Základy Javy

Obsah

- Data v Javě
- Operátory a další elementy
- Základní příkazy
- Procedury a funkce
- Jednoduchá kolekce

Datové typy

- Abstrahujeme od binární podoby paměti počítače
  - data jako hodnoty různých datových typů uložené v datových objektech
- Datový typ (zkráceně jen typ) specifikuje:
  - množinu hodnot (hodnota je elementem datového typu)
  - množinu operací, které lze s hodnotami daného typu provádět
- Příklad typu: celočíselný typ *int* v jazyce Java:
  - množina hodnot – 32 bitů= celá čísla z intervalu: *cca +- 2 mld.*
  - množina operací
    - aritmetické operace +, -, *, /, jejichž výsledkem je hodnota typu *int*
    - relační operace ==, !=, >, >=, <, <=, jejichž výsledkem je hodnota typu *boolean*
    - a další
- Typ *int* je jednoduchý (primitivní) typ, jehož hodnoty jsou atomické - z hlediska operací dále nedělitelné – nelze pracovat s podtypy

Jednoduché datové typy

- V Javě jsou tyto jednoduché (primitivní) typy jazyka Java :
  - ***byte*** celá čísla z intervalu –128 .. 127
  - ***int*** celá čísla z intervalu -2147483648 .. 2147483647
  - ***double*** aproximace reálných čísel v absolutní hodnotě od $4.940654584124654 4 E^{-324}$ do $1.7976931348623157 E^{+308}$
  - ***boolean*** *false*, *true*
  - ***char*** znaky Unicode

- Další primitivní typy – celočíselné: ***short***, ***long***, reálné: ***float***)

- Reprezentace hodnot jednoduchých typů v paměti počítače:
  - ***byte*** 8 bitů v doplňkovém kódu
  - ***int*** 32 bitů v doplňkovém kódu
  - ***double*** 64 bitů v pohyblivé řádové čárce (mantisa, exponent)
  - ***boolean*** 8 bitů, 0 nebo 1 (zabírá celý bajt)
  - ***char*** 16 bitů (!) v kódu Unicode

Primitivní datové typy

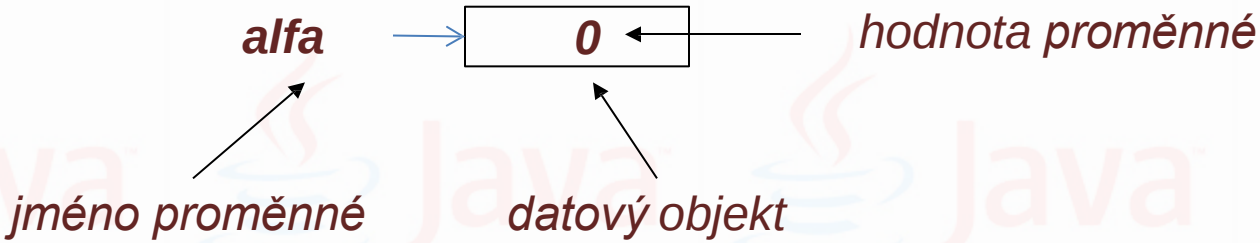
zdroj: http://interval.cz/clanky/naucte-se-javu-datove-typy/

typ	popis	velikost	min. hodnota	max. hodnota
byte	celé číslo	8 bitů	-128	+127
short	celé číslo	16 bitů	-32768	+32767
int	celé číslo	32 bitů	-2147483648	+2147483647
long	celé číslo	64 bitů	-9223372036854775808	+9223372036854775807
float	reálné číslo	32 bitů	-3.40282e+38	+3.40282e+38
double	reálné číslo	64 bitů	-1.79769e+308	+1.79769e+308
char	znak UNICODE	16 bitů	/u0000	/uFFFF
boolean	logická hodnota	1 bit	-	-

- kromě primitivních typů lze použít proměnné a konstanty pro uložení odkazů na objekty

Proměnné a přiřazení

- Proměnná je datový objekt, který je označen jménem a je v něm uložena hodnota nějakého typu, která se může měnit



- Deklarace (a inicializace) – oznámení překladači o úmyslu používat danou proměnnou a nastavení její první hodnoty

```
int alfa;
int alfa = 0;
```

- Hodnotu proměnné lze změnit přiřazovacím příkazem

```
alfa = 37;
```

Deklarace proměnných

- **Příklady** deklarací proměnných

```
int i;          // deklarace proměnné i typu int
double x;       // deklarace proměnné x typu double
```

- Proměnná deklarovaná uvnitř funkce (lokální proměnná) nemá deklarací definovanou hodnotu
- Použití proměnné s nedefinovanou hodnotou v jazyku Java je chyba při překladu!

- **Příklad**:

```
int x, y;       // lze i několik proměnných současně
x = y + 2;      // chyba při překladu (na rozdíl od chyby při běhu)
```

- Deklarace s inicializací proměnné
- `int x = 10;` // deklarovaná proměnná má hodnotu **10**

Literály a pojmenované konstanty

- Přímý zápis hodnoty v programu se nazývá literál
- Příklady literálů:
  - pro datový typ ***int*** *34 45*
  - pro datový typ ***double*** *25.3 1.2E-3* (= 0,0012)
  - pro datový typ ***boolean*** *false true*
  - pro datový typ ***char*** *'a' '1' '+'*
- Literál typu *String* (nejedná se o jednoduchý typ!): *"abcd " "Nazdar"*
- Lze používat i pojmenované konstanty
  - deklarují se jako inicializované proměnné, v deklaraci je navíc klíčové slovo ***final***
- Příklad deklarace konstant:

```
final int MAX = 100;
final String NAZEV_PREDMETU = "OBP1";
```

- Konvence: jména konstant píšeme velkými písmeny
- Konstantě nelze změnit hodnotu přiřazovacím příkazem

```
MAX = 20;                    // chyba při překladu
```

Výrazy

- Výraz předepisuje způsob výpočtu hodnoty určitého typu:
- Výraz může obsahovat:
  - proměnné
  - konstanty
  - volání funkcí
  - binární operátory
  - unární operátory
  - závorky
- Pořadí operací předepsaných výrazem je dáno:
  - prioritou operátorů
  - asociativitou operátorů

- Příklad:

výraz	*pořadí operací*	*zdůvodnění*
`x + y * z`	`x + (y * z)`	* má vyšší prioritu než +
`x + y + z`	`(x + y) + z`	+ je asociativní zleva

Přiřazovací příkaz

- Přiřazovací příkaz slouží pro přiřazení hodnoty proměnné

```
<proměnná> = <výraz>;
```

- **Příklad:**

```
x = y + z;      // proměnné x se přiřadí součet hodnot proměnných y a z
x = x + 1;      // hodnota proměnné x se zvětší o 1
```

- Proměnné v jazyku Java lze přiřadit pouze hodnotu odpovídajícího typu!

- Příklady nedovolených přiřazovacích příkazů:

```
boolean b; int i; double d;
b = 1;
i = 1.4;
d = true;
```

Přiřazovací příkaz

- Modifikace proměnné s využitím její předchozí hodnoty
  - 1. varianta:
    <proměnná> = <proměnná> <OP> <výraz>
  - 2. varianta:
    <proměnná> <OP>= <výraz>
  - je možné vynechat jméno proměnné na pravé straně, je-li shodné se jménem proměnné na levé straně

- Příklad ekvivalentního přiřazení: `j += 5;` `j = j + 5;`

- Přiřazení je možné použít jako výraz !!!

```
int x, y;
x = 7;
y = x = x + 6;
```

// má hodnotu *7* a lze tedy opět přiřadit!!

// vyhodnotí se jako *y = (x = (x + 6));*

Typové konverze

- Typová konverze - hodnota jednoho typu se převede na hodnotu jiného typu
- Typová konverze může být implicitní (vyvolá se automaticky) nebo explicitní (v programu je třeba ji explicitně předepsat – tzv. přetypování)
- např. konverze typu ***int*** na ***double*** je v jazyku Java implicitní, ale převod hodnoty typu ***double*** na ***int*** (odseknutím necelé části) je třeba explicitně předepsat - ztráta přesnosti !!!

- Příklad:

```
double x; int i = 1;    // hodnota 1 typu int se automaticky převede na
x = i;                  // hodnotu 1.0 typu double

double x = 1.2; int i;  // hodnota 1.2 typu double se odseknutím
i = (int) x;            // necelé části převede na hodnotu 1 typu int
```

Převod mezi typy int a boolean

Příklad: ukázka řešení "převodu" mezi *boolean* a *int* programově:

```
boolean b = false;
int i=1;

b = (i != 0);
// b je true, když i je různé od nuly, jinak false

i = b?1:0;
// tzv. ternární výraz: když b, tak i=1, jinak i=0
```

Operátory a další prvky jazyka

Aritmetické operátory

- Pro operandy typu ***int*** a ***double*** (seřazeno sestupně podle priority):
  - unární `-` (změna znaménka)
  - binární `*`, `/` a `%` (násobení, dělení a zbytek po dělení)
  - binární `+` a `-` (sčítání a odčítání)

- Jsou-li oba operandy stejného typu, výsledek operace téhož typu
- Jsou-li operandy různého typu, operand typu ***int*** se implicitní konverzí převede na hodnotu typu ***double*** a výsledkem je typu ***double***

- Výsledkem dělení dvou operandů typu ***int*** je celá část podílu:
  Příklad:
  7/3 je 2 -7/3 je -2
- Pro zbytek po dělení platí: ***x % y = x – (x / y) * y***
  Příklad:
  7%3 je 1 -7%3 je -1 7%-3 je -1 -7%-3 je -1
- Speciální inkrementační a dekrementační operátory:
  `++x, x++, --x, x--` rozdílná návratová hodnota podle polohy `x`

Relační operátory

- Hodnoty všech jednoduchých typů jsou uspořádané
  - lze je porovnávat relačními operátory (priorita je nižší, než aritmetických operátorů):

`>, <, >=, <=` (větší než, menší než, větší nebo rovno, menší nebo rovno)
`==, !=` (rovná se, nerovná se)

- Výsledek relační operace je typu *boolean* (*true*, *false*)
- Jestliže jsou číselné operandy různého typu, operand typu ***int*** se konvertuje na typ ***double***

- Příklady relačních výrazů:

```
int i=10;
double x=12.3;
boolean b;
System.out.println(i==10);      // vypíše true
System.out.println(i+1==10);    // vypíše false
b = i>x;                        // proměnné b se přiřadí false
```

Logické operátory

- Logické operátory jsou definovány pro operandy typu *boolean*

unární	*!*	(negace)
binární	&& resp. &	(konjunkce, logický součin)
binární	|| resp. |	(disjunkce, logický součet)

x	*y*	*!x*	*x && y*	*x || y*
false	*false*	*true*	*false*	*false*
false	*true*	*true*	*false*	*true*
true	*false*	*false*	*false*	*true*
true	*true*	*false*	*true*	*true*

- Negace má stejnou prioritu, jako změna znaménka, logický součin má nižší prioritu než relační operátory
- Operace && a || se vyhodnocují zkráceným způsobem
  - druhý operand se nevyhodnocuje, jestliže lze výsledek určit již z prvního operandu

```
int n = 10; boolean b1 = false, b2 = true;
System.out.println(1<=n && n<=20);      // vypíše se true
System.out.println(b1 || !b2);          // vypíše se false
if (y != 0 && x/y < z)                  // zkrácené vyhodnocení
```

Matematické funkce

- Při číselných výpočtech často potřebujeme matematické funkce jako *abs*, *sin*, *cos*, *sqrt* (druhá odmocnina), *log* (přirozený logaritmus) atd.
- Tyto funkce poskytuje knihovní třída *Math*

```
package prepona;
import java.util.*;

public class Prepona {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Zadejte délku odvěsen pravoúhlého
                            trojúhelníka");
        double x = sc.nextDouble();
        double y = sc.nextDouble();
        double z = Math.sqrt(x * x + y * y);
        // double z = Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2));
        System.out.println("Délka přepony je " + z);
    }}
```

- Knihovní třídu *Math* není třeba importovat příkazem *import* – je součástí implicitního balíku *java.lang*

Knihovní třída Math

- Některé z funkcí poskytovaných třídou *Math*
  - `int abs(int a)`
  - `double abs(double a)`
  - `int max(int a, int b)`
  - `double max(double a, double b)`
  - `int min(int a, int b)`
  - `double min(double a, double b)`
  - `double sqrt(double a)`
  - `double sin(double a)`
  - `double random()` – pseudonáhodné číslo větší nebo rovno *0.0* a menší než *1.0*
- Třída poskytuje též dvě konstanty: *E* a *PI*

Knihovní třída Math - příklad

```
package obvodkruhu;
import java.util.*;

public class ObvodKruhu {

      public static void main(String[] args) {
          Scanner sc = new Scanner(System.in);
        System.out.println("zadejte poloměr kruhu");
        double r = sc.nextInt();
        System.out.println("obvod kruhu je " + 2 * Math.PI * r);
    }
}
```

Vstup a výstup v Javě

Příkazy výstupu

- Výpis dat na obrazovku - příkaz z knihovny ***System***:
  `System.out.println(parametr);`
  - parametrem musí být výraz typu *String* (řetězec)
- Příklad výpisu textu: ***`System.out.println("Nazdar");`***

- Pro každý datový typ existuje implicitní konverze na typ ***String*** (implicitně volaná metoda ***toString***) - parametrem příkazu výstupu tak může být libovolná proměnná či výraz

- Příklad výpisu hodnoty proměnné n typu *int*:
  `System.out.println(n);`

- Řetězce lze spojovat pomocí operátoru +; je tedy dovolen např. tento příkaz:
  `System.out.println("hodnota proměnné n = " + n);`

Zápis programu v Javě

- Vzhledem k objektové podstatě jazyka Java je nutné pracovat s komplikovanější syntaxí
- **Příklad**:

```
package mujprogram;

public class MujProgram {
  public static void main(String[] args) {
    int x = 10, y;
    y = x + 20;
    System.out.println("hodnota proměnné x je "+x);
    System.out.println("hodnota proměnné y je "+y);
    System.out.println("Cislo Pi = ", 3.14);
  }
}
```

- Program po překladu a spuštění vypíše:

  hodnota proměnné ***x*** je ***10***
  hodnota proměnné ***y*** je ***30***
  Cislo Pi = 3,14

Formátovaný výstup

Příklad:

```
System.out.printf("Cislo Pi = %6.2f %n ", 3.142);
```

vypíše na obrazovku: *Cislo Pi = 3,14*

Specifikace výstupního formátu:

%[$indexParametru][modifikátor][šířka][.přesnost]konverze

- konverze - povinný parametr
  - typ celé číslo *d,o,x* - dekadicky, oktalově a hexadecimálně
  - typ *double f* je desetinný zápis; *e,E* vědecký s exponentem
- šířka - počet sázených míst, zarovnání vpravo
- přesnost - počet desetinných míst
- *modifikátor* - v závislosti na typu konverze určuje další vlastnosti, například pro konverzi f (typ *double*):
  - symbol + určuje, že má být vždy sázeno znaménko,
  - symbol - určuje zarovnání vlevo,
  - symbol 0 doplnění čísla zleva nulami.
- *%n* - přechod na další řádek
- a další (formátování data, měny, …)

Poznámka k příkazu výstupu

Co vypíší následující příkazy:

příkazy:	výstup:
`int x = 1, y = 2;`	
`System.out.println(x+y);`	*3*
`System.out.println("součet je "+(x+y));`	*součet je 3*
`System.out.println("součet je "+x+y);`	*součet je 12*

Význam operátoru + (a většiny dalších) je odvozen od typu na jeho levé straně (je asociativní zleva).

Výraz: `"součet je " + x + y`

se vyhodnotí jako `("součet je " + x) + y`

Vstup dat pomocí třídy Scanner

- Jedna z možností je použití třídy Scanner z knihovny java.util

- Příklad:

Zadejte dvě celá čísla
1
2
Součet čísel: 1 + 2 = 3

```
package program3;
import java.util.Scanner;
public class Program3 {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int x, y, z;
      System.out.println("Zadejte dvě celá čísla");
      x = sc.nextInt();
      y = sc.nextInt();
      z = x + y;
      System.out.println("Součet čísel: " +x +" + "+y+" = "+z);
   }
}
```

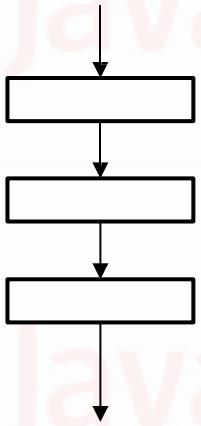
Vstup pomocí třídy Scanner

- Třída *Scanner* musí být připojena k programu:
  - `import java.util.*;` //připojení knihovny *java.util*
- Je třeba vytvořit objekt třídy *Scanner* a napojit jej na standardní vstupní proud:
  - `Scanner sc = new Scanner(System.in);`
- Metody třídy Scanner
  - ***sc.nextInt()*** - přečte celé číslo z řádku zadaného klávesnicí (řádek je zakončen klávesou ***Enter***, číslo je zakončeno mezerou nebo ***Enter***) a vrátí je jako hodnotu typu ***int***
  - ***sc.nextDouble()*** - přečte číslo z řádku zadaného klávesnicí a vrátí je jako hodnotu typu ***double***, jako oddělovač použijte znak '.' nebo znak ',' v závislosti na definovaném OS. Toto lze změnit před vytvořením Scanneru ***Locale.setDefault(Locale.ENGLISH);***
  - ***sc.nextLine()*** - přečte zbytek řádku zadaného klávesnicí a vrátí je jako funkční hodnotu typu ***String***

Základní příkazy Javy

Posloupnost (sekvence)

- Složený příkaz: { <posloupnost příkazů jazyka> }
- Blok: { <posloupnost deklarací a příkazů>}

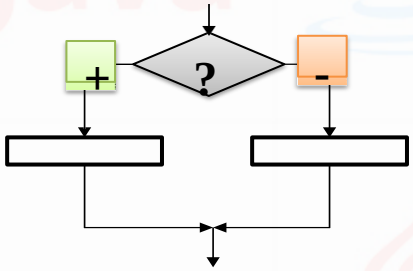


Poznámka:

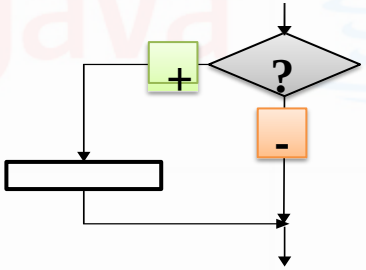
- Deklarace proměnných jsou v bloku lokální, tzn. neplatí vně bloku
- V rámci bloku se mohou vyskytovat nejen přiřazovací příkazy ale i větvení a cykly

Příkaz větvení

- Příkaz **if** (podmíněný příkaz)
- Dvě varianty
  - if-else - úplný
  - if - neúplný




```
if (x < y)
  min = x;
else
  min = y;
System.out.println(min);
```




```
min = x;
if (y < min) min = y;
System.out.println(min);
```

Příkaz **if** - vlastnosti

- Jestliže v případě splnění či nesplnění podmínky má být provedeno více příkazů, je třeba z nich vytvořit složený příkaz nebo blok

- Příklad: jestliže ***x*** < ***y***, vyměňte hodnoty těchto proměnných:

Špatně:

```
if (x < y)
   pom = x;                  //tady končí větvení
x = y;                       //x vždy nabude hodnoty y a y hodnoty pom
y = pom;
```

Správně (složený příkaz):

```
if (x < y) {
   pom = x;
   x = y;
   y = pom;
}
```

Příkaz **if** - vnoření

- Do příkazu ***if*** lze **vnořit** libovolný příkaz, tedy i podmíněný příkaz

- **Příklad**: do ***s*** uložte ***–1***, ***0*** nebo ***1*** podle toho, zda ***x*** je menší než nula, rovno nule nebo větší než nula

```
if (x < 0) s = -1;
else
    if (x == 0)
        s = 0;
    else s = 1;
```

- **Pozn.**:
  - ***else*** patří k nejbližšímu předchozímu ***if***
  - formátor *IDE NetBeans* vždy doplní závorky
  - pozor na vnoření **neúplného *if*** do **úplného *if***

```
if (x > y)
if (x > z) max = x;
else max = z;
else
if (y > z) max = y;
else max = z;
```

- **Příklad**: do ***max*** uložte největší z čísel ***x***, ***y*** a ***z***

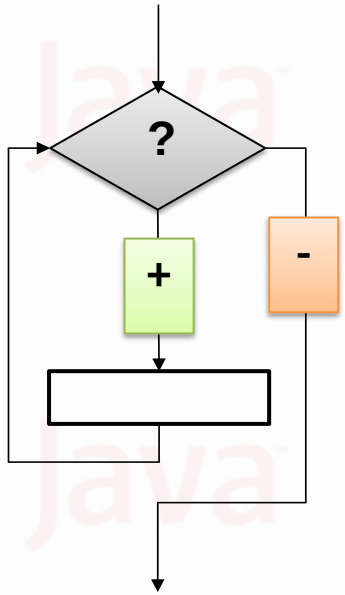
Příklad **if**

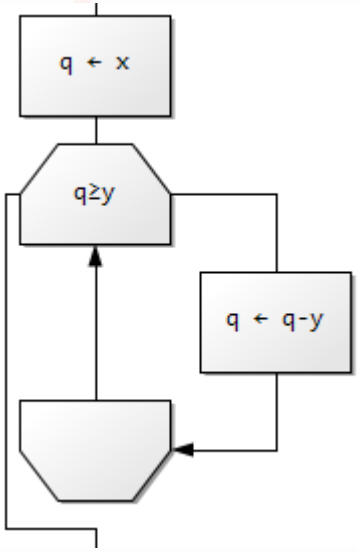
- **Příklad**
  - program, který pro zadaný rok zjistí, zda je přestupný
  - Přestupný rok musí být dělitelný 4 **a současně** (není dělitelný 100 **nebo** je dělitelný 400)

```
package rok;

public class Rok {

    public static void main(String[] args) {
        int rok = 2020;
        System.out.print("rok " + rok);
        if (rok % 4 == 0 && (rok % 100 != 0 || rok % 400 == 0)) {
            System.out.println("je přestupný");
        } else {
            System.out.println("není přestupný");
        }
    }
}
```

Příkaz cyklu **while**

- Cyklus s podmínkou na začátku
  - ***while*** **(***podmínka***)** *příkaz;*





Co dané příkazy řeší?

```
q = x;
while (q>=y)
   q = q-y;
```

```
q = x;
p = 0;
while (q>=y){
   q = q-y;
   p = p+1;
}
```

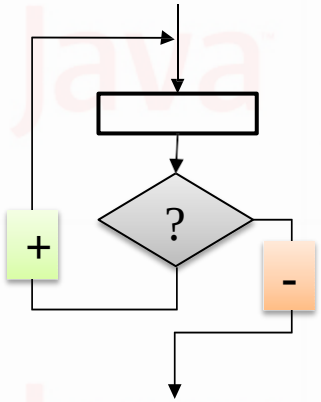
Příklad

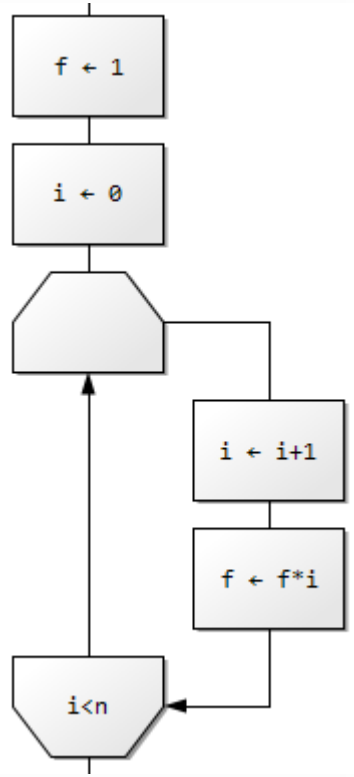
Výpočet faktoriálu přirozeného čísla ***n*** ($n! = 1 \times 2 \times \ldots \times n$)

```
package faktorial;
import java.util.Scanner;

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

Příkaz **do**

- Cyklus s podmínkou na konci
  do *příkaz* ***while*** ***(****podmínka****);***



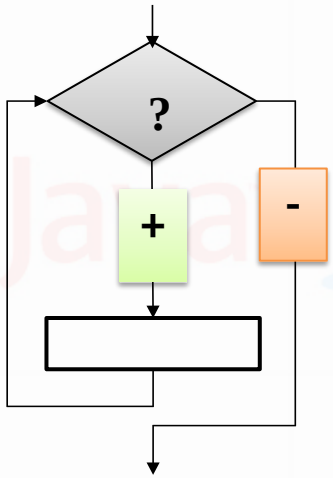


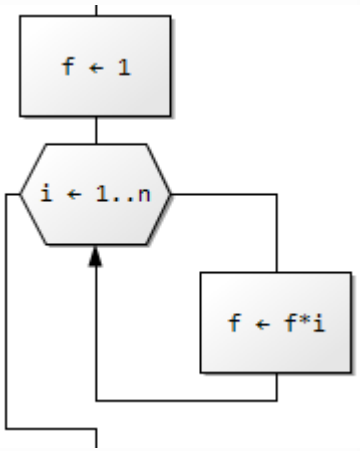
- **Příklad** (faktoriál):

```
f = 1;
i = 0;
do {
    i = i + 1;
    f = f * i;
 } while (i<n);
```

Příkaz **for**

- Cyklus s daným počtem opakování
  - často řízen proměnnou, pro kterou je stanovena počáteční hodnota (inicializace řídící proměnné), koncová hodnota (podmínka opakování/ukončení cyklu), nastavení způsobu změny hodnoty po každém provedení těla cyklu (aktualizace řídící proměnné)
- Schematicky stejné jako *while*





- Cykly tohoto druhu lze zkráceně předepsat příkazem ***for***:

```
f = 1;
for (i=1; i<=n; i=i+1)
   f=f*i;
```

Příkaz **for**

- Tvar příkazu *for:*
  for (inicializace ; podmínka ; aktualizace) příkaz;

- Provedení příkazu *for* :
  inicializace;
  while (podmínka) {
  příkaz
  změna;
  }

- Změnu řídicí proměnné lze zkráceně předepsat pomocí operátoru inkrementace resp. dekrementace:

x++	***x*** se zvětší o ***1***
x--	***x*** se zmenší o ***1***

- **Příklad:**

```
f = 1;
for (i=1; i<=n; i++)
   f=f*i;
```

Příkaz continue

- Někdy je třeba ukončit cyklus v nějakém místě uvnitř těla cyklu (které je v tom případě tvořeno složeným příkazem)
- Příkaz *continue* předepisuje předčasné ukončení průchodu těla cyklu
- **Příklad:**

```
for (int i=1; i<=100; i++) {
  if (i%10==0) continue;
  System.out.println(i);
}
```

příkaz vypíše čísla od ***1*** do ***100*** s výjimkou čísel dělitelných ***10***

Příkaz break

- Příkaz ***break*** (obvykle vnořený do podmíněného příkazu v rámci těla cyklu) ukončí předčasně celý příkaz cyklu

- Schematicky:

```
while (...) {
  ...
  if (ukončit) break;
  ...
}
```

- **Příklad:**

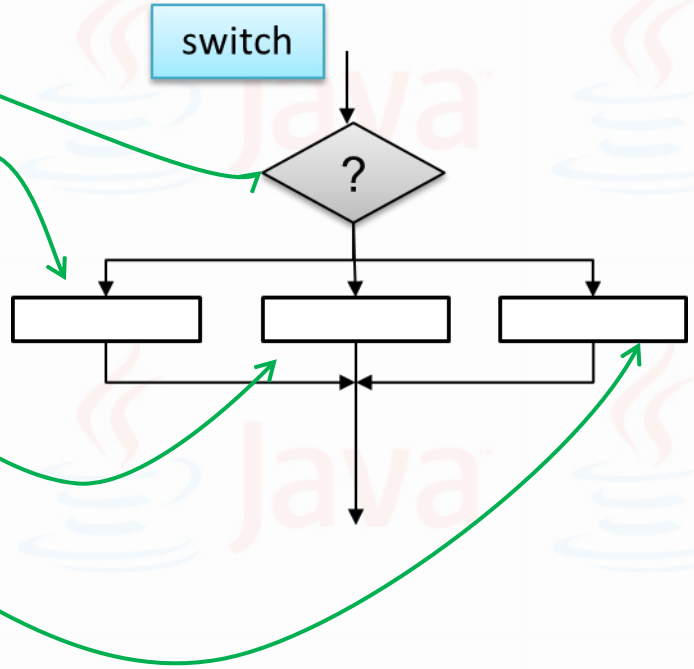
```
for (int i=1; i<=100; i++) {
   if (i%10==0) break;
   System.out.println(i);
}
```

příkaz vypíše čísla od ***1*** do ***9***

Další příkaz pro větvení - **switch**

- Switch umožňuje vícenásobné větvení

```
switch (výraz)
{
 case hodnota_1 příkaz_1;
 :                  break;
          …
 case hodnota_npříkaz_n;
 :                  break;

 default:       příkaz_def;
                  break;
}
```



Příkaz **switch** - vlastnosti

- Rozhodovací výraz smí být pouze typu:
  - *char*, *byte*, *short*, *int*
- Počet větví není omezen
- V každé větvi může být skupina příkazů, není nutno uzavírat do ***{}***
- Větev ***default*** se provádí pokud žádná větev case nevyhovuje
- Provádění končí na ***break, return*** nebo na konci přepínače
- Není-li větev ukončena provedením příkazu ***break***, pokračuje seprováděním příkazů další větve

Příkaz **switch** - příklad

Příklad:

– program vypíše 123 po stisku klávesy a nebo b nebo c, po stisku d vypíše 23, jinak 3

```
switch (sc.nextLine().charAt(0)) {
    case 'a' :
    case 'b' :
    case 'c' : System.out.println("1");
    case 'd' : System.out.println("2");
    default : System.out.println("3");
}
```

Zpracování posloupností

- **Příklad**: program pro součet posloupnosti čísel
- Hrubé řešení:

```
suma = 0;
while (nejsou přečtena všechna čísla) {
  dalsi = přečti celé číslo;
  suma = suma+dalsi;
}
```

- Varianty podmínky
  1. počet čísel bude vždy stejný, např. ***5***
  2. počet čísel bude dán na začátku vstupních dat
  3. vstupní data budou končit "*zarážkou*", např. nulou

- Struktura vstupních dat formálně:
  1. $a_1\ a_2\ a_3\ a_4\ a_5$
  2. $n\ a_1\ a_2 \ldots a_n$
  3. $a_1\ \ a_2\ \ \ldots\ \ a_5\ \ 0$ kde $a_i \neq 0$

Zpracování posloupností

- **Řešení 1** - vstupní data: $a_1\, a_2\, a_3\, a_4\, a_5$

```
package suma1;
import java.util.*;

public class Suma1 {
  public static void main(String[] args) {
   Scanner sc = new Scanner(System.in);
   int dalsi, suma, i;
    System.out.println ("zadejte 5 čísel");
    suma = 0;
    for (i=1; i<=5; i++) {
      dalsi = sc.nextInt();
      suma = suma+dalsi;
    }
    System.out.println ("součet je " + suma);
  }
}
```

Zpracování posloupností

- **Řešení 2** - vstupní data: $n\ a_1\ a_2 \ldots a_n$

```
package suma2;
import java.util.Scanner;

public class Suma2 {
  public static void main(String[] args) {
    int dalsi, suma, i, n;
    Scanner sc = new Scanner(System.in);
    System.out.println("zadejte počet čísel");
    n = sc.nextInt();
    System.out.println("zadejte " + n +" čísel");
    suma = 0;
    for (i=1; i<=n; i++) {
      dalsi = sc.nextInt();
      suma = suma+dalsi;
    }
    System.out.println("součet je " + suma);
  }
}
```

Zpracování posloupností

Řešení 3 - vstupní data: $a_1\ a_2 \ldots a_n\ 0$, kde a_i není 0

```
package suma3;
import java.util.Scanner;

public class Suma3 {
public static void main(String[] args) {
    int dalsi, suma;
    Scanner sc = new Scanner(System.in);
    System.out.println("zadejte řadu čísel zakončenou nulou");
    suma = 0;
    dalsi = sc.nextInt();
    while (dalsi != 0) {
        suma = suma + dalsi;
        dalsi = sc.nextInt();
    }
    System.out.println("součet je " + suma);
}
}
```

Procedury a funkce

Code reusability

- Funkce a procedury jsou příkladem nástrojů pro zajištění opakované použitelnosti kódu
  - jedná se o pojmenované části kódu, které lze vyvolávat opakovaně, ev. s různými vstupními parametry
- Rozdíl
  - funkce vrací hodnotu
  - procedura nevrací hodnotu
- V jazyce Java se obecně hovoří o metodách

Funkce

- Program pro výpočet faktoriálu (bez importů)

```
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 vstupu

algoritmus výpočtu faktoriálu

- Jde o dva dílčí podproblémy a úkoly – vhodné použít samostatné funkce

Faktoriál pomocí funkcí

- Funkce pro čtení přirozeného čísla

```
static int ctiPrirozene() {          // hlavička metody
    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 - funkce nemá parametry a výsledkem volání funkce (návratovou hodnotou) je hodnota typu ***int***
- Příkaz ***return*** definuje co bude návratovou hodnotou - výsledkem volání funkce 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;
}
```

- Funkce má jeden parametr typu *int* a výsledkem je hodnota typu *int*
  - reálný argument funkce je zastupován označením *n*

- Příklad volání funkce: `int ff = faktorial(4);`

Faktoriál pomocí funkcí

- Výsledné řešení

```
package faktorial;
import java.util.*;
public class Faktorial {
      static int ctiPrirozene() {
            ...
      }
      static int faktorial(int n) {
            ...
      }
      public static void main(String[] args) {
            int n = ctiPrirozene();
            System.out.println (n + "! = " + faktorial(n));
   }
}
```

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

```
System.out.println(
  "Fakt" + "! = " + faktorial(ctiPrirozene()));
```

Faktoriál pomocí funkcí - celek

```
package faktf;
import java.util.Scanner;
public class FaktF {
    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;
    }
    static int faktorial(int n) {
        int i = 1;
        int f = 1;
        while (i < n) {
            i = i + 1;
            f = f * i;
        }
        return f;
    }
    public static void main(String[] args) {
        System.out.println("faktoriál je "+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 (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 a musí končit příkazem ***return x***
  - *x* je *výraz*, jehož hodnota je výsledkem volání funkce

Parametry funkce

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

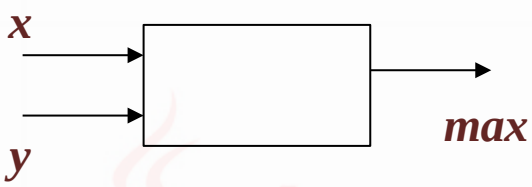
```
public class Max1 {
    static int max(int x, int y) {
            if (x>y) return x; else return y;
    }
    public static void main(String[] args) {
            int a = 10, b = 20;
            System.out.println(max(a+20, b)); // OK
            System.out.println(max(1.1, b));  // Chyba při překladu
    }
}
```

Parametry funkce

- Parametry funkce slouží pro předání vstupních dat algoritmu, který je funkcí realizován
- Častá chyba začátečníka:

  čtení hodnoty parametrů pomocí operace vstupu dat

```
static int max(int x, int y) {
   x = sc.nextInt();          // přepis x (ale nic se nehlásí)
   y = sc.nextInt();          // přepis y (ale nic se nehlásí)
   if (x>y)  return x;
  else  return y;
}

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



Přetěžování jmen funkcí

- Funkce lišící se v počtu nebo typu parametrů se mohou jmenovat stejně tzv.: přetěžování jmen (overloading of names) – neplést s tzv.: překrýváním

```
package max2;
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

- Zjištění, zda daný rok je přestupný

```
public class Rok {
 static boolean prestupny(int rok) {
  return (rok%4==0 && (rok%100!=0 || rok%400==0));
 }

 public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
  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

Jeden z algoritmů výpočtu největšího společného dělitele

1. *je-li x = y, pak nsd(x,y) = x*
2. *je-li x > y, pak nsd(x,y) = nsd(x-y,y)*
3. *je-li x < y, pak nsd(x,y) = nsd(x,y-x)*

Příklad:

16,12

4,12

4,8

4,4

```
static int nsd(int x, int y) {
    while (x != y) {
        if (x > y) x = x - y;
        else y = y - x;
    }
return x;
}
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Zadej a:");
        int a = sc.nextInt();
        System.out.println("Zadej b:");
        int b = sc.nextInt();
        System.out.println("NSD čísel "+a+", "+b+" je "+nsd(a,b));
    }
```

Procedury

- Metoda, která nevrací žádnou hodnotu
  - typ výsledku je *void*

- 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);
}
```

Statické proměnné

- Proměnné definované a společné pro danou třídu (bude vysvětleno později) použitelné ve všech metodách dané třídy jako nelokální proměnné

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

rozsah platnosti x,y - scope

- Poznámka: statické číselné proměnné jsou inicializovány hodnotou nula

Zastínění nelokální proměnné

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

- Příklad:

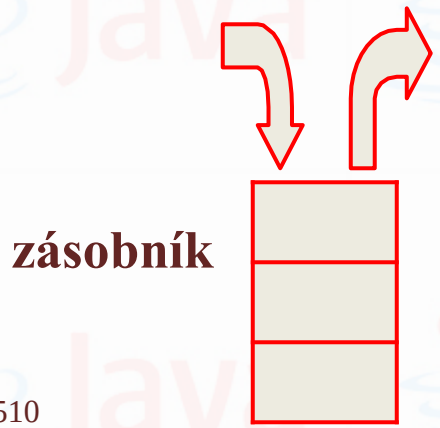
```
public class Zastineni {
   static int a = 10;
   public static void main(String[] args) {
      f();
   System.out.println(a);
   }
   static void f() {
      int a = 20;
   System.out.println(a);  // Zastineni.a
   }
}
```

Program vypíše:

20
10

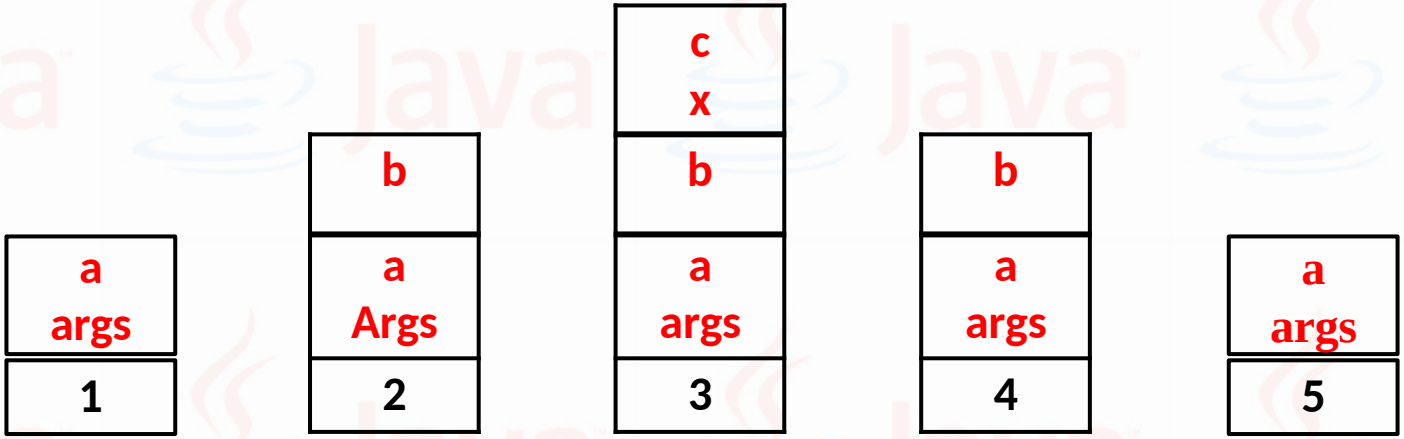
Přidělování paměti proměnným

- Přidělení paměti - určení adresy umístění proměnné v paměti počítače
- Poznali jsme doposud dva druhy proměnných:
  - statické proměnné třídy
  - lokální proměnné funkcí
- Statickým proměnným třídy se přidělí paměť v okamžiku, kdy se do paměti zavádí kód metod 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 automaticky 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



Přidělování paměti proměnným

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



Jednoduchá kolekce - Pole

Pole

- Strukturovaný datový typ skládající se z pevného počtu prvků stejného typu, identifikovatelných pomocí indexu
  - např. teploty naměřené v jednotlivých dnech v týdnu
  - úloha - vypočítat průměrnou teplotu a pro každý den vypsat odchylku od průměru
  - řešení bez pole (s proměnnými typu *int*) – řada nevýhod

```
int t1, t2, t3, t4, t5, t6, t7, prumer;
…      // načtení hodnot
prumer = (t1+t2+t3+t4+t5+t6+t7)/7;
System.out.println(t1-prumer);
...
System.out.println(t7-prumer);
```

- V Javě se pole indexuje celými čísly *0*, *1*, ... *(počet prvků – 1)*
  - počet prvků je dán při vytvoření pole a je neměnitelný

Pole

- Řešení pomocí pole
  - pole 7 prvků typu `int`

```
int [] teploty = new int[7];
```

  - načtení dat cyklem

```
for (int i=0; i<7; i++)
   teploty[i] = nacteni();
```

  - průměrná teplota jako součet prvků pole dělený 7:

```
int prumer = 0;
for (int i=0; i<7; i++)
   prumer = prumer + teploty[i];
prumer = prumer / 7;
```

  - výpis odchylek od průměru:

```
for (int i=0; i<7; i++)
   System.out.println(teploty[i]-prumer);
```

Pole v jazyku Java

- Pole ***p*** obsahující ***n*** prvků typu T vytvoříme deklarací

```
T[] p = new T[n];
```

  - T může být libovolný typ a ***n*** musí být celočíselný výraz s nezápornou hodnotou
  - prvky takto zavedeného pole mají nulové hodnoty
- Lze vytvořit i pole tvořené prvky s danými hodnotami

```
int p[] = {1,2,3,4,5,6};
```

- Zápis: `p[i]`
  - kde ***i*** je celočíselný výraz, jehož hodnota je nezáporná a menší než počet prvků, označuje prvek pole ***p*** s indexem ***i*** a má vlastnosti proměnné typu T
  - nedovolená hodnota indexu způsobí chybu při výpočtu
- Počet prvků pole p lze zjistit pomocí zápisu **`p.length`**
- Příklad:

```
for(int i=0; i<p.length; i++) System.out.println(p[i]);
```

Příklad

- Vstup: $n\ a_1\ a_2 \ldots\ a_n$, kde n, a_i jsou celá čísla
- Výstup: čísla a_i v opačném pořadí

- Řešení:

```
package obratpole1;

public class ObratPole1 {
 public static void main(String[] args) {
    System.out.println("zadejte počet čísel");
    int[] pole = new int[Sys.readInt()];
    System.out.println("zadejte "+pole.length+"
    čísel");
    for (int i=0; i<pole.length; i++)
      pole[i] = Sys.readInt();
    System.out.println("výpis čísel v obráceném
    pořadí");
    for (int i=pole.length-1; i>=0; i--)
      System.out.println(pole[i]);
 }
}
```

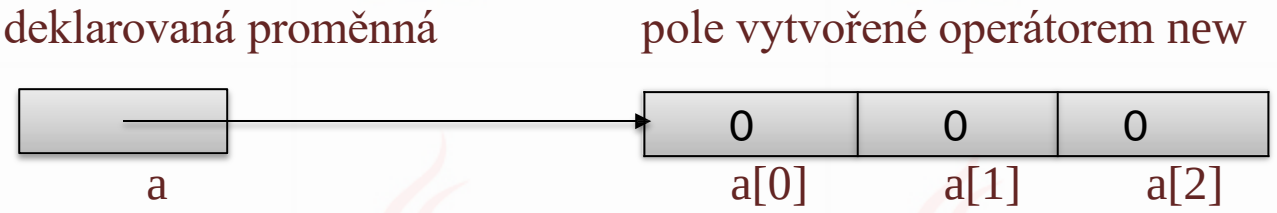
Přidělení paměti poli

- Např. lokální deklarace, která vytvoří pole 3 prvků typu int:

```
int[] a = new int [3];
```

- Deklarace má tento efekt
  - lokální proměnné ***a*** se přidělí paměťové místo na zásobníku, které však neobsahuje prvky pole, ale je dimenzováno na uložení čísla reprezentujícího adresu jiného paměťového místa (tzv. hromada, heap)
  - operátorem ***new*** se v jiné paměťové oblasti rezervuje (alokuje) úsek potřebný pro pole 3 prvků typu *int*
  - adresa tohoto úseku se uloží do **a**

- Při grafickém znázornění reprezentace v paměti místo adres kreslíme šipky



Pozn: reprezentace pole je zde zjednodušená, obsahuje ještě počet prvků

Referenční proměnné pole

- Shrnutí:
  - pole ***n*** prvků typu *T* lze v jazyku Java vytvořit pouze dynamicky pomocí operace ***new*** *T*[*n*]
  - adresu dynamicky vytvořeného pole prvků typu *T* lze uložit do proměnné typu *T[ ]*; takovou proměnnou nazýváme referenční proměnnou pole prvků typu *T*

- Referenční proměnnou pole lze deklarovat bez i vytvoření pole; Deklarací `int[] a;` se zavede referenční proměnná, která má:
  - nedefinovanou hodnotu, jde-li o lokální proměnnou,
  - nebo speciální hodnotu ***null***, která nereferencuje žádné pole

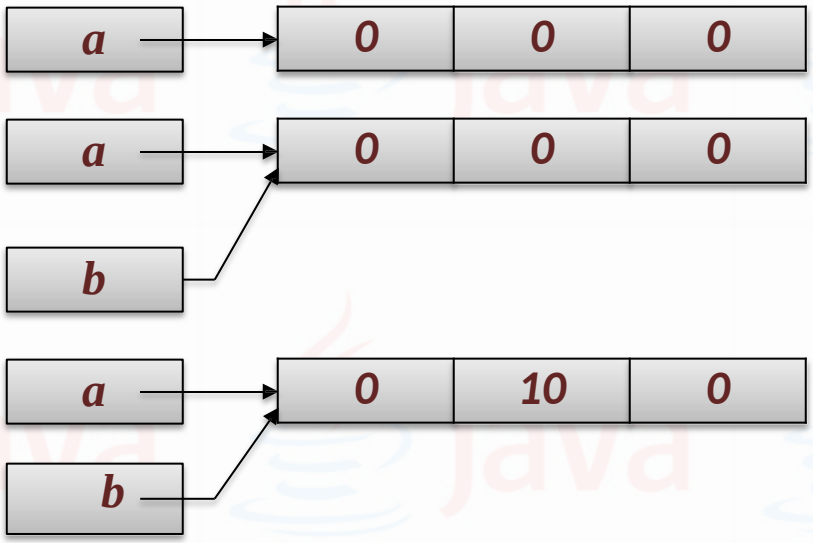
- V obou předchozích případech je třeba, před dalším použitím referenční proměnné pole, jí přiřadit referenci na vytvořené pole, např. příkazem `a = new int[10];`

Přiřazení mezi referenčními proměnnými pole

- V jazyku Java je dovoleno přiřazení mezi dvěma referenčními proměnnými polí stejných typů
- Po přiřazení pak obě proměnné referencují totéž pole

- Příklad:

```
int[] a = new int[3];

int[] b = a;

b[1] = 10;
```




```
System.out.println(a[1]);
```
// vypíše se 10

- **Pozn:** Přiřazení mezi dvěma poli není v jazyku Java definováno!

Pole jako parametr a výsledek funkce

- Reference pole může být parametrem funkce i jejím výsledkem
  - pole vytvořené uvnitř metody existuje i mimo ni, dojde-li k předání reference např. návratovou hodnotou

- Příklad: načtení pole, obrácení pořadí prvků pole, výpis pole

```
package obratpole2;

public class ObratPole2 {
 public static void main(String[] args) {
  int[] vstupniPole = ctiPole();
  int[] vystupniPole = obratPole(vstupniPole);
  vypisPole(vystupniPole);
 }

 static int[] ctiPole() { ... } //alokuje pole, naplní ho a vrátí referenci
  static int[] obratPole(int[] pole) { ... } //parametr + výsledek
  static void vypisPole(int[] pole) { ... } //parametr
}
```

Změna pole daného parametrem

- Varianta řešení bez alokace dalšího pole - metodou nevytvoříme nové pole, ale obrátíme pole dané parametrem:

```
static void obratPole(int[] pole) {
 int pom;
 for (int i=0; i<pole.length/2; i++) {
  pom = pole[i];
  pole[i] = pole[pole.length-1-i];
  pole[pole.length-1-i] = pom;
 }
}
```

- Použití:

```
public static void main(String[] args) {
 int[] vstupniPole = ctiPole();
 obratPole(vstupniPole);
 vypisPole(vstupniPole);
}
```

- Metoda *obratPole* dostane referenci na *vstupniPole*

Pole jako tabulka

- Pole lze použít též pro realizaci tabulky (zobrazení), která hodnotám typu indexu (v jazyku Java to je pouze interval celých čísel počínaje nulou) přiřazuje hodnoty nějakého typu

- Příklad: přečíst řadu čísel zakončených nulou a vypsat tabulku četnosti čísel od *1*do *100* (ostatní čísla ignorovat)
  - Tabulka četnosti bude pole *100* prvků typu *int*, počet výskytů čísla *x*, kde $1 \leq x \leq 100$, bude hodnotou prvku s indexem *x-1*
  - Aby program byl snadno modifikovatelný pro jiný interval čísel, zavedeme dvě konstanty:
    - `final static int MIN = 1;`
    - `final static int MAX = 100;`
  - Pole vytvoříme s počtem prvků *Max-Min+1*

```
int[] tab = new int[MAX-MIN+1];
```

Příklad – tabulka četnosti čísel

- Funkce, která přečte čísla a vytvoří tabulku četnosti

```
static int[] tabulka() {
 int[] tab = new int[MAX-MIN+1];
 System.out.println("zadejte řadu celých čísel zakončenou nulou");
 int cislo = sc.readInt();
 while (cislo!=0) {                          //na indexu 0 je četnost MIN atd.
  if (cislo>=MIN && cislo<=MAX) tab[cislo-MIN]++;
  cislo = sc.readInt();
 }
 return tab;
}
```

- Funkce, která tabulku četnosti vypíše

```
static void vypis(int[] tab) {
 for (int i=0; i<tab.length; i++)
    if (tab[i]!=0)
       System.out.println("četnost čísla "+(i+MIN)+" je "+tab[i]);
}
```

Příklad – tabulka četnosti čísel

- Celkové řešení:

```
public class CetnostCisel {

 final static int MIN = 1;
 final static int MAX = 100;

 public static void main(String[] args) {
  vypis(tabulka());
 }

 static int[] tabulka() {
  ...
 }

 static void vypis(int[] tab) {
    ...
 }
}
```

Příklad – tabulka četnosti čísel

```
package uswa6;

public class CetnostCisel {
    final static int MIN = 1;
    final static int MAX = 100;

    public static void main(String[] args) {
        vypis(tabulka());
    }

    static int[] tabulka() {
         Scanner sc = new Scanner(System.in);
        int[] tab = new int[MAX - MIN + 1];
        System.out.println("zadejte řadu celých čísel zakončenou nulou");
        int cislo = sc.readInt();
        while (cislo != 0) {
            if (cislo >= MIN && cislo <= MAX) tab[cislo - MIN]++;
            cislo = sc.readInt();
        }
        return tab;
    }

    static void vypis(int[] tab) {
        for (int i = 0; i < tab.length; i++) {
            if (tab[i] != 0) {
                System.out.println(„četnost čísla " + (i + MIN) + " je " + tab[i]);
            }
        }
    }
}
```

Pole reprezentující množinu

- Příklad: vypsat všechna prvočísla menší nebo rovna zadanému *max*
  - Algoritmus:
    1. Vytvoříme množinu obsahující všechna přirozená čísla od *2* do *max*.
    2. Z množiny vypustíme všechny násobky čísla *2*.
    3. Najdeme nejbližší číslo k tomu, jehož násobky jsme v předchozím kroku vypustili, a vypustíme všechny násobky tohoto čísla.
    4. Opakujeme krok *3*, dokud číslo, jehož násobky jsme vypustili, není větší než *odmocnina* z *max*.
    5. Čísla, která v množině zůstanou, jsou hledaná prvočísla.
  - pro reprezentaci množiny čísel použijeme pole prvků typu *boolean*
  - prvek *mnozina[x]* bude udávat, zda číslo *x* v množině je (*true*) nebo není (*false*)
  - prvky pole *mnozina* s indexem *0* a *1* nebudou využity

Příklad - Eratosthenovo síto

- Funkce pro vytvoření množiny prvočísel do ***max***:

```
static boolean[] sito(int max) {
 boolean[] mnozina = new boolean[max+1];
  for (int i=2; i<=max; i++)
   mnozina[i] = true;                    //inicializace na true
  int p = 2;
  int pmax = (int)Math.sqrt(max);        //zarážka
  do {
   // vypuštění všech násobků čísla p
   for (int i=p+p; i<=max; i+=p)
    mnozina[i] = false;
   // hledání nejbližšího čísla k p  do {
   p++;   //do p nejbližší další dosud nevypuštěné číslo
  } while (!mnozina[p]);
 } while (p<=pmax);
 return mnozina;
}
```

Příklad - Eratosthenovo síto

- Funkce pro výpis množiny

```
static void vypis(boolean[] mnozina) {
 for (int i=2; i<mnozina.length; i++)
  if (mnozina[i]) System.out.println(i);
}
```

- Hlavní funkce

```
public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 System.out.println("zadejte max");
 int max = sc.readInt();
 boolean[] mnozina = sito(max);
 System.out.println("prvočísla od 2 do "+max);
 vypis(mnozina);
}
```

Vícerozměrné pole

- Vícerozměrným polem se obecně rozumí takové pole, k jehož prvkům se přistupuje pomocí více než jednoho indexu
- V Javě vícerozměrná pole jako pole, jejichž prvky jsou opět pole
- Příklady dvojrozměrného pole prvků typu *int*:
  - deklarace referenční proměnné:
    ```
    int mat[][];
    ```
  - vytvoření pole se 3 x 4 prvky (3 řádky, 4 sloupce):
    ```
    mat = new int[3][4];      // prvky mají hodnotu 0
    ```
  - deklarace referenční proměnné a vytvoření pole 3 x 4 s inicializací:
    ```
    int mat[][] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
    ```
  - součet všech prvků pole *mat*
    ```
    int suma = 0;
    for (int i=0; i<mat.length; i++)
     for (int j=0; j<mat[i].length; j++)
      suma += mat[i][j];
    ```