Funkce

- Připomeňme si program pro výpočet faktoriálu:

```
public class Faktorial {
  public static void main(String[] args) {
    Sys.pln("zadejte přirozené číslo");
    int n = Sys.readInt();
    if (n<1) {
      Sys.pln(n + " není přirozené číslo");
      System.exit(0);
    }
    int i = 1;
    int f = 1;
    while (i<n) {
      i = i+1;
      f = f * i;
    }
    Sys.pln(n + "! = " + f);
  }
}
```

čtení přirozeného čísla

algoritmus výpočtu faktoriálu

- Čtení přirozeného čísla a výpočet faktoriálu jsou dva dílčí podproblémy, jejichž řešení popíšeme samostatnými funkcemi

Faktoriál pomocí funkcí

- Funkce pro čtení přirozeného čísla

```
static int ctiPrirozene() {
  Sys.pln("zadejte přirozené číslo");
  int n = Sys.readInt();
  if (n<1) {
    Sys.pln(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 n = 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 f = faktorial(4);
```

Faktoriál pomocí funkcí

- Výsledné řešení:

```
package alg4;
import sugar.Sys;

public class Faktorial {
  static int ctiPrirozene() {
    ...
  }
  static int faktorial(int n) {
    ...
  }
  public static void main(String[] args) {
    int n = ctiPrirozene();
    Sys.pln(n + "! = " + faktorial(n));
  }
}
```


- Proměnnou *n* ve funkci *main* lze vynechat:

```
public static void main(String[] args) {
  Sys.pln(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

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;
    Sys.pln(max(a+20, b));     // O.K.
    Sys.pln(max(1.1, b));// Chyba při překladu
  }
}
```

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 = Sys.readInt();      // nesmyslný příkaz
  y = Sys.readInt();      // nesmyslný příkaz
  if (x>y)
    return x;
  else
    return y;
}
```

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) {
    Sys.pln(max(3,4));
    Sys.pln(max(1,2,3));
    Sys.pln(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;
    Sys.pln("zadejte rok");
    rok = Sys.readInt();
    Sys.p("rok "+rok);
    if (prestupny(rok))
      Sys.pln(" je přestupný");
    else
      Sys.pln(" není přestupný");
  }
}
```

Funkce pro výpočet NSD

- 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;
}
```

Funkce pro výpočet NSD

- 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;
}
```

Euklidův algoritmus pro výpočet NSD

- 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;
 }
```

Procedury

- Funkce, jejíž typ výsledku je *void*, nevrací žádnou hodnotu
- Funkce, která nevrací žádnou hodnotu, se obecně nazývá procedura
- Příkladem procedury je hlavní funkce *main*
- Dalšími příklady procedur jsou 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++) Sys.p(' ');
  Sys.p(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) {
    Sys.pln("zadejte dvě celá čísla");
    x = Sys.readInt(); y = Sys.readInt();
    vypisSoucet();
  }

  static void vypisSoucet() {
    Sys.pln("součet čísel je "+(x+y));
  }
}
```


- Poznámka: statické proměnné jsou inicializovány hodnotou nula

Zastínění nelokální proměnné

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

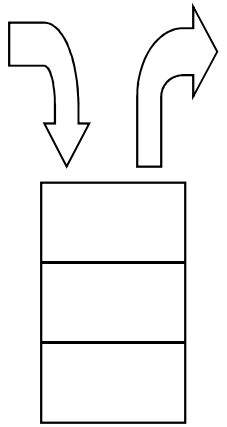
```
public class Zastineni {
  static int a = 10;
  public static void main(String[] args) {
    f();
    Sys.pln(a);
  }
  static void f() {
    int a = 20;
    Sys.pln(a);
  }
}
```

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 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): úseky se přidávají a odebírají, přičemž se vždy odebere naposledy přidaný 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
  ...
}
```