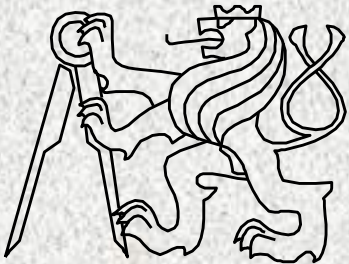


Metody (funkce a procedury)



BD6B36PJV 003

Fakulta elektrotechnická

České vysoké učení technické

Obsah

- Metody, funkce a procedury, důvody
- Deklarace a volání metod
- Předávání parametrů
- Hodnota funkce
- Procedura
- Struktura programu

Funkce a procedury

- Příklad

- výpočet sin, cos, výpočet faktoriálu, ...
- vstupy, výstupy - `System.out.print("vysledek"); Math.random();`

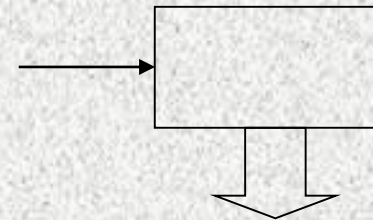
- Funkce

- definovány vstupní hodnoty a hodnota funkce



- Procedura

- definovány vstupní hodnoty a činnost procedury



Funkce

- 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);
    }
}
```

čtení
přirozeného
čísla

algoritmus výpočtu faktoriálu

zobrazení

Čtení přirozeného čísla, výpočet faktoriálu a zobrazení 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

Typická procedura

```
static void vystup(int n, int vysl) {  
    System.out.println(n + "! = " + vysl);  
}
```

Faktoriál pomocí funkcí

- Výsledné řešení:

```
public class Faktorial {
    static int ctiPrirozene() {
        ...
    }
    static int faktorial(int n) {
        ...
    }
    static void vystup(int n, int vysl) {
        ...
    }
    public static void main(String[] args) {
        int nuu = ctiPrirozene();
        int vysl = faktorial(nuu);
        vystup(nuu, vysl);
    }
}
```

- Proměnnou `nuu` ve funkci `main` lze vynechat:

```
S. .println("Fakt"+n+"!="+faktorial(ctiPrirozene()));
```


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

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.)

Částka: 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"
```

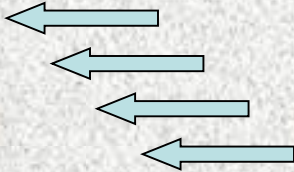
```
+ p500 + ",\n"
```

```
+ p100 + " \n, \na zbytek " + plat);
```

```
}}
```

Výplata hotovosti s funkcí s nelokální proměnnou

```
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 + " \n, \na zbytek " + plat);
    }
}
```



The diagram consists of four horizontal arrows pointing to the left, stacked vertically. Each arrow points to one of the four function calls in the main method: pocetBankovek(5000), pocetBankovek(1000), pocetBankovek(500), and pocetBankovek(100). The arrows are light blue with a white outline and are arranged in a descending staircase pattern from top-left to bottom-right.

Výplata hotovosti bez nelokální

```
static int pocetBankovek(int platf, int hodnotaBankovky) {  
    return platf / hodnotaBankovky;  
}
```

```
static int novyPlat(int platf, int hodnotaBankovky) {  
    return platf % hodnotaBankovky;  
}
```

```
platX = plat;  
p5000 = pocetBankovek(plat, 5000);  
plat = novyPlat(plat, 5000);  
p1000 = pocetBankovek(plat, 1000);  
plat = novyPlat(plat, 1000);  
p500 = pocetBankovek(plat, 500);  
plat = novyPlat(plat, 500);  
p100 = pocetBankovek(plat, 100);  
plat = novyPlat(plat, 100);
```

Výplata hotovosti objektově

```
class Vycetka {  
    static int plat;  
    int bankovka;  
    int pocetBankovek(int hodnotaBankovky) {  
        int pom;  
        pom = plat / hodnotaBankovky;  
        plat = plat % hodnotaBankovky;  
        return pom;  
    }  
    Vycetka(int plats) {plat = plats;  
    }  
}
```

```
Vycetka v = new Vycetka(plat);  
p5000 = v.pocetBankovek(5000);  
p1000 = v.pocetBankovek(1000);  
p500 = v.pocetBankovek(500);  
p100 = v.pocetBankovek(100);
```

Předávání parametrů

- **Formální a skutečné parametry** metody – slouží k **předávání dat** do metod
- **Formální parametry**
 - Jsou proměnné uvedené v kulaté závorce hlavičky metody.
 - Mají typ a jméno
- **Formální parametry** metody jsou **lokální proměnné** metody, kterým se při **volání** metody **přiřadí** hodnoty **skutečných parametrů**.
- **Skutečné parametry** metody se **dosazují na místo formálních** parametrů
- Způsob **přiřazení skutečných parametrů formálním** se v Javě řídí metodou "volání hodnotou" ("Call by Value") (viz dále) – „**fiktivní příkaz**“

Důležité

- **Přípustný** datový typ skutečného parametru vzhledem k datovému typu formálního parametru se řídí **stejnými pravidly jako v případě přiřazení** tj.:
 - shodné typy
 - automatická konverze typu
 - vynucená konverze typu
 - zakázaná konverze typu (např. boolean->int ... nelze přiřadit



Formální a skutečné parametry metody

```
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);  
}
```

Skutečné parametry

Skutečné parametry obsahují aktuální předávaná data do metody

Formální parametry

Formální parametry umožní v těle metody popsat, jak má algoritmus zpracovat vstupní data

Formální parametry jsou lokální proměnné

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



Vlastnosti proměnných – způsob předávání parametrů

- Pro předávání parametrů (skutečné → formální) dvě metody:
 - "Volání hodnotou" ("Call by Value")
 - "Volání odkazem" ("Call by Reference")
- Volání hodnotou:
 - V místě předání skutečných parametrů do formálních se předá kopie hodnoty skutečného parametru do parametru formálního.
 - Důsledek – změnou formálního parametru v metodě **nelze změnit** hodnotu původního skutečného parametru (metoda zná jen kopii hodnoty nikoliv adresu skutečného parametru). Je to **bezpečnostní prvek** (metoda nemůže ovlivnit své okolí) a **zároveň omezení** (formální parametr volaný hodnotou **nelze použít** jako **výstupní bod** z metody).
- Volání odkazem
 - V místě předání skutečných parametrů do formálních se předá reference („adresa“) na skutečný parametr (více v lekci o polích)

Důležité



Důležité



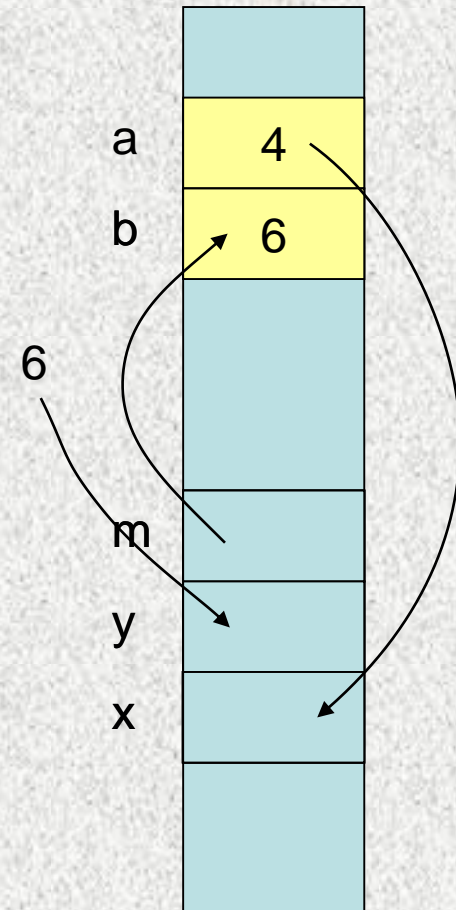
-
- **Java zná jen volání hodnotou!**
 - v dalších lekcích si ukážeme, že volání hodnotou pro referenční proměnné lze využít pro volání odkazem - týká se pouze polí a objektů!



Volání hodnotou "Call by Value"

```
public static void main(String[]args) {  
    int a = 4,b;  
    b = max(a,6);  
    ...           //dalsi prikazy  
}  
  
static int max(int x,int y) {  
    int m = y;  
    if (x > y)  
        m = x;  
    return m;  
}
```

Paměť dat



Parametry funkce, chyba

- 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) {  
    Scanner sc = new Scanner(System.in);
```

```
    x = sc.nextInt();
```

```
    y = sc.nextInt();
```

```
    if (x>y) return x;
```

```
    else return y;
```

```
}
```

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



Parametry funkce, chyba

- 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) {  
    Scanner sc = new Scanner(System.in);
```

```
    x = sc.nextInt();
```

```
    y = sc.nextInt();
```

```
    if (x>y) return x;
```

```
    else return y;
```

```
}
```

```
-----
```

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



Návrh metody - zásady

- Metodu navrhovat pro opakované použití s různými daty (universálnější)
- Ověřit platnost vstupních dat (rozsah)
- Omezená délka kódu metody (v Javě cca 40-60 řádků)
- Omezený počet parametrů (5-8)
- Volba mnemotechnického jména metody (např. korenyKvadratickeRovnice)
- Volba mnemotechnických jmen proměnných (např: polomerKruhu)
- Volba mnemotechnických jmen konstant (např. MAX_POLOMER)
- Data do metod předávat pomocí formálních parametrů
- Data z metod předávat funkční hodnotou nebo předáním "reference" (odkazu) (u procedur) – jen u polí a objektů
- Uvnitř metod nepoužívat statické (globální) proměnné
- Místo hluboce vnořených `if` používat `switch`
- Na začátku metody – komentář - co metoda počítá, jaká vyžaduje vstupní data, jaká data jsou na výstupu.
- Neobvyklé (i obvyklé) programátorské obraty komentovat
- Dodržovat formátování zdrojového kódu



Statické proměnné

Třída může obsahovat deklarace statických proměnných

Statické proměnné třídy jsou použitelné ve všech funkcích dané třídy – jsou to pro ně nelokální proměnné

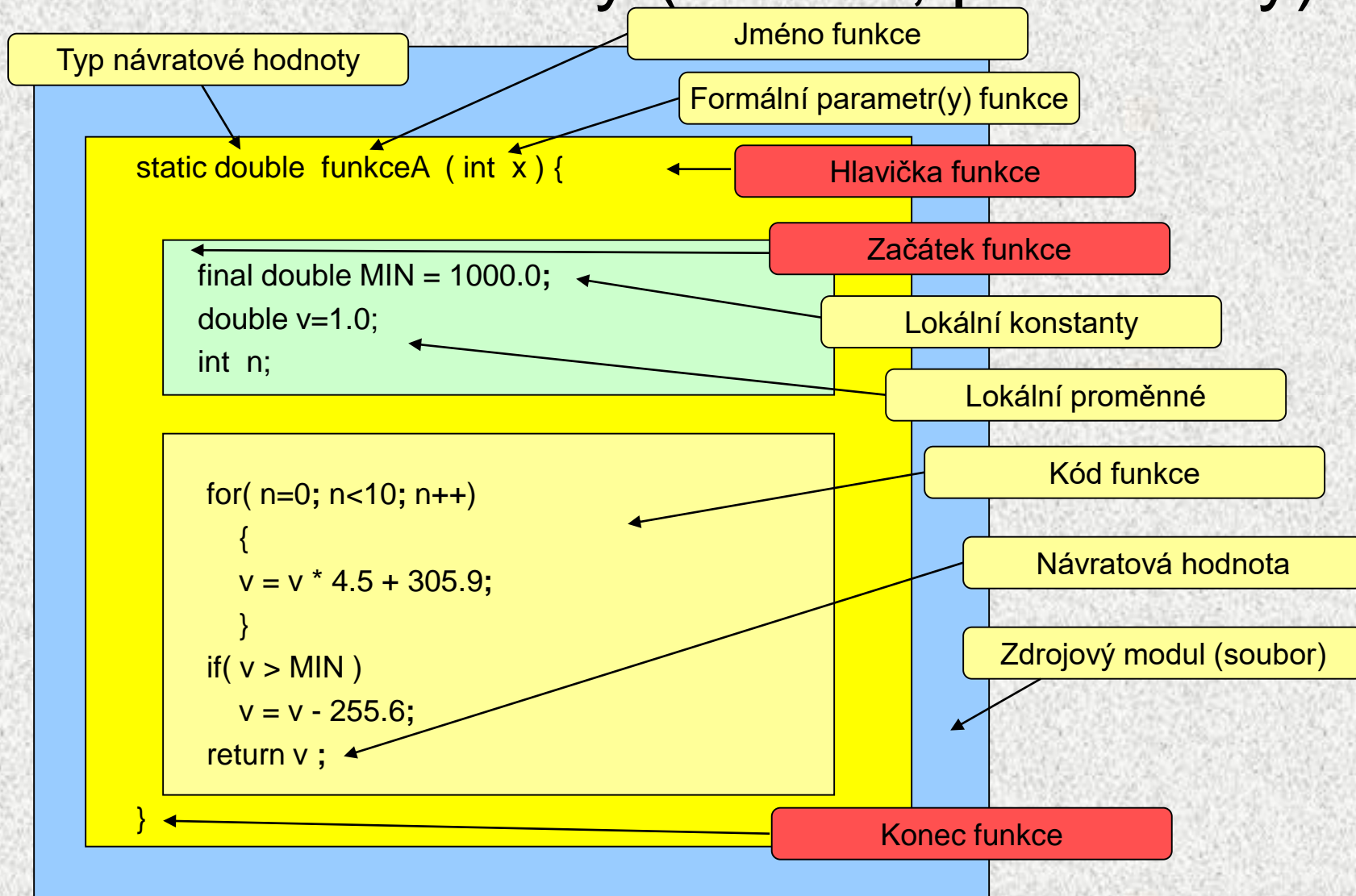
```
public class StatickePromenne {  
    static int x=0, y; // statické proměnné třídy  
  
    public static void main(String[] args) {  
        System.out.println("zadejte dvě celá čísla");  
        Scanner sc = new Scanner(System.in);  
        x = sc.nextInt(); y = sc.nextInt();  
        // System.out.println(i);  
        vypisSoucet();  
    }  
    static void vypisSoucet() {  
        int i=6;  
        System.out.println("součet čísel je "+(x+y+i));  
    }  
}
```



Platí x,y

Poznámka: statické proměnné jsou inicializovány nulovou hodnotou (0, 0.0,null, false)

Struktura metody (funkce, procedury)



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));
    }
}
```


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 `print` a `println`
- 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);  
}
```

Hvězdičky - procedury

```
static void hvezdickyln(int pocet) {
    for (int i=1; i<=pocet; i++) System.out.print('*');
    System.out.println("");
}

static void hvezdicky(int pocet) {
    for (int i=1; i<=pocet; i++) System.out.print('*');
}

static void mezeryln(int pocet) {
    for (int i=1; i<=pocet; i++) System.out.print(' ');
    System.out.println("");
}

static void mezery(int pocet) {
    for (int i=1; i<=pocet; i++) System.out.print(' ');
}
```

Hvězdičky - procedury

```
public class Hvezdicky {  
    public static void main(String[] args ) {  
        hvezdickyln(5);  
        mezery(4);  
        hvezdicky(10);  
        mezeryln(0);  
        hvezdickyln(5);  
    }  
}
```

```
*****  
      *****  
*****
```

Zastínění nelokální proměnné

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

```
public class Zastineni {  
    static int a = 10;  
    public static void main(String[] args) {  
        p();  
        System.out.println(a);  
    }  
    static void p() {  
        int a = 20;  
        System.out.println(a);  
    }  
}
```

Zastineni.a

Program vypíše

20

10

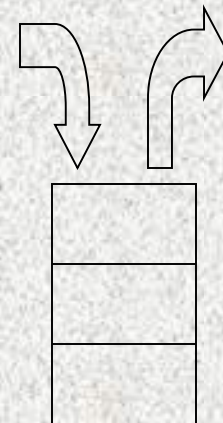
Přidělování paměti proměnným

- Poznali jsme doposud dva druhy proměnných:
 - statické proměnné třídy
 - lokální proměnné funkce
- Přidělením paměti proměnné rozumíme určení adresy umístění proměnné v paměti počítače
- Statickým proměnným třídy se přidělí paměť v okamžiku, kdy se do paměti (do JVM) zavádí kód funkcí třídy, a zůstane jim přidělena až do ukončení běhu programu
- Lokálním proměnným a parametrům funkce se paměť přidělí při volání funkce a zůstane jim přidělena jen do návratu z funkce (při návratu z funkce se přidělené adresy uvolní pro další použití)

Přidělování paměti proměnným

- Úseky paměti přidělované lokálním proměnným a parametrům tvoří tzv. **zásobník** (stack):
úseky se přidávají a odebírají, přičemž se vždy odebere naposledy přidáný úsek

zásobník

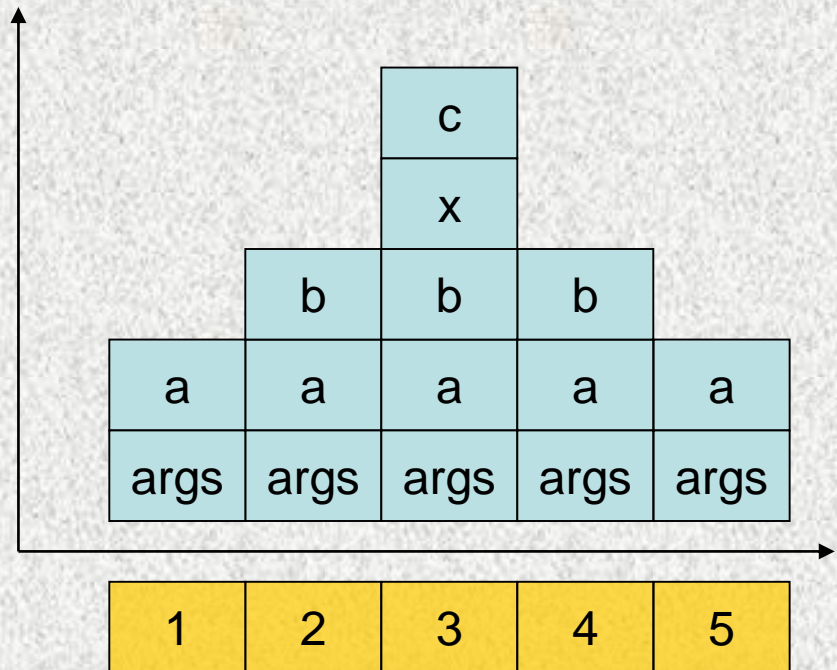


Přidělování paměti proměnným

- 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  
    ...  
}
```

Proměnné
v paměti



krok

Příklady

Problém: prvočísla

- Zjistěte počet **prvočísel** menších než zadané číslo n a určete čas, který na to budete potřebovat
- **prvočísl**o: **přirozené číslo** větší než 1, které je beze zbytku dělitelné **pouze** jedničkou a samo sebou.
- **přirozené číslo**: **kladné celé číslo** větší než 0
- Algoritmus:
 - Pro každé přirozené číslo menší než n rozhodněte, zda jde o prvočísl
 - Nasčítejte prvočísla
- **podproblémy** :
 1. zjištění zda dané číslo je prvočísl
 2. měření času

Příklad: prvočísla I

Dle definice: Přirozené číslo n je **prvočíslo**, právě tehdy když jej beze zbytku dělí pouze číslo n a číslo 1

```
public static boolean isPrvocislo1(long n) {
    if (n < 2 ) return false;
    // ostatní celá čísla nejsou prvočísla
    int pocetDelitelu = 0;
    for (int i = 1; i <= n; i++) {
        if (n%i==0) pocetDelitelu++;
    }
    if (pocetDelitelu==2) return true;
        else return false;
}
```

Příklad: prvočísla II

- Vylepšení 1. - najdeme-li prvního dělitele jiného než čísla 1 a n , již to není prvočíslo

```
public static boolean isPrvocislo2(long n) {  
    if (n < 2 ) return false;  
    for (int i = 2; i < n; i++) {  
        // vynecháme číslo 1 a číslo n  
        if (n%i==0) return false;  
    }  
    return true;  
}
```

Příklad: prvočísla III

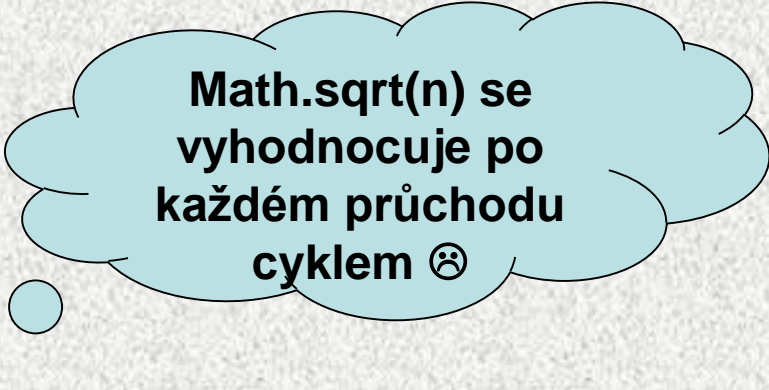
- Vylepšení 2. Číslo složené lze napsat jako součin dělitelů, např. $35=7*5$; jedno je „menší“ a druhé „větší“

```
public static boolean isPrvocislo3(long n) {  
    if (n < 2 ) return false;  
    for (int i = 2; i <= (n/2) ; i++) {  
        // zkoušíme jen do půlky  
        if (n%i==0) return false;  
    }  
    return true;  
}
```

Příklad: prvočísla IV

- Vylepšení 2. Číslo složené lze napsat jako součin dělitelů, např. $35=7*5$; jedno je „menší“ a druhé „větší“, pokud nejsou stejné jako $49 = 7*7$

```
public static boolean isPrvocislo(long n) {  
    if (n < 2 ) return false;  
    for (long i = 2; i <= Math.sqrt(n) ; i++) {  
        if (n%i==0) return false;  
    }  
    return true;  
}
```



Math.sqrt(n) se
vyhodnocuje po
každém průchodu
cyklem ☹

Příklad: prvočísla V

- Vylepšení 2. Číslo složené lze napsat jako součin dělitelů, např. $35=7*5$; jedno je „menší“ a druhé „větší“, pokud nejsou stejné jako $49 = 7*7$

```
public static boolean isPrvocislo(long nn) {  
    if (nn < 2 ) return false;  
    long n=(long)Math.sqrt(nn);  
    for (long i = 2; i <= n ; i++) {  
        if (n%i==0) return false;  
    }  
    return true;  
}
```

Math.sqrt(n) se
nevyhodnocuje po
každém průchodu
cyklem 😊

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 je celé nezáporné číslo
 - stroj musí z počtu p zápalek odebrat x zápalek tak, aby platilo $(\text{počet} - x) = 4n + 1$
 $(\text{počet} - 1) - 4n = x$, ~ hledáme zbytek po dělení 4
 - z tohoto vztahu po úpravě
 $x = (\text{počet} - 1) \bmod 4 = (\text{počet} - 1) \% 4$
 - vyjde-li $x=0$, znamená to, že okamžitý počet zápalek je pro stroj nevýhodný a bude-li protihráč postupovat správně, stroj prohraje.

Hra NIM

- Hrubé řešení:

```
int pocet;  
boolean stroj = true;  
"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"
```

abstraktní příkazy



- Podproblémy „zadání počtu zápalek“, „odebrání zápalek strojem“ a „odebrání zápalek hráčem“ (abstraktní příkazy) budeme realizovat procedurami, proměnné *pocet* a *stroj* pro ně budou 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("vyhrál jsem");
        else
            System.out.println("vyhrál jste, gratuluji");
    }

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



Hra NIM

```
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

```
static void zadaniPoctu() {  
do {  
System.out.print("zadejte počet zápalek(od 15 do 35)");  
pocet = sc.nextInt();  
    } while (pocet<15 || pocet>35);  
}
```



Hra NIM

```
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;  
}
```

Další 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%400==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, v 2

- Jednoduchý algoritmus výpočtu největšího společného dělitele jsme již uvedli

- Efektivnější

je-li $x = y$

je-li $x > y$

je-li $x < y$

- Řešení 2:

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

```
public static void main(String[] args)
{
    int a=5, b=10;
    System.out.println("Nejvetsi spolecny delitel" +
```

```
        a + ", " + b + " je " + nsd(a,b));
}
```

```
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;
}
```

Funkce pro výpočet NSD, v 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

Vnitřní cykly řešení 3 počítají nenulový zbytek po dělení většího čísla menším

Pro výpočet zbytku po dělení máme operaci %

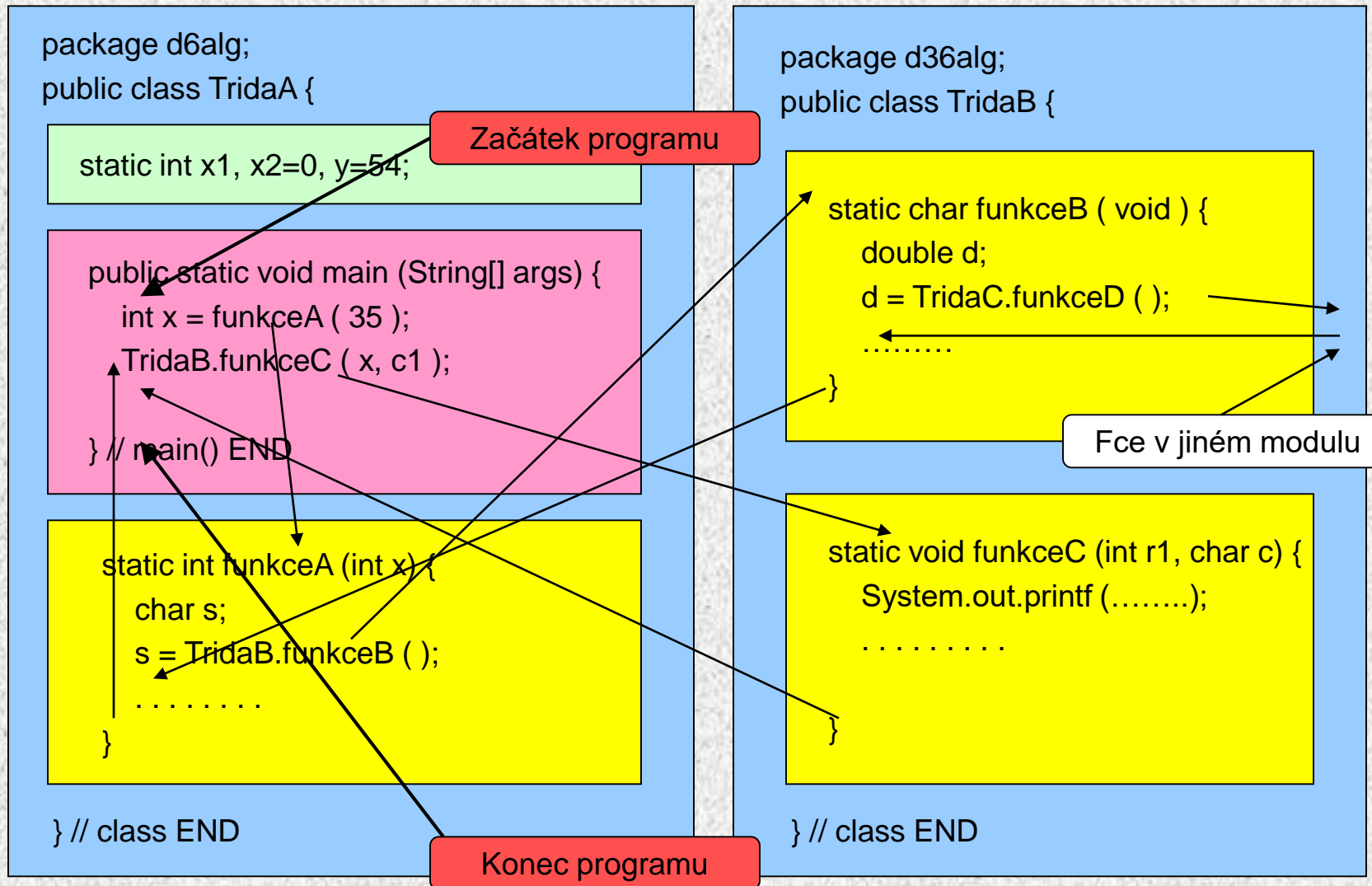
Řešení 4:

```
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(Scanner sc) {
    int a=sc.nextInt();int b=sc.nextInt();
    System.out.println("Největší NSD čísel " + a + " a " + b + " je " + nsd(a,b));
}
```

Využitím operátoru % dostaneme Euklidův algoritmus: určíme zbytek po dělení daných čísel, zbytkem dělíme dělitele a určíme nový zbytek, až dosáhneme nulového zbytku; poslední nenulový zbytek je nsd

Bloková struktura programu



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  
}  
}
```

- Řešení – přetěžování funkcí