A0B36PRI - PROGRAMOVÁNÏ

Metody (procedury, funkce)

Jazyk JAVA

České vysoké učení technické Fakulta elektrotechnická

v1.01

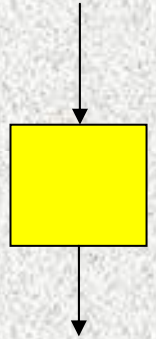
Obsah

- Bloková struktura programu
- Struktura metody (funkce, procedury)
- Rozklad na dílčí problémy
- Faktoriál pomocí funkcí
- Deklarace funkce
- Vstupní parametry funkce
- Výstupní parametr funkce (návratová hodnota)
- Formální a skutečné parametry funkce
- Přetěžování jmen funkcí
- Příklady funkcí
- Funkce pro výpočet NSD
- Euklidův algoritmus pro výpočet NSD
- Procedury
- Statické proměnné
- Zastínění nelokální proměnné
- Přidělování paměti proměnným

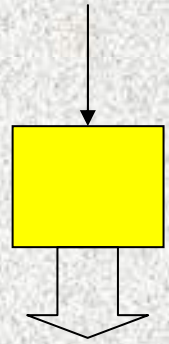
Funkce a procedury

- Příklad
  - výpočet sin, cos, výpočet faktoriálu, …
  - vstupy, výstupy - Systém.out.print (“vysledek”); Math.random();

- Funkce
  - definovány vstupní hodnoty a hodnota funkce



- Procedura
  - definovány vstupní hodnoty a činnost procedury



Funkce, rozklad na dílčí problémy

- Připomeňme si program pro výpočet faktoriálu:

```
public class Faktorial {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("zadejte přirozené číslo");
    int n = sc.nextInt();
    if (n < 1) {
      System.out.println(n + " není přirozené číslo");
      System.exit(0);
    }
    int i = 1;
    int f = 1;
    while (i < n) {
      i = i + 1;
      f = f * i;
    }
    System.out.println (n + "! = " + f);
  }// main() END
}// class Faktorial END
```

Čtení přirozeného čísla

Algoritmus výpočtu faktoriálu

Tisk výsledku

Čtení přirozeného čísla, výpočet faktoriálu a tisk výsledku jsou tři dílčí podproblémy, jejichž řešení popíšeme samostatnými funkcemi

Faktoriál pomocí funkcí

- Funkce pro čtení přirozeného čísla

```
static int ctiPrirozene() {
   Scanner sc = new Scanner(System.in);
   System.out.println ("Zadejte přirozené číslo");
   int n = sc.nextInt();
   if ( n < 1) {
   System.out.println (n + " není přirozené číslo");
   System.exit(0);
    }
   return n;
 }
```

- Hlavička funkce
  `static int ctiPrirozene()`
  vyjadřuje, že funkce nemá parametry a že výsledkem volání funkce je hodnota typu *int*
- Příkaz
  `return n;`
  předepisuje návrat z funkce, výsledkem volání je hodnota *n*
- Příklad volání funkce:
  `int nn = ctiPrirozene();`

Faktoriál pomocí funkcí

- Funkce pro výpočet faktoriálu

```
static int faktorial(int n) {
   int i = 1;
   int f = 1;
   while (i < n) {
     i = i + 1;
     f = f * i;
   }
   return f;
 }
```

- Hlavička funkce vyjadřuje, že funkce má jeden parametr typu *int* a že výsledkem je hodnota typu *int*

- Příklad volání funkce

```
int ff = faktorial(4);
```

Faktoriál pomocí funkcí

- Funkce pro tisk výsledku výpočtu

```
static void tiskVysledku(int n, String napis, int x) {
    Systém.out.println(n + napis + s);
}
```

- Hlavička vyjadřuje, že procedura má tři vstupní parametry (int, String, int). Procedura nevrací žádnou hodnotu (void).

- Příklad volání procedury

```
tiskVysledku(n, "! = ",f);
```

Faktoriál pomocí funkcí

- Výsledné řešení:

```
package pri04;

import java.util.*;
public class Faktorial {

   public static void main(String[] args) {
      int n = ctiPrirozene();
      int f = faktorial(n);
      tiskVysledku(n,"! = ",f);
   }

   static int ctiPrirozene() { … }
   static int faktorial(int n) { … }
   static void tiskVysledku(int n, String napis,int x){ … }
}
```

Deklarace funkce

- Deklaraci funkce tvoří

  hlavička funkce

  tělo funkce

- Hlavička funkce v jazyku Java má tvar

  ```
  static typ jméno(specifikace parametrů)
  ```

  kde

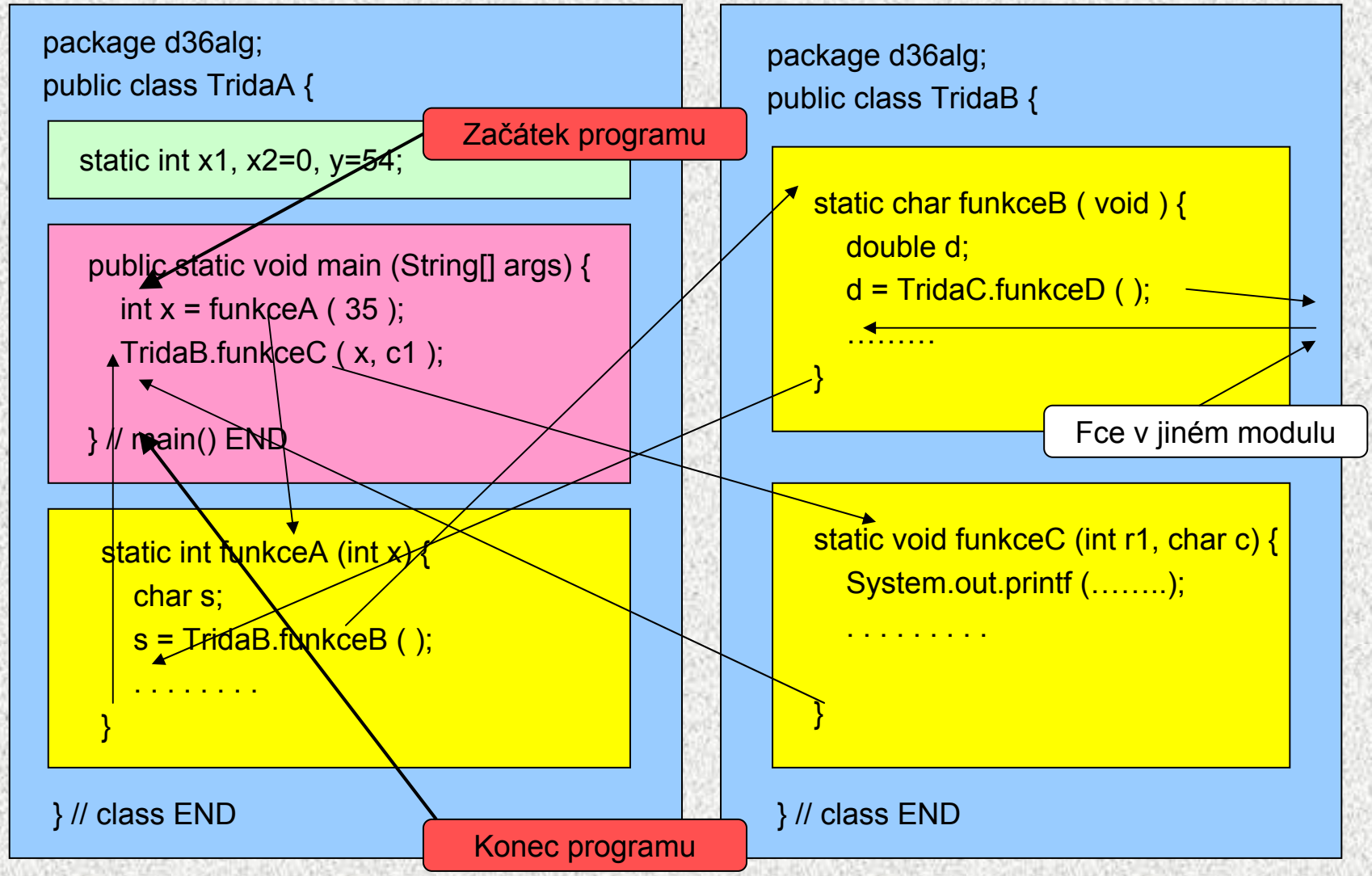
  - *typ* je typ výsledku funkce (funkční hodnoty)
  - *jméno* je identifikátor funkce
  - *specifikací parametrů* se deklarují parametry funkce, každá deklarace má tvar *typ_parametru jméno_parametru* (a oddělují se čárkou)
  - specifikace parametrů je prázdná, jde-li o funkci bez parametrů

- Tělo funkce je složený příkaz nebo blok, který se provede při volání funkce
- Tělo funkce musí dynamicky končit příkazem

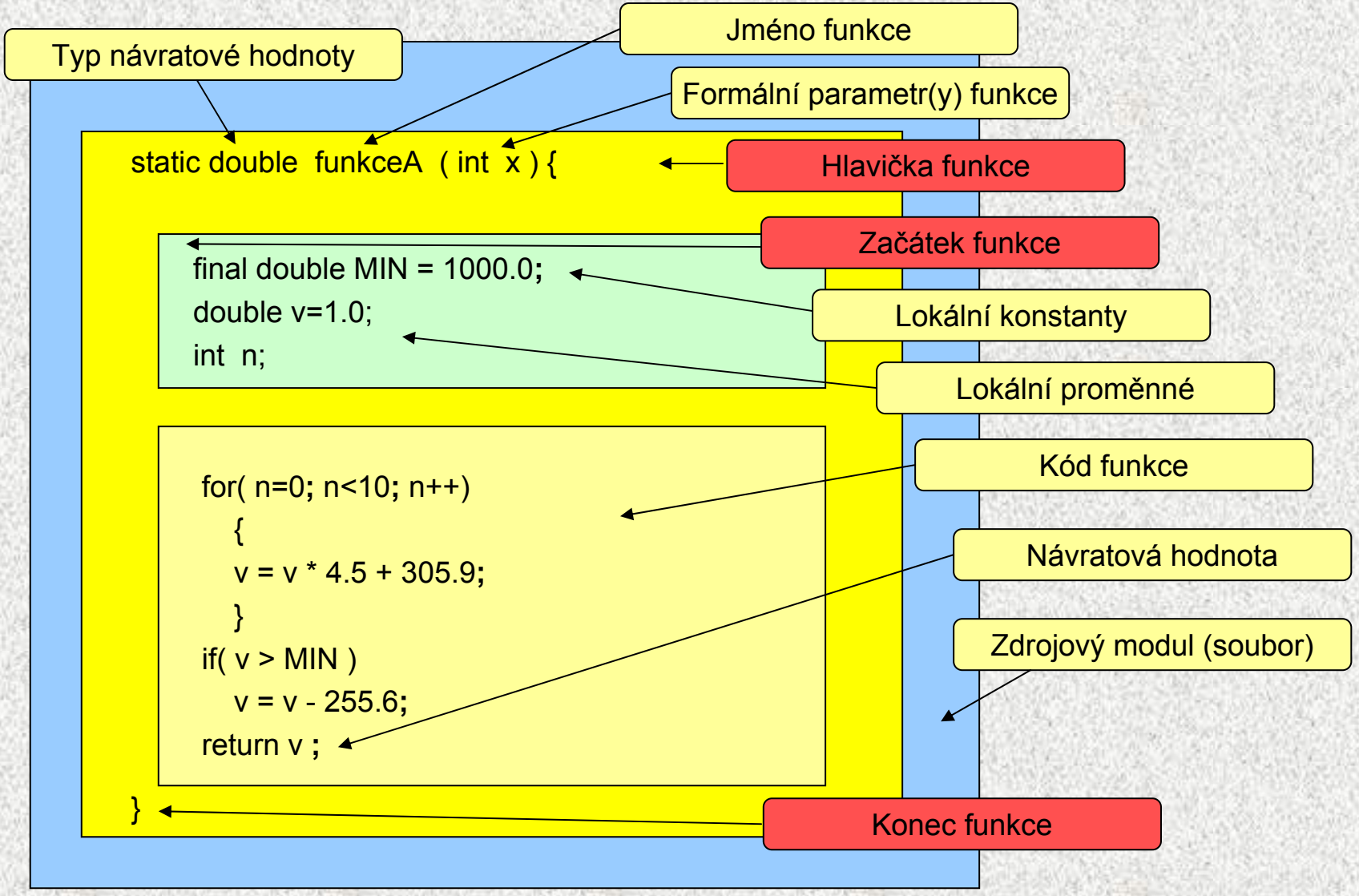
  ```
  return x;
  ```

  kde *x* je výraz, jehož hodnota je výsledkem volání funkce

Bloková struktura programu



Struktura metody (funkce, procedury)



Vstupní parametry funkce

- Parametry funkce jsou lokální proměnné funkce, kterým se při volání funkce přiřadí hodnoty skutečných parametrů
- Jestliže parametr funkce je typu *T*, pak přípustným skutečným parametrem je výraz, jehož hodnotu lze přiřadit proměnné typu *T* (stejná podmínka, jako u přiřazení)
- Příklad:

```
public class Max1 {
  static int max(int x, int y) {
    if (x>y) return x;
    else return y;
    }

  public static void main(String[] args) {
    int a = 10, b = 20;
    System.out.println(max(a+20, b));// O.K.
    System.out.println(max(1.1, b)); // Chyba při překladu
  }
}
```

Vstupní parametry funkce

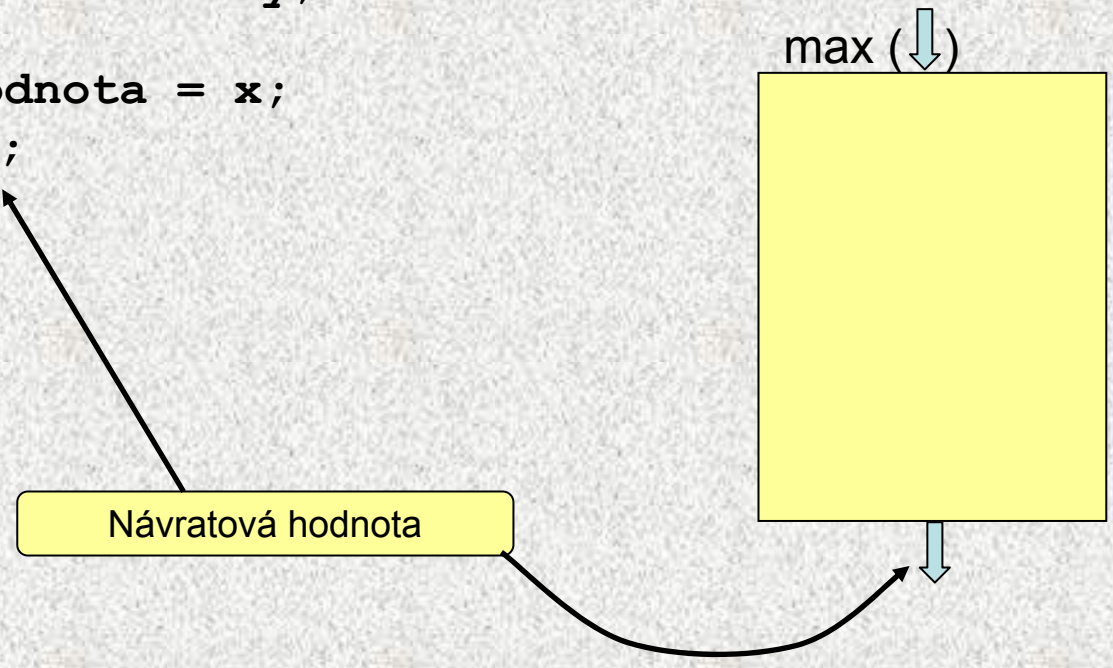
- Parametry funkce slouží pro předání vstupních dat algoritmu, který je funkcí realizován
- Častá chyba začátečníka: funkce, která čte hodnoty parametrů pomocí operace vstupu dat

```
static int max(int x, int y) {
  x = sc.nextInt();     // nesmyslný příkaz
  y = sc.nextInt();     // nesmyslný příkaz
  if (x>y)
    return x;
  else
    return y;
}

int z= max(7,99);
```

Výstupní parametr funkce (návratová hodnota)

- Typ návratové hodnoty se uvádí před názvem funkce
- Navratová hodnota (proměnná, výraz) se uvádí za příkazem return
- Funkce může vracet pouze jednu návratovou hodnotu
- Návratovou hodnotou může být i referenční proměnná (viz „pole“)

```
static int max(int x, int y) {
   int maxHodnota = y;
   if (x>y)
      maxHodnota = x;
   return x;
   }
```



Formální a skutečné parametry funkce

```
public static void main(String[] args){
    int a = 4;
    int b = 2;
    int m = max(a,b);
    System.out.println("max="+ m);
    m = max(b,6);
    System.out.println("max="+ m);
}

static int max(int x, int y) {
    int maxHodnota = y;
    if (x>y)
       maxHodnota = x;
    return x;
    }
```

Skutečné parametry

Zde volání hodnotou (Call by value)

Formální parametry

Formální parametry jsou zároveň lokální proměnné

Příklad – Výplata hotovosti

Výplata hotovosti (výčetka platidel) se odevzdává bance, uvádí vždy hodnotu bankovky a množství

My budeme hledat takovou výčetku, která pro danou částku obsahuje celkově nejmenší počet bankovek

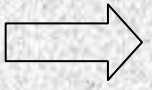
Příklad: (pro jednoduchost uvažujme pouze bankovky 5000, 1000, 500 a 100 Kč.)

38 900

bankovka (Kč)	počet (ks)
5000	7
1000	3
500	1
100	4

Příklad - Výplata hotovosti

```
public class Vycetka {
  public static void main(String[] args) {
    int plat, platX;
    int p5000,p1000,p500,p100;
    platX=sc.nextInt(); plat=platX;
    p5000=plat/5000;
    plat=plat%5000;
    p1000=plat/1000;
    plat=plat%1000;
    p500=plat/500;
    plat=plat%500;
    p100=plat/100;
    plat=plat%100;
```



```
pocet = plat/hodnBankovky;
plat  = plat%hodnBankovky;
```

```
    System.out.println("Vyplata castky " + platX + " je:\n"
     + p5000 +   ",\n" + p1000 +   ",\n„
     + p500  +   ",\n"  + p100  +   ", \na zbytek " + plat);
  }}
```

Výplata hotovosti s funkcí

```
public class VycetkaFunkce {
static  int plat;
  static int pocetBankovek(int hodnotaBankovky){
    int pocet;
    pocet = plat/hodnotaBankovky;
    plat  = plat%hodnotaBankovky;
    return pocet;
  };
  public static void main(String[] args) {
  int p5000,p1000,p500,p100, platX;
  plat=sc.nextInt();
  platX=plat;
  p5000=pocetBankovek(5000);
  p1000=pocetBankovek(1000);
  p500=pocetBankovek(500);
  p100=pocetBankovek(100);
 System.out.println ("Vyplata castky " + platX + " je:\n"
     + p5000 + ",\n" + p1000 +   ",\n„
     + p500  +   ",\n" + p100  +   ", \na zbytek " + plat);
 }}
```

Přetěžování jmen funkcí

- Funkce lišící se v počtu nebo typu parametrů se mohou jmenovat stejně (přetěžování jmen, overloading of names)
- Příklad:

```
public class Max2 {
  static int max(int x, int y) {
    if (x>y) return x; else return y;
  }

  static int max(int x, int y, int z) {
    return max(x, max(y, z));
  }

  static double max(double x, double y) {
    if (x>y) return x; else return y;
  }
  public static void main(String[] args) {
    System.out.println (max(3,4));
    System.out.println (max(1,2,3));
    System.out.println (max(1.0,2.4));
  }
}
```

Příklady funkcí

- Funkce pro zjištění, zda daný rok je přestupný

```
public class Rok {
  static boolean prestupny(int rok) {
    if (rok%4==0 && (rok%100!=0 || rok%1000==0))
      return true;
    else
      return false;
  }
  public static void main(String[] args) {
    int rok;
    System.out.println ("zadejte rok");
    rok = sc.nextInt();
    System.out.print("rok "+rok);
    if (prestupny(rok))
      System.out.println (" je přestupný");
    else
      System.out.println (" není přestupný");
  }
}
```

Funkce pro výpočet NSD 2

- Jednoduchý algoritmus výpočtu největšího společného dělitele jsme již uvedli
- Efektivnější algoritmus lze sestavit na základě těchto vztahů:

  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 $nsd(x,y) = nsd(x,y-x)$

- Řešení 2:

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

```
int nsd(int x, int y)
{
  int d;
  if (x<y) d=x; else d=y;
  while (x%d!=0 | y%d!=0)
     d--;
  return d;
}
```

```
public static void main(String[] args) {
int a=sc.nextInt();int b=sc.nextInt();

System.out.println("Nejvetsi spolecny delitel" +
                   a + ", " + b + "je " + nsd(a,b));
}
```

Funkce pro výpočet NSD 3

- Do těla cyklu vnoříme místo podmíněného příkazu pro jediné zmenšení hodnoty *x* nebo *y* dva cykly pro opakované zmenšení hodnot *x* a *y*
- Řešení 3:

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

public static void main(String[] args) {
int a=sc.nextInt();int b=sc.nextInt();

System.out.println("Nejvetsi spolecny delitel" +
                   a + ", " + b "je " + nsd(a,b));
}
```

Euklidův algoritmus pro výpočet NSD 4

- Vnitřní cykly řešení 3 počítají nenulový zbytek po dělení většího čísla menším
- Pro výpočet zbytku po dělení máme operaci %
- Jejím využitím dostaneme Euklidův algoritmus, který lze slovně formulovat takto: určíme zbytek po dělení daných čísel, zbytkem dělíme dělitele a určíme nový zbytek, až dosáhneme nulového zbytku; poslední nenulový zbytek je nsd
- Řešení 4:

```
  static int nsd(int x, int y) {
     int zbytek = x%y;
     while (zbytek != 0) {
       x = y; y = zbytek; zbytek = x%y;
     }
     return y;
   }
public static void main(String[] args) {
int a=sc.nextInt();int b=sc.nextInt();

System.out.println("Nejvetsi spolecny delitel" +
                   a + ", " + b "je " + nsd(a,b));
}
```

Procedury

- Funkce, jejíž typ výsledku je *void*, nevrací žádnou hodnotu
- Funkce, která nevrací žádnou hodnotu, se obecně nazývá procedura
- Příklady procedury:
  - hlavní funkce *main*
  - výstupní operace *p* a *pln* poskytované třídou *Sys*

- Příklad uživatelské procedury: výpis znaku *z* doplněného zleva mezerami na celkový počet *n* znaků

```
static void vypisZnak(char z, int n) {
  for (int i=1; i<n; i++) System.out.print(' ');
  System.out.print(z);
}
```

Statické proměnné

- Třída může obsahovat deklarace statických proměnných
- Statické proměnné třídy jsou použitelné ve všech funkcích dané třídy – jsou to pro ně nelokální proměnné
- Příklad:

```
public class StatickePromenne {
  static int x, y;      // statické proměnné třídy

  public static void main(String[] args) {
    System.out.println("zadejte dvě celá čísla");
    x = sc.nextInt(); y = sc.nextInt();
    // System.out.println(i); // Nelze, i zde neznama
    vypisSoucet();
  }

  static void vypisSoucet() {
    int i=6;
    System.out.println("součet čísel je "+(x+y+i));
  }
}
```

- Poznámka: statické proměnné jsou inicializovány hodnotou nula

Zastínění nelokální proměnné

- Deklarace lokální proměnné *p* zastíní deklaraci nelokální proměnné *p*
- Příklad:

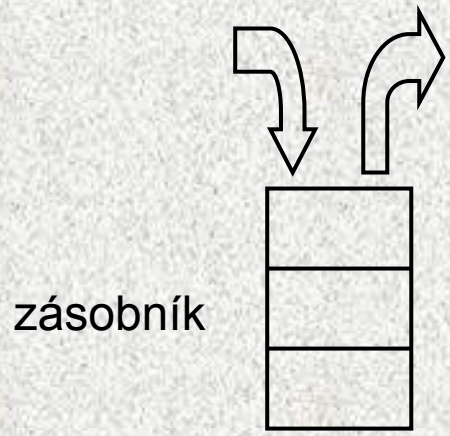
```
public class Zastineni {
  static int a = 10;

  public static void main(String[] args) {
    f();
    System.out.println(a);
  }

  static void f() {
    int a = 20;
    System.out.println(a);
  }
}
```

Program vypíše

```
20
10
```

Přidělování paměti proměnným

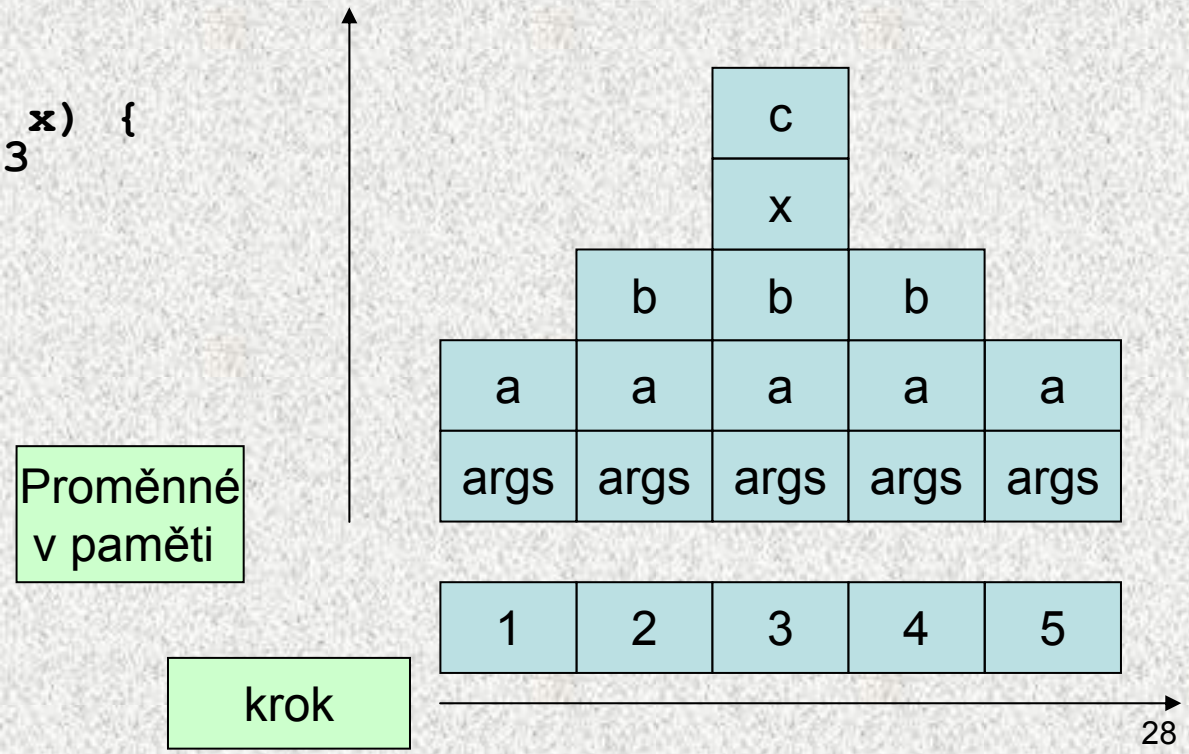
- Poznali jsme doposud dva druhy proměnných:
  - statické proměnné třídy
  - lokální proměnné funkcí
- Přidělením paměti proměnné rozumíme určení adresy umístění proměnné v paměti počítače
- Statickým proměnným třídy se přidělí paměť v okamžiku, kdy se do paměti zavádí kód funkcí třídy, a zůstane jim přidělena až do ukončení běhu programu
- Lokálním proměnným a parametrům funkce se paměť přidělí při každém volání funkce a zůstane jim přidělena jen do návratu z funkce (při návratu z funkce se přidělené adresy uvolní pro další použití)
- Úseky paměti přidělované lokálním proměnným a parametrům tvoří tzv. zásobník (stack, LIFO): úseky se přidávají a odebírají, přičemž se vždy odebere naposledy přidaný úsek



Přidělování paměti proměnným

- Příklad

```
public static void main(String[] args) {
   int a; ... //1
   f();       //5
     ...
   }
static void f() {
     int b; ... //2
     g(10);     //4
     ...
}
static void g(int x) {
     int c; ... //3
     ...
   }
```



Metody (procedury, funkce)

Jazyk JAVA

Konec

České vysoké učení technické Fakulta elektrotechnická