5. přednáška - Rozklad problému na podproblémy

- Obsah přednášky:
  - Rozklad problému na podproblémy.
  - Rekurze.

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
- 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;
"zadani_poctu_zapalek"
do {
      if ( stroj ) "odebrani_zapalek_strojem"
      else "odebrani_zapalek_hracem"
      stroj = !stroj;
} while ( pocet>0 );
if ( stroj ) "vyhral_stroj"
else "vyhral_hrac"
```

- Podproblémy „zadani_poctu_zapalek“, „odebrani_zapalek_strojem“ a „odebrani_zapalek_hracem“ 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) System.out.println( "vyhral stroj" );
      else System.out.println( "vyhral 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.println( "zadejte pocet zapalek (15 - 35)");
      pocet = sc.nextInt();
   } while (pocet<15 || pocet>30);
}
```

Hra NIM

```
static void bereHrac() {
   int x; boolean chyba;
   Scanner sc = new Scanner(System.in);
   do {
      chyba = false;
      System.out.println( "pocet zapalek " + 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 4*n*+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( "pocet zapalek " + pocet);
   int x = (pocet-1) % 4;
   if (x==0)
      x = 1;
   System.out.println( "odebiram " + x);
   pocet -= x;
}
```

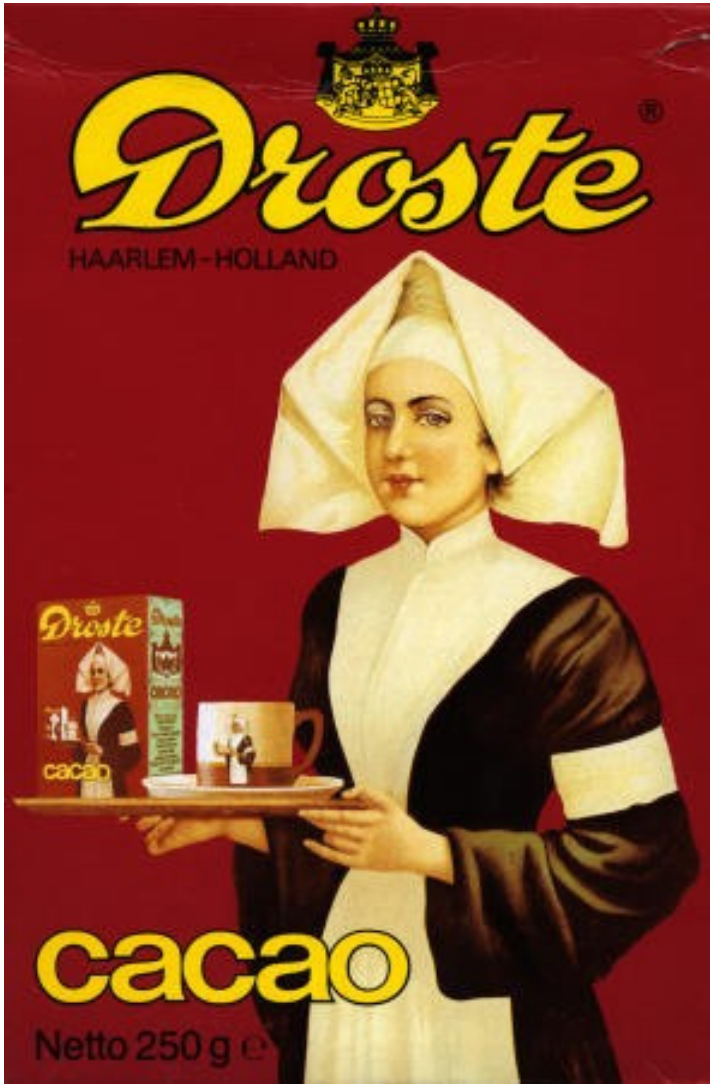
Rekurze

- Rekurzivní algoritmus volá v průběhu svého běhu sám sebe.

  Příklad: výpočet faktoriálu: n!

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

  $n! = n \cdot$ **$(n-1)!$** , pro $n > 1$



Faktoriál pomocí rekurze a iterace

- Rekurze

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

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

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

Faktoriál pomocí rekurze a iterace

- Iterace

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

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

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

```
static int f
  return n<=
}
```

operátor

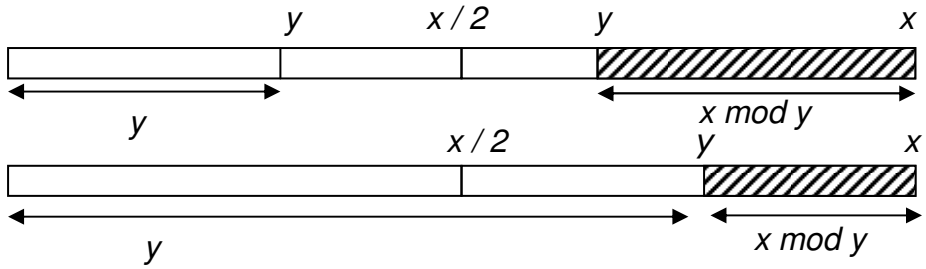
Iterační algoritmus – NSD() - připomínka

Bez použití rekurze, iteračně

```
static int nsd(int x, int y) {
  int zbytek;
  while( y != 0 ) {
    zbytek = x % y; x = y; y = zbytek;
  }
  return x;
}
```

- **Kolikrát se provede tělo cyklu while ?**
- Platí: Je-li $x \geq y$ (> 0), pak x mod $y < x/2$

  (př.: *55 88, 88 55, 55 33, 33 22, 22 11, 11 11, 11 0*)
- <u>Důkaz:</u>
  - buď je $y \leq x/2$
  - nebo je $y > x/2$



- Nechť ***n*** je počáteční hodnota ***y***. Každé dva průchody cyklem se ***y*** zmenší na polovinu, takže na hodnotu 0 dospěje nejpozději po $2.\log_2 n$ průchodech.

Rekurzivní algoritmus – NSD() I

- Platí: je-li x, y > 0, pak 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 $nsd(x,y) = nsd(x, y\%x)$

```
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);
}
```

Rekurze

```
static int nsd(int x, int y) {
  int zbytek;
  while( y != 0 ) {
    zbytek = x % y; x = y; y = zbytek;
  }
  return x;
}
```

Iterace

Rekurze a rozklad problému na podproblémy

- Příklad:

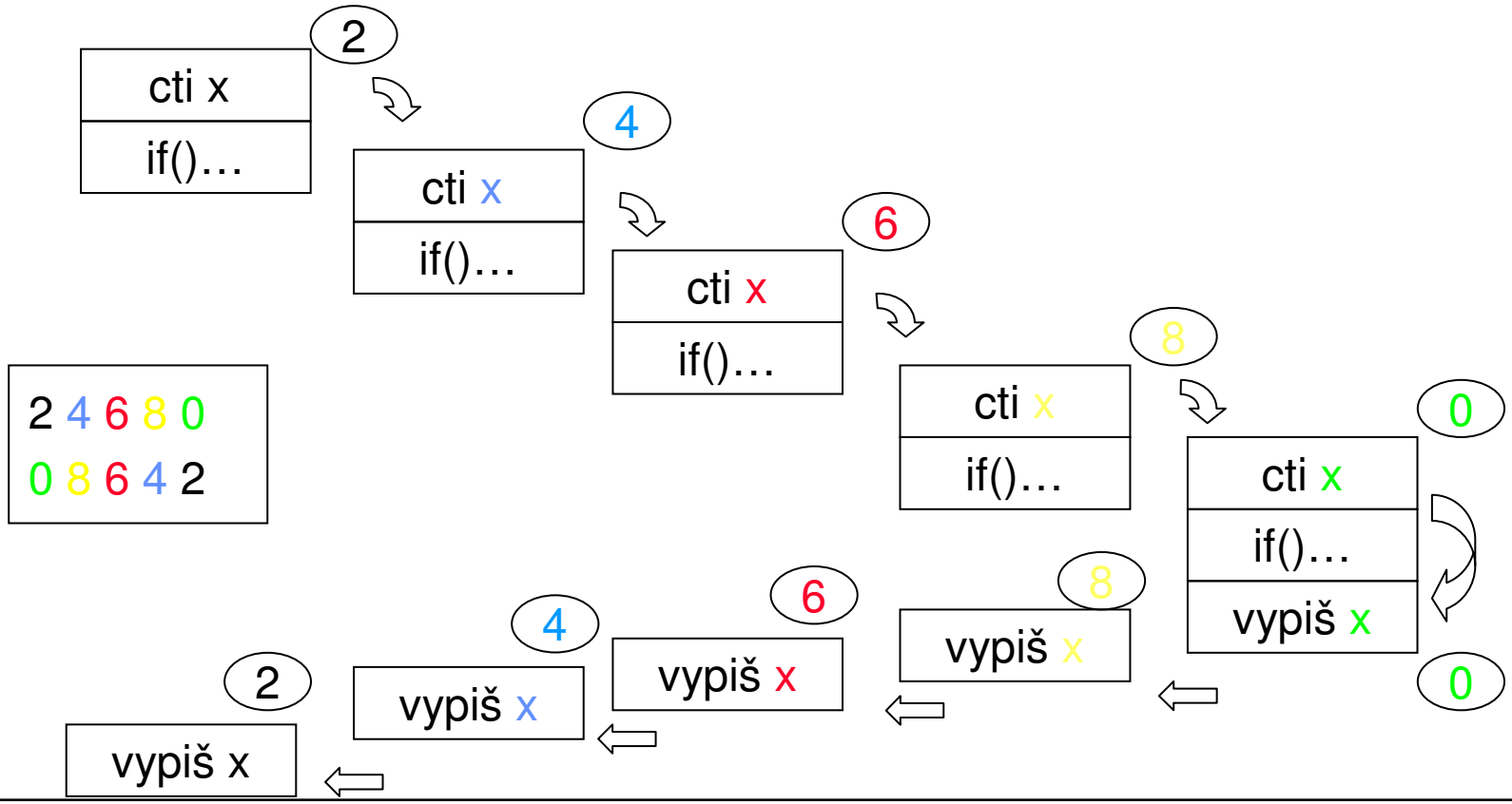
Program, který přečte posloupnost čísel zakončenou nulou a vypíše ji obráceně

- Rozklad problému:
  - zavedeme abstraktní příkaz „*obrať posloupnost*“
  - příkaz rozložíme do tří kroků:
    - *přečti číslo* (***“a ulož si ho”***)
    - if (*přečtené číslo není nula*) „*obrať posloupnost*“ (***“zbytek!!”***)
    - *vypiš číslo* (***“uložené”***)

Příklad rekurze „*Obrať posloupnost*“

„obrať posloupnost“

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

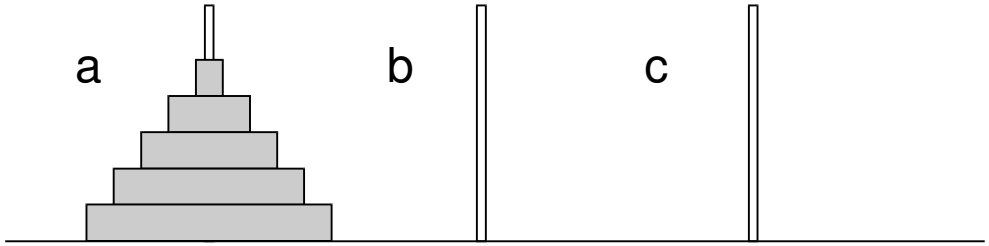


Příklad rekurze - obrat()

- Řešení:

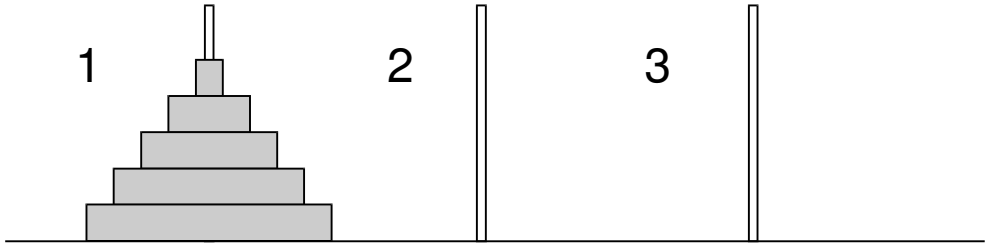
```
public class Obrat {
   static Scanner sc = new Scanner(System.in);
   public static void main(String[] args) {
      System.out.println("zadejte ……… zakončenou nulou");
      obrat();
   }

   static void obrat() {
      int x = sc.nextInt();                    // načtení
      if (x!=0) obrat();                       // otočení zbytku
      System.out.print(x + " ");               // výpis uloženého
   }
}
```

Příklad rekurze - Hanojské věže



- Úkol: přemístit disky na druhou jehlu s použitím třetí pomocné jehly, přičemž musíme dodržovat tato pravidla:
  - v každém kroku můžeme přemístit pouze jeden disk, a to vždy z jehly na jehlu (disky nelze odkládat mimo jehly),
  - není možné položit větší disk na menší.

Příklad rekurze - Hanojské věže



- Zavedeme abstraktní příkaz

  `prenes_vez(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

```
1. prenes_vez(n-1,1,3,2)
2. "přenes disk z jehly 1 na jehlu 2"
3. prenes_vez(n-1,3,2,1)
```

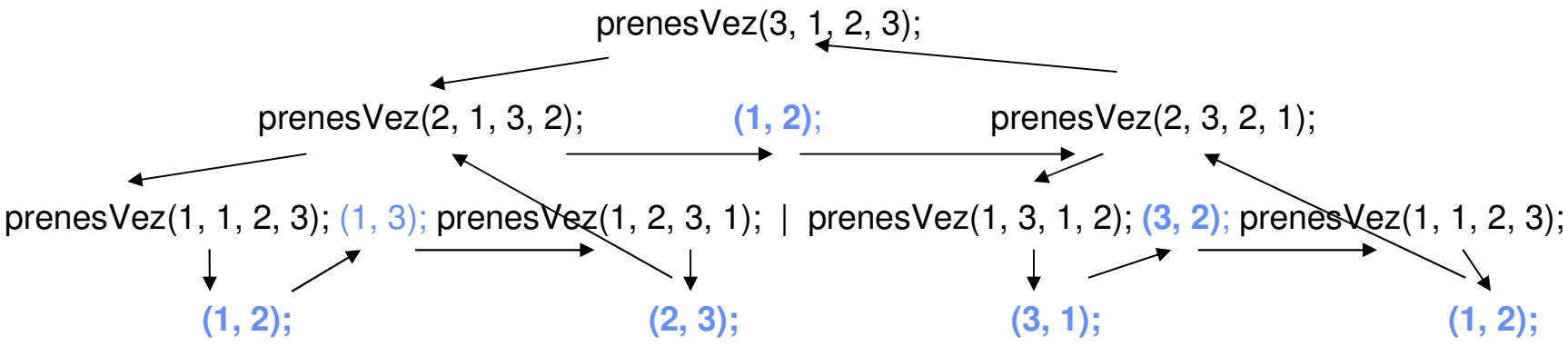
Příklad rekurze - Hanojské věže

```
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);
   }
}
```

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

Obecně k rekurzivitě

- 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

Fibonacciho posloupnost - historie

- Pingala (Chhandah-shāstra, the Art of Prosody, 450 or 200 BC)
- **Leonardo Pisano** (Leonardo z Pisy), známým také jako Fibonacci (cca 1175–1250) - králíci
- Henry E. Dudeney (1857 - 1930) - krávy
- „Jestliže každá kráva vyprodukuje své první tele (jalovici) ve věku dvou let a poté každý rok jednu další jalovici, kolik budete mít krav za 12 let, jestliže Vám žádná nezemře?“

rok	počet krav (jalovic)
1	1
2	1
3	2
4	3
5	5
6	8
...	
12	144
...	
50	20 365 011 074 (20 miliard)

počet krav = počet_krav_vloni + počet_narozených
// odpovídá počtu krav předloni)

$f_n = f_{n-1} + f_{n-2}$

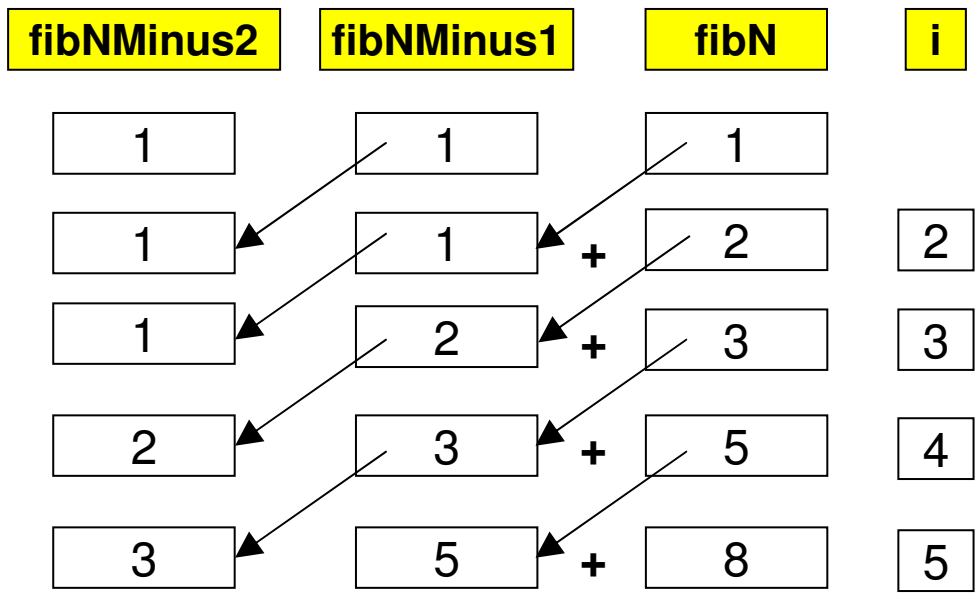
Fibonacciho posloupnost - iteračně

- Platí:

  $f_0 = 1$

  $f_1 = 1$

  $f_n = f_{n-1} + f_{n-2}$, pro $n > 1$



Iteračně:

```
static int fib(int n) {
   int i,  fibNMinus2=1, fibNMinus1=1, fibN=1;
   for (i=2; i<=n; i++) {
      fibNMinus2 = fibNMinus1;
      fibNMinus1 = fibN;
      fibN = fibNMinus1 + fibNMinus2;
   }
   return fibN;
}
```

Složitost:

3∗n

Fibonacciho posloupnost rekurzivne

- Platí:

$f_0 = 1$
$f_1 = 1$
$f_n = f_{n-1} + f_{n-2}$, pro $n > 1$

Rekurzivní funkce:

```
static int fib(int i) {
  if (i<2)
      return 1;
  return fib(i-1)+fib(i-2);
}
```

Rekurze je hezká - zápis „odpovídá“ rekurentní definici.
Je ale i efektivní?

Složitost výpočtu Fibonacciho čísla

- Iterační metoda: 3*n
- **Rekurzivní metoda**:

```
static int fib(int i) {
   if (i<2)
      return 1;
   return fib(i-1)+fib(i-2);
}
```

příklad pro fib(10):

fib(10)

fib(9) + fib(8)

fib(8) + fib(7) fib(7) + fib(6)

fib(7) + fib(6) fib(6) + fib(5) fib(6) + fib(5) fib(5) + fib(4)

Jaká to je složitost?

Složitost výpočtu Fibonacciho čísla

- Iterační metoda: 3*n
- **Rekurzivní metoda**:

```
static int fib(int i) {
   if (i<2)
      return 1;
   return fib(i-1)+fib(i-2);
}
```

příklad pro fib(10):

fib(10)

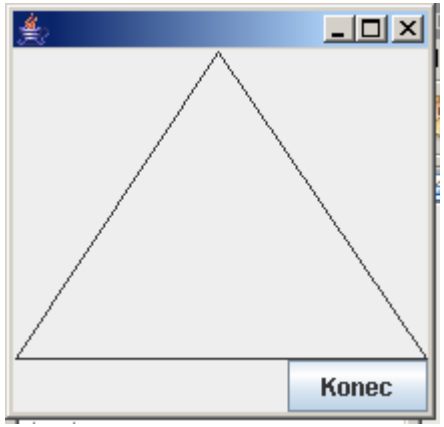
fib(9) + fib(8)

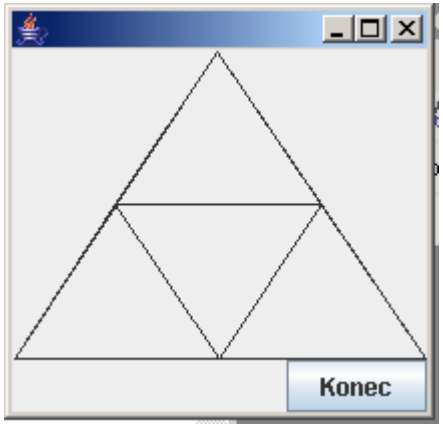
fib(8) + fib(7) fib(7) + fib(6)

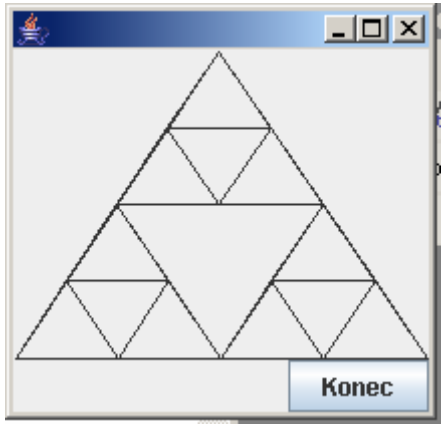
fib(7) + fib(6) fib(6) + fib(5) fib(6) + fib(5) fib(5) + fib(4)

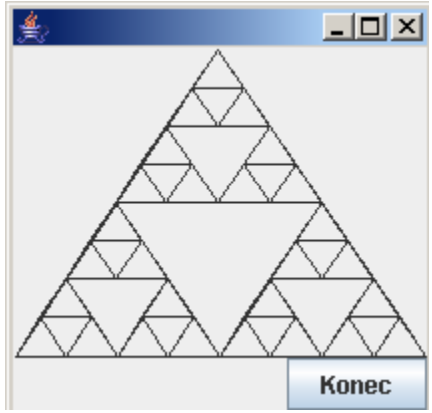
Složitost je exponenciální!!!

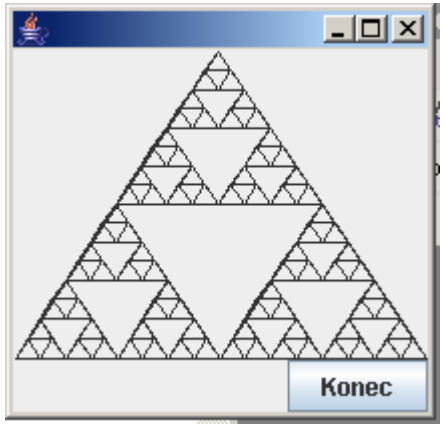
Další příklady rekurze - fraktály

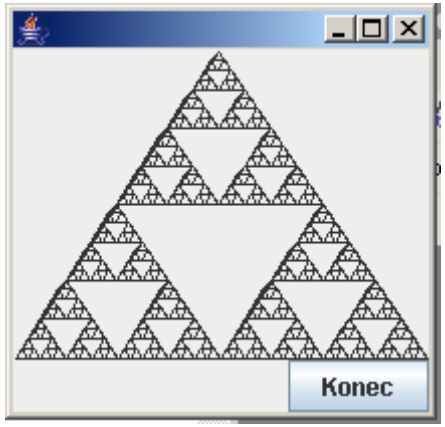












Příklad rekurze, základní schema – součin

```
public static void main(String[] args) {
   int x, y;
   ...;
   System.out.println(" " + souI(x, y) + souR(x, y));
}
static int souR(int s, int t) {
   int souR;
   if (s > 0)
      souR = souR(s - 1, t) + t;
   else
      souR = 0;
   return souR;
}
static int souI(int s, int t){
   int souI=0;
   for (int i = 0; i < s; i++) souI=souI+t;
   return souI;
}
```

pro zájemce

Rozklad na prvočinitele

- Rozklad přirozeného čísla n na součin prvočísel

  Řešení:

  - dělit 2, pak 3, atd. , a dalšími prvočísly, … n-1
  - každé dělení beze zbytku dodá jednoho prvočinitele

  Příklad:

  60/2=>30/2=>15/3=>5/5

  60 má prvočinitele 2, 2, 3, 5

pro zájemce

Rozklad na prvočinitele - iterací

```
public class PrvociniteleIter {
   static int rozklad(int x, int d) {
      while (d < x && x % d != 0) d++;
      System.out.print(d + " ");
      return d;
   }
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.println("zadejte přirozené číslo: ");
      int x = sc.nextInt();
      if (x < 1) {
         System.out.println("číslo není přirozené");
         System.exit(0);
      }
      int d = 2;
      while (d <= x) {
         d = rozklad(x, d);
         x = x/d;
      }
```

```
zadejte přirozené číslo: 144
2 2 2 2 3 3
```

pro zájemce

Rozklad na prvočinitele - rekurzí

```
public class Prvocinitele {
   static void rozklad(int x, int d) {
      if (d <= x) {
         while (d < x && x%d != 0) d++;
         System.out.print(d + " ");
         rozklad(x/d, d);
      }
   }
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.println("zadejte přirozené číslo: ");
      int x = sc.nextInt();
      if (x < 1) {
         System.out.println("číslo není přirozené");
         System.exit(0);
      }
      rozklad(x, 2);
```

```
zadejte přirozené číslo: 144
2 2 2 2 3 3
```

pro zájemce