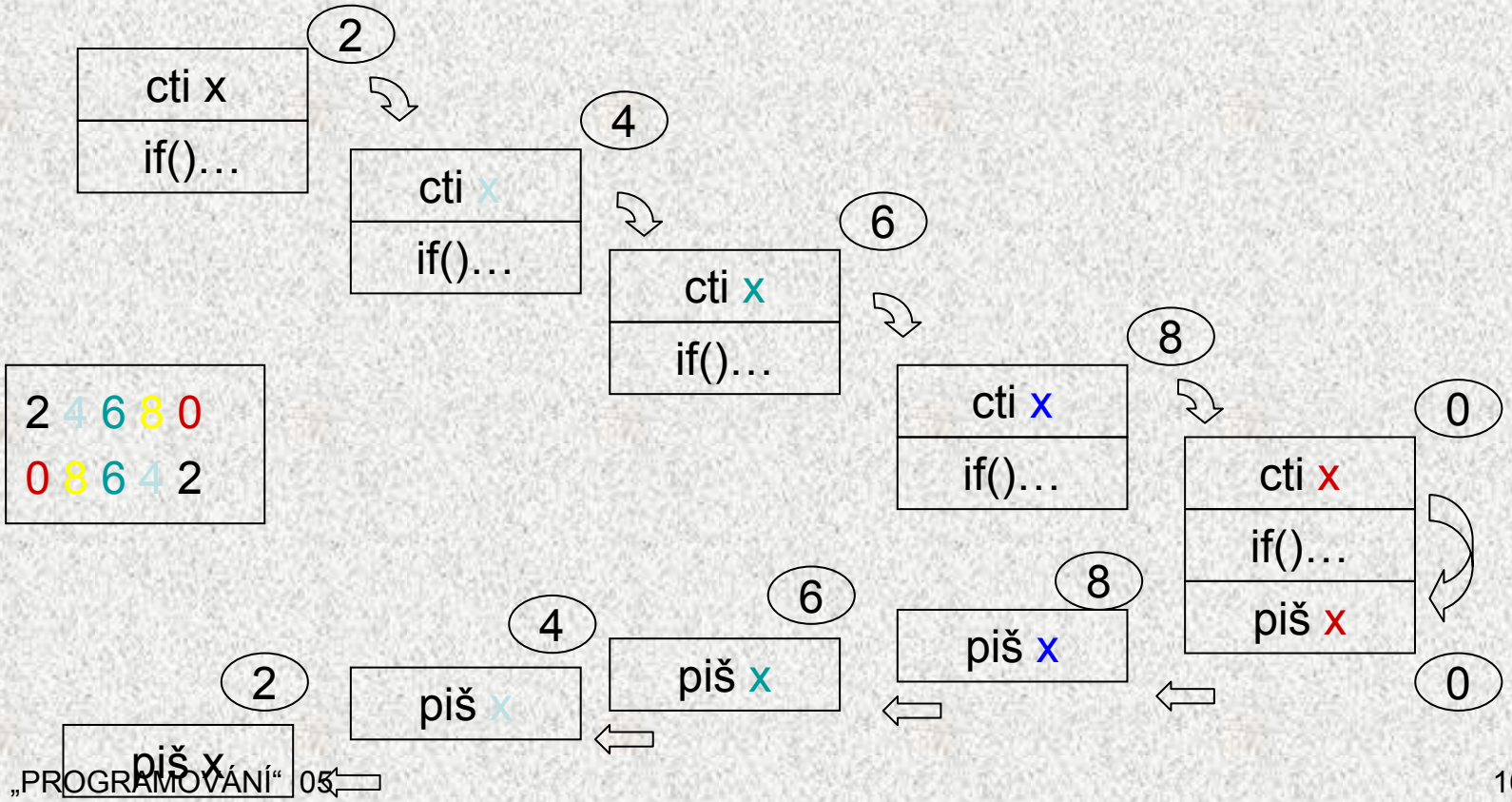
```
public static void main(String[] args) {
   obrat();
 }

static void obrat() {
   int x = Sys.readInt();
   if (x!=0) obrat();
   Sys.pln(x);
 }
```

Příklad rekurze „*Obrať posloupnost*“

„obrať posloupnost“

- *přečti číslo*
- if (*přečtené číslo není nula*) *„obrať posloupnost, tj. zbytek“*
- *vypiš číslo*



složitější

Příklad rekurze - obrat()

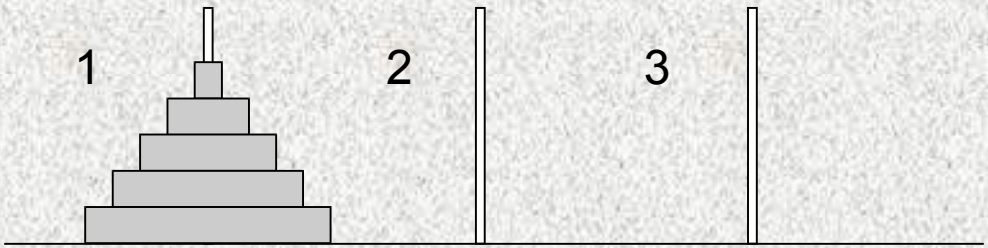
- Řešení:

```
public class Obrat {
static Scanner sc = new Scanner(System.in);

public static void main(String[] args) {
System.out.println("zadejte ……..      zakončenou nulou");
 obrat();
  }
 static void obrat() {
 int x = sc.nextInt();                  // načtení
 if (x!=0) obrat();                     // otočení zbytku
 System.out.print(x + " ");             // výpis uloženého
  }
}
```

složitější

Příklad rekurze - Hanojské věže



- Zavedeme abstraktní příkaz
  - *přenes_věž*(*n*,*1*,*2*,*3*)

který interpretujeme jako "přenes *n* disků z jehly *1* na jehlu *2* s použitím jehly *3*".

- Pro *n*>0 lze příkaz rozložit na tři jednodušší příkazy
  - *přenes_věž*(*n*-1,*1*,*3*,*2*)
  - "*přenes disk z jehly 1 na jehlu 2*",
  - *přenes_věž*(*n*-1,*3*,*2*,*1*)

složitější

Příklad rekurze - Hanojské věže

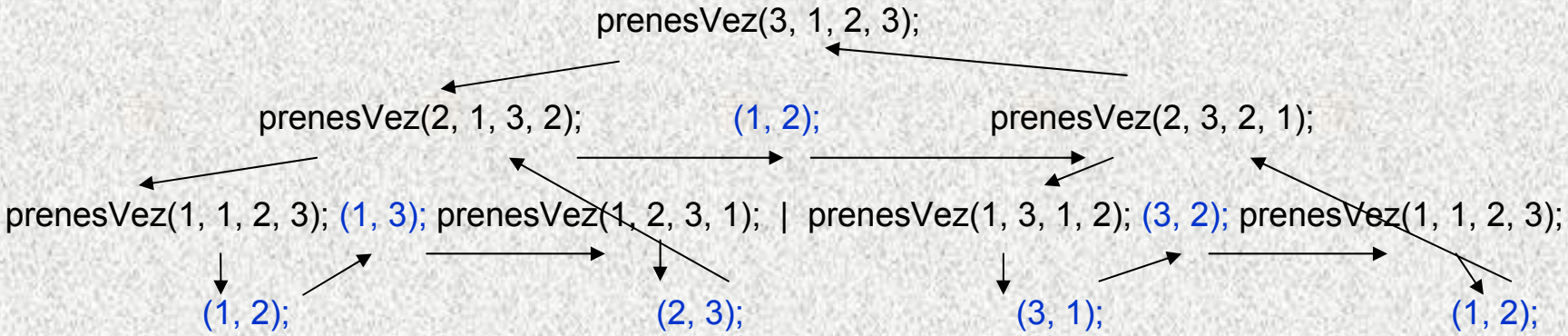
```
public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 System.out.println("zadejte výšku věže");
 int pocetDisku = sc.nextInt();
 prenesVez(pocetDisku, 1, 2, 3);
}
```

```
3
přenes disk z 1 na 2
přenes disk z 1 na 3
přenes disk z 2 na 3
přenes disk z 1 na 2
přenes disk z 3 na 1
přenes disk z 3 na 2
přenes disk z 1 na 2
```

```
static void prenesVez
            (int vyska, int odkud, int kam, int pomoci) {
if (vyska>0) {
 prenesVez(vyska-1, odkud, pomoci, kam);
 System.out.println("přenes disk z "+odkud+" na "+kam);
 prenesVez(vyska-1, pomoci, kam, odkud);
   –      }
}
```

složitější

Příklad rekurze - Hanojské věže




```
prenesVez(int vyska, int odkud, int kam, int pomoci) {
     if (vyska>0) {
      prenesVez(vyska-1, odkud, pomoci, kam);
      System.out.println("přenes disk z "+odkud+" na "+kam);
      prenesVez(vyska-1, pomoci, kam, odkud);
       }
     }
   }
```

```
3
přenes disk z 1 na 2
přenes disk z 1 na 3
přenes disk z 2 na 3
přenes disk z 1 na 2
přenes disk z 3 na 1
přenes disk z 3 na 2
přenes disk z 1 na 2
```

složitější

Obecně k rekurzivitě

- Rekurzivní metody jsou přímou realizací rekurzivních algoritmů
- Rekurzivní algoritmus předepisuje výpočet „shora dolů“ v závislosti na velikosti (složitosti) vstupních dat:
  - pro nejmenší (nejjednodušší) data je výpočet předepsán přímo
  - pro obecná data je výpočet předepsán s využitím téhož algoritmu pro menší (jednodušší) data
- Výhodou rekurzivních metod je jednoduchost a přehlednost
- Nevýhodou může být časová náročnost způsobená např. zbytečným opakováním výpočtu

Příklad: Fibonacciho posloupnost:

$f_0 = 0$
$f_1 = 1$
$f_i = f_{i-1} + f_{i-2}$ pro $i > 1$

```
static int fibonacci(int i) {
    if (i==0) return 0;
    if (i==1) return 1;
    return fibonacci(i-1)+fibonacci(i-2);
}
```

Od rekurze k iteraci

- Řadu rekurzívních algoritmů lze nahradit iteračními, které počítají výsledek „zdola nahoru“, tj, od menších (jednodušších) dat k větším (složitějším)

```
static int fakt(int n) {
    int f = 1;
    while (n>1) {
    f *= n; n--;
    }
  return f;
}

static int fib(int n) {
    int i, fn = 0, fp, fpp;
    if (n>0) {
      fp = fn; fn = 1;
      for (i=2; i<n; i++) {
        fpp = fp; fp = fn; fn = fp + fpp;
      }
    }
  return fn;
  }
```

- Pokud algoritmus výpočtu „zdola nahoru“ nenajdeme (např. při řešení problému Hanojských věží), lze rekurzivitu odstranit pomocí tzv. zásobníku

Programovací styly

- Na příkladu programu simulujícího čítač si ukážeme programovací styly:
  - naivní
  - procedurální
  - objektově orientovaný

    (viz přednáška „Úvod do objektově orientovaného programování")
- Příklad komunikace programu:

  Hodnota = 10

  0) Konec
  1) Zvětši
  2) Zmenši
  3) Nastav

  Vase volba: 1

  Hodnota = 11
  0) Konec
  1) Zvětši
  2) Zmenši
  3) Nastav
  ...

Naivní styl

- Jednoduché úlohy lze řešit přímočaře:

```
package pri05;
import java.util.*;
public class Citac1 {
  final static int pocHodn = 0;
  static int hodn, volba;
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    hodn = pocHodn;
    do {
      System.out.prinln( "Hodnota = " + hodn );
      System.out.println( "0) Konec\n1) Zvětši\n2) Zmenši\n3) Nastav" );
      System.out.print( "Vaše volba: " );
      volba = sc.nextInt();
      switch ( volba ) {
        case 0: break;
        case 1: hodn++; break;
        case 2: hodn--; break;
        case 3: hodn = pocHodn; break;
        default: System.out.println( "nedovolená volba" );
      }
    } while ( volba>0 );
    System.out.println( "Konec" );
  }
}
```

Procedurální styl

- Připomeňme hlavní zásady:
  - Zadaný problém se snažíme rozložit na podproblémy
  - Pro řešení podproblémů zavádíme abstraktní příkazy, které nejprve specifikujeme a pak realizujeme pomocí metod
- Aplikováno na "čítač" identifikujeme tyto dílčí podproblémy:
  - komunikace s uživatelem, jejímž výsledkem je kód požadované operace
  - provedení požadované operace s čítačem
- Řešení:

```
public class Citac2 {
  final static int POC_HODN = 0; // třídní konstanta
  static int hodn;               // třídní proměnná
  static void operace(int op) {  // pro všechny 3 operace
    switch (op) {
      case 1: hodn++; break;
      case 2: hodn--; break;
      case 3: hodn = POC_HODN; break;
    }
  }
```

Procedurální styl

```
static int menu(){
   Scanner sc = new Scanner(System.in);
   int volba;
   do {
      System.out.println( "0. Konec" );
      System.out.println( "1. Zvětši" );
      System.out.println( "2. Zmenši" );
      System.out.println( "3. Nastav" );
      System.out.print( "Vaše volba: " );
      volba = sc.nextInt();
      if ( volba<0 || volba >3 ) {
        System.out.println( "nedovolená volba" );
        volba = -1;
      }
    } while ( volba<0 );
    return volba;
  }
```

Procedurální styl

```
public static void main(String[] args) {
    int volba;
    hodn = POC_HODN;
    do {
      System.out.println("hodnota = " +hodn);
      volba = menu();
      if (volba>0) operace(volba);
    } while (volba>0);
    System.out.println("Konec");
 }
```

- Poznámky:
  - procedurální řešení čítače (jediná procedura pro všechny operace, třídní proměnná pro hodnotu, třídní konstanta udávající počáteční hodnotu) je nedokonalé (proč?)
  - čítač je typ, pro který jsou definovány určité operace (realizované metodami) a jehož implementace (proměnná pro hodnotu) by neměla být volně přístupná
  - tedy pro realizaci čítače je vhodnější objektově orientovaný styl (nebo lokální statická proměnná – viz úvod do jazyka "C").

Procedurální programování a rekurze

Jazyk JAVA

Konec

České vysoké učení technické Fakulta elektrotechnická