

Řetězce a řídicí struktury

Jan Faigl

Katedra počítačů

Fakulta elektrotechnická

České vysoké učení technické v Praze

Přednáška 3

A0B36PR1 – Programování 1

Část 1 – Soubory a vytvoření projektu

Diskové oddíly, soubory a domovský adresář

Vytvoření projektu v Netbeans

Část 2 – Připomínka (NSD)

Výpočetní problém, algoritmus a program

Výpočet největšího společného dělitele

Část 3 – Textové řetězce, operátory a načítání vstupu

Reprezentace znaku

Textový Řetězec

Vstup programu

Část 4 – Řídicí struktury

Řídicí struktury

Větvení

Cykly

Část I

Soubory a vytvoření projektu

Přístup k souborům v operačním systému

- V operačním systému pracujeme se soubory, které jsou uloženy na lokálních nebo síťových discích

- Soubory jsou uloženy v adresářové struktuře

Strukturu lze reprezentovat grafem, zpravidla stromem

- Struktura začíná **kořenovým adresářem** označovaným /

- Disková uložení se připojují jako adresáře

Zajišťuje unifikovaný přístup

- „Adresa souboru“ na „disku“ se skládá z

- posloupnosti adresářů oddělených znakem /
- a jména souboru

obdoba URL webové adresy

- Soubory uživatele jsou zpravidla umístěny v jeho domovském adresáři

Označován zpravidla znakem ~

Příklad výpisu souborů na lokálním úložišti 1/2

Index of file:///

Show hidden objects

Name	Size	Last Modified
.rpmdb		02/09/2013 02:10:41 PM
bin		07/29/2014 11:17:41 AM
boot		09/12/2014 07:30:21 AM
cdrom		05/04/2012 05:49:35 PM
dev		10/08/2014 11:45:55 AM
etc		10/08/2014 11:45:56 AM
home		10/16/2012 03:00:11 PM
initrd.img	13860 KB	09/12/2014 07:30:21 AM
initrd.img.old	13861 KB	09/12/2014 07:26:39 AM
lib		09/12/2014 07:26:51 AM
libnss3.so	1202 KB	09/09/2014 02:30:32 PM
lost+found		05/04/2012 05:46:13 PM
media		07/29/2014 04:07:08 PM
mnt		04/19/2012 11:32:24 AM
opt		09/12/2014 07:22:54 AM
proc		10/08/2014 11:45:30 AM
root		09/12/2014 07:23:13 AM
run		10/08/2014 11:45:57 AM
sbin		09/12/2014 07:30:02 AM
selinux		03/05/2012 02:48:47 PM
srv		04/23/2012 01:34:53 PM
sys		10/08/2014 11:45:32 AM
tmp		10/08/2014 11:59:36 AM

Firefox automatically sends some data to Mozilla so that we can improve your experience. [Choose What I Share](#)

Příklad výpisu souborů na lokálním úložišti 2/2

The screenshot shows a web browser window displaying the index of a file system. The browser address bar shows the file path: `file:///home/jf/pr1/pr1-lab03/`. The index lists the following files:

Name	Size	Last Modified
<code>build.xml</code>	3 KB	09/22/2014 03:49:24 PM
<code>lab.klwd</code>	1 KB	09/22/2014 03:29:21 PM
<code>lib</code>		09/22/2014 03:04:09 PM
<code>maze.klwd</code>		
<code>src</code>		

Below the browser window, a terminal window is open, showing the following commands and their outputs:

```

jf@ubuntu-VirtualBox:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       18G   12G   5.0G   71% /
udev            100M   4.0K  100M   1% /dev
tmpfs           202M   788K   201M   1% /run
none            5.0M   0     5.0M   0% /run/lock
none            1007M 124K   1007M   1% /run/shm

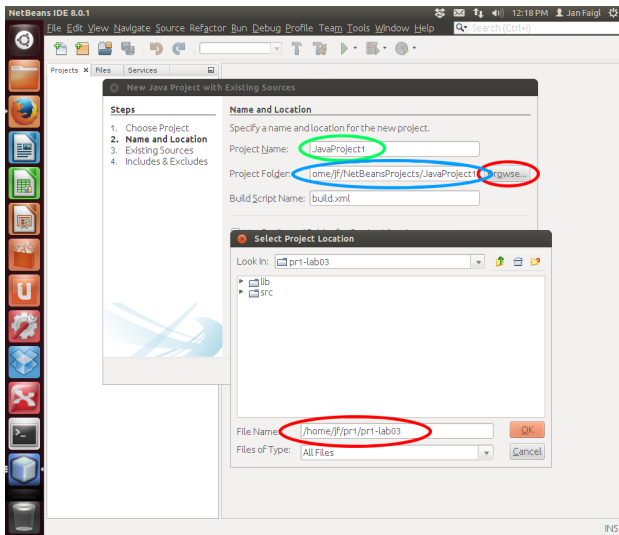
jf@ubuntu-VirtualBox:~$ ls /
bin    dev    lntrd.ing  libnss3.so  mnt  root  selinux  usr  vnlinux
boot  etc    lntrd.ing.old  lost+found  opt  run  srv      var  vnlinux.old
cdrom  home  lib        media       proc /sbin  sys      vnc  var

jf@ubuntu-VirtualBox:~$ ls /home/jf/pr1/
pr1-lab03

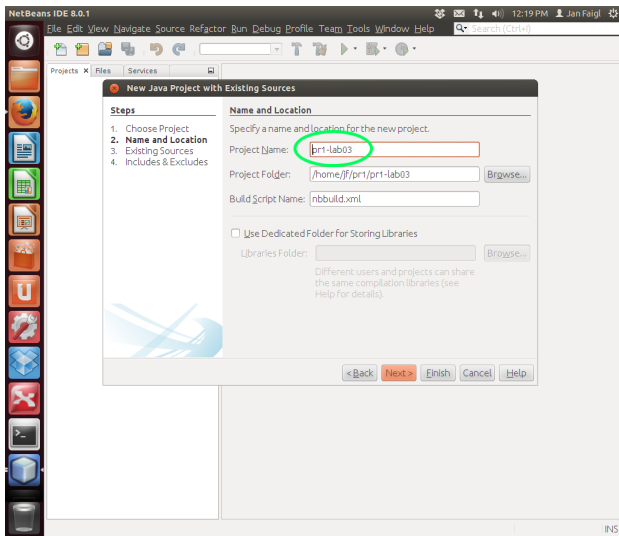
jf@ubuntu-VirtualBox:~$ ls /home/jf/pr1/pr1-lab03/
build.xml  lab.klwd  lib  maze.klwd  src

jf@ubuntu-VirtualBox:~$ ls -l /home/jf/pr1/pr1-lab03/
total 20
-rw-r--r-- 1 jf jf 2423 Sep 22 15:49 build.xml
-rw-rw-r-- 1 jf jf 535 Sep 22 15:29 lab.klwd
drwxrwxr-x 2 jf jf 4096 Sep 22 15:04 lib
-rw-rw-r-- 1 jf jf 923 Oct 6 12:17 maze.klwd
drwxrwxr-x 3 jf jf 4096 Sep 16 10:02 src
jf@ubuntu-VirtualBox:~$
  
```

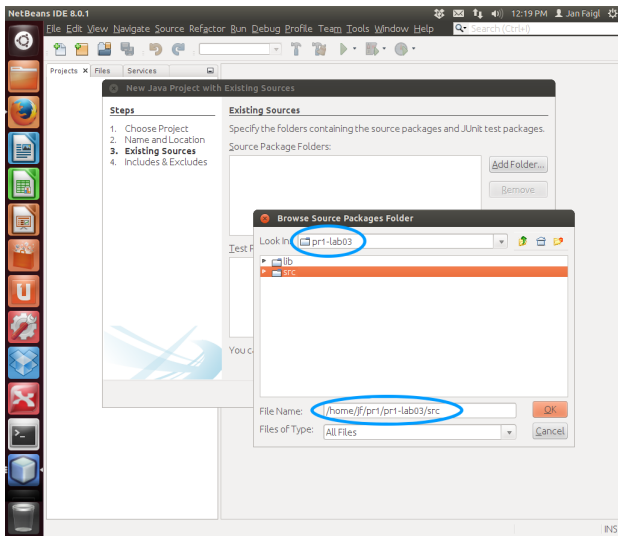
Netbeans – projektový adresář



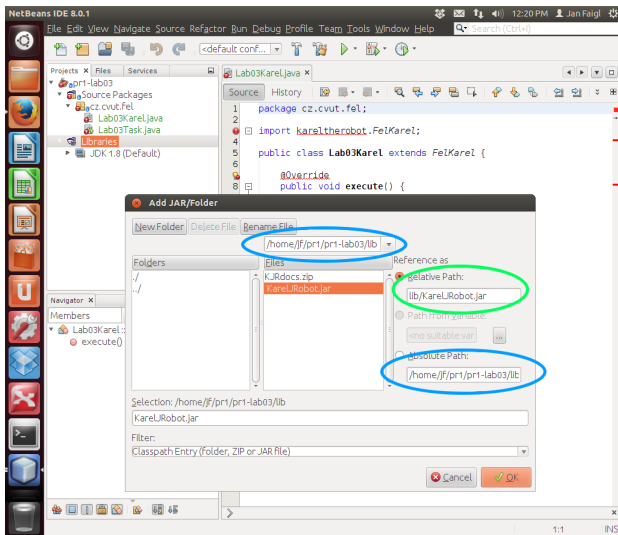
Netbeans – jméno projektu



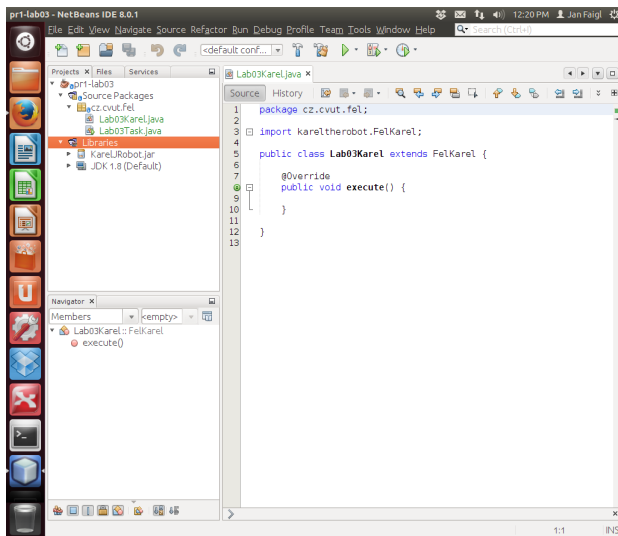
Netbeans – nastavení zdrojových souborů



Netbeans – přidání knihovny .JAR



Netbeans – otevřený projekt



Část II

Přípomínka

Výpočetní problém, algoritmus a program jako jeho řešení

Příklad: Najít největšího společného dělitele čísel 6 a 15.

- Víme co musí platit pro číslo d aby bylo největším společným dělitelem čísel x a y
- Známost **deklarativní znalost** o problému můžeme využít pro návrh výpočetního postupu jak takové číslo najít, např.
 1. Necht' máme nějaký odhad čísla d
 2. Potom můžeme ověřit, zda-li d splňuje požadované vlastnosti
 3. Pokud ano, jsme u cíle
 4. Pokud ne, musíme d vhodně modifikovat a znovu testovat
- Výpočetní problém chceme vyřešit využitím konečné množiny primitivních operací počítače
- Konkrétní úlohu pro čísla 6 a 15 zobecňujeme pro „libovolná“ čísla x a y , pro který navrhne algoritmus
- Algoritmus následně přepíšeme do programu využitím konkrétního programovacího jazyka

Algoritmus a program

- Algoritmus je postup řešení třídy problému
- Algoritmus je recept na výpočetní řešení problému
- Program je implementací algoritmus s vyžitím zápisu příkazů programovacího jazyka
- Program je posloupnost instrukcí počítače

- Předpokládáme, že náš problém lze výpočetně řešit a je výpočetně zvladatelný

Naše problémy na PR1 takové jsou, v praktických problémech tomu však vždycky být nemusí a můžeme narážet problém jak úlohu vůbec formulovat či problém potřebného výpočetního výkonu.

Výpočetní, algoritmické a programové řešení problému

- Množina primitivních instrukcí počítače je relativně malá a zahrnuje následující operace:
 - Práce s číselnými hodnotami (v operační paměti počítače)
Odkazované jmény deklarovaných proměnných
 - Výpočetní operace (výrazy)
Binární nebo unární operace, tj. čtení jedno nebo dvou číselných hodnot z paměti, provedení operace a zápis výsledku do operační paměti.
 - Testování hodnot proměnných (podmínky a větvení výpočtu)
Pokud podmínka platí, vykonej instrukci, jinak udělej něco jiného nebo nedělej nic.
 - Skoky na provedení konkrétní posloupnosti instrukcí v závislosti na splnění podmínky
„Program Counter“ (PC) jako ukazatel z jaké adresy v paměti čte počítač instrukce pro vykonání
- Tyto instrukce se objevují ve své abstraktní podobě
 - v zápisu algoritmu např. jako bloky vývojového diagramu
 - v zápisu programu jako příkazy a vyhrazená klíčová slova

Slovní popis způsobu výpočtu

■ Úloha:

Najděte největšího společného dělitele čísel 6 a 15.

Co platí pro společného dělitele čísel?

■ Řešení

Návrh postupu řešení pro dvě libovolná přirozená čísla

*Definice **vstupu** a **výstupu** algoritmu*

- Označme čísla x a y
 - Vyberme menší z nich a označme jej d
 - Je-li d společným dělitelem x a y končíme
 - Není-li d společným dělitelem pak zmenšíme d o 1 a opakujeme test až d bude společným dělitelem x a y
-
- Symboly x , y a d reprezentují **proměnné** (paměťové místo), ve kterých jsou uloženy hodnoty, které se v průběhu výpočtu mohou měnit.

Slovní popis činnosti algoritmu

■ Úloha:

Najít největší společný dělitel přirozených čísel x a y .

■ Popis řešení

■ **Vstup:** dvě přirozená čísla x a y

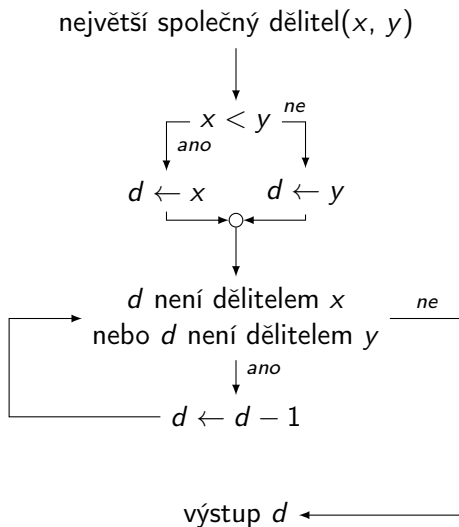
■ **Výstup:** přirozené číslo d – největší společný dělitel x a y

■ Postup

1. Je-li $x < y$, pak d nastav na hodnotu x , jinak na hodnotu y
2. Pokud d není dělitelem x nebo d není dělitelem y opakuj krok 3, jinak proved' krok 4
3. Zmenši d o 1
4. Výsledkem je hodnota d

Algoritmus = výpočetní postup jak zpracovat vstupní data a určit (vypočítat) požadované výstupní hodnoty (data) s využitím elementárních výpočetních instrukcí a pomocných dat.

Postup výpočtu algoritmu vyjádřený formou vývojového diagramu



Zápis algoritmu v pseudojazyku

- Zápis algoritmu využitím klíčových a dobře pochopitelných slov

Algoritmus 1: Nalezení největšího společného dělitele

Vstup: x, y – kladná přirozená čísla

Výstup: d – největší společný dělitel x a y

if $x < y$ **then**

$d \leftarrow x$;

else

$d \leftarrow y$;

while d není dělitelem x nebo d není dělitelem y **do**

$d \leftarrow d - 1$;

return d

Neodpovídá přesně zápisu programu v konkrétním programovacím jazyku, ale je čitelný a lze velmi snadno přepsat.

Zápis algoritmu v programovacím jazyku Java

```
1  int getGreatestCommonDivisor(int x, int y) {
2      int d;
3      if (x < y) {
4          d = x;
5      } else {
6          d = y;
7      }
8      while ( (x % d != 0) || (y % d != 0)) {
9          d = d - 1;
10     }
11     return d;
12 }
```

- Nebo také s využitím **ternárního operátoru**
podmínka ? výraz : výraz

```
1  int getGreatestCommonDivisor(int x, int y) {
2      int d = x < y ? x : y;
3      while ( (x % d != 0) || (y % d != 0)) {
4          d = d - 1;
5      }
6      return d;
7  }
```

lec03/DemoGCD.java

Část III

Znak, textové řetězce a vstupu

Text, znaková sada a kódování

Text zapisujeme jako posloupnost znaků, ale jak jsou znaky uloženy v paměti počítače?

■ Znaková sada - je množina dvojic znak–číslo

Předepíše znaku číslo

■ ASCII – 7 bitů, znaky s čísly 0–127

■ Unicode – 16 bitů

<http://unicode.org/charts>

■ ISO-8859-2 – 8 bitů

CP-1250, CP852, Bratři Kamenických

■ ISO 10 646 – 32 bitů

■ Kódování znakové sady – předepíše, jak jsou kódy znaků převedeny na posloupnost bajtů

■ UCS-2 – jeden znak 2 bajty

UCS – Universal Character Set

■ UCS-4 – jeden znak 4 bajty

■ UTF-8 – ASCII znaky 1 bajt, ostatní znaky jsou kódovány 2 až 6 bajty

České znaky jsou kódovány 2 bajty

Typ `char` – reprezentace znaků v Javě

- Kódování UTF-16 s variabilní délkou znaku
- Umožňuje kódovat 1 112 064 znaků znakové sady Unicode

Pro rozsah 0-0xFFFF stejný výsledek jako UCS-2

- Programy v Javě provádí konverzi znaků mezi vnitřní reprezentací v programech a operačním systémem automaticky (většinou) podle lokálního nastavení LOCALE

<http://www.oracle.com/technetwork/articles/javase/supplementary-142654.html>

Příklad

```

1 import java.util.Date;
2
3 Date date = new Date();
4 System.out.printf("Double number: %4.2f%n", Math.PI);
5 System.out.printf("Today's date : %tF%n", date);
6 System.out.printf("Current time : %1$str that is %1$tT%n", date);

```

```

LANG='' java DemoLocale
Double number: 3.14
Today's date : 2014-10-05
Current time : 01:51:46 PM that
is 13:51:46

```

```

LANG=cs_CZ.UTF-8 java DemoLocale
Double number: 3,14
Today's date : 2014-10-05
Current time : 01:52:22 ODP.
that is 13:52:22

```

`lec03/DemoLocale.java`

Text v programu, literál typu **String**

- Text v programu se zapisuje do literálu typu **String**

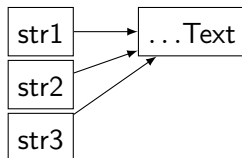
Není to primitivní typ

- Text se zapisuje jako posloupnost znaků do dvojice (dvojitých) uvozovek "text"
- Operátor `+` je spojení řetězců (konkatenace)
Výsledkem `"abc" + "123"` je řetězec `"abc123"`
- Je-li jeden operand operátoru `+` typu **String** a druhý jiného typu, převede se druhý na typ **String** a výsledkem je konkatenace řetězců
 - `"abc" + 5` výsledek je `"abc5"`
 - `"a" + 1 + 2` výsledek je `"a12"`
 - `"a" + 1 + (-2)` výsledek je `"a1-2"`
- Proměnné "typu" řetězec jsou instance knihovnické třídy `java.lang.String`

Třída `String`

- Objekt typu `String` lze vytvořit literálem textového řetězce
- Hodnotu objektu typu `String` nelze jakkoli změnit
- Operace zřetězení je realizována jak metodou `concat` tak *přetíženým* operátorem `+`
- Příklad referenčních proměnných typu `String`

```
String str1 = "Text";  
String str2 = str1;  
String str3 = "Text";
```



Java si eviduje při překladu seznam všech vytvořených řetězců a pokud již stejný existuje, nevytváří jeho kopii.

Operace s řetězci

- Spojení řetězců operátorem +
- Porovnání řetězců
 - Relační operátory == a != porovnávají *reference*, nikoliv obsah řetězců
 - Pro porovnání řetězců slouží metody equals nebo compareTo
- Hodnotu referenční proměnné typu **String** lze změnit, odkazuje pak ale na jiný řetězec), vlastní řetězec změnit nelze!

Příklad

```
String s1 = "abcd";  
String s2 = "ab";  
String s3 = s2 + "cd";
```

```
System.out.println(s1 == s3); // vypise false  
System.out.println(s1.equals(s3)); // vypise true
```

Porovnání řetězců

Příklad

```
String str = "string";
System.out.println(str.equals("string")); // true
System.out.println(str.equals("string two")); // false

String str1 = "aa-string";
String str2 = "bb-string";
String str3 = "cc-string";

final int comp21 = str2.compareTo(str1);
final int comp23 = str2.compareTo(str3);
final int comp22 = str2.compareTo(str2);

System.out.println("Comparison str2 vs str1: " + comp21); // 1
System.out.println("Comparison str2 vs str3: " + comp23); // -1
System.out.println("Comparison str2 vs str2: " + comp22); // 0

String concat = "aa-" + str;
String str1b = str1;

System.out.printf("%s\n%s\n%s\n", str1, concat, str1b);
// vypise tri radky s aa-string
System.out.println((str1 == concat)); // false
System.out.println((str1 == str1b)); // true
```

lec03/DemoString.java

Standardní vstup programu

- Vstup programu je možné znakově orientovanému programu předat prostřednictvím **standardního vstupu**
- V Javě je standardní vstup přístupný prostřednictvím knihovny `System` jako *proud* **`System.in`**
- Z proudu lze načítat jednotlivé znaky zadané uživatelem (z klávesnice)

Podobně jako `System.out` i standardní vstup můžeme přesměrovat v příkazovém interpretu.

- Pro přístup a načítání zadaného vstupu jako hodnot základních typů můžeme použít třídu **`Scanner`** ze standardní knihovny
- Třída `Scanner` poskytuje metody pro zpracování znaků a načítání slov (tokenů), které jsou odděleny mezerou nebo dalšími bílými znaky

Znaky, které jsou standardně zobrazeny jako mezery.

Třída **Scanner** – načítání vstupu

- Použití třídy **Scanner** je třeba deklarovat příkazem **import java.util.Scanner;**
- Dále je nutné vytvořit objekt třídy **Scanner** a napojit jej na standardní vstup (**System.in**), např.
Scanner sc = new Scanner(System.in);
- **Scanner** postupně načítá *token*, tj. posloupnost znaků oddělenou tzv. bílými mezerami („*whitespaces*“)
mezera, konec řádku, tabelátor (IFS – „Internal Field Separator“)
- Základní služby třídy **Scanner** jsou:
 - **sc.nextInt()** – přečte celé číslo, vrací hodnotu typu **int**
 - **sc.nextDouble()** – přečte číslo, vrací hodnotu typu **double**
Oddělovač desetinné části v závislosti na nastavení OS
Local.setDefault(Locale.ENGLISH);
 - **sc.nextLine()** – vrací posloupnost znaků do konce řádku a vrací jako hodnotu typu **String**

Standardní vstup a národní nastavení

Příklad načítání čísel

```

1 import java.util.Scanner;
2
3 public static void main(String[] args) {
4     Scanner sc = new Scanner(System.in);
5     double x;
6     double y;
7
8     System.out.print("Enter coordinates x and y: ");
9     x = sc.nextDouble(); // nacte token reprez. cislo typu double
10    y = sc.nextDouble(); // nacte token reprez. cislo typu double
11
12    System.out.println("Coordinates are (" + x + " " + y + ")");
13    System.out.printf("Coordinates are (%.3f %.3f)%n", x, y);
14 }

```

<pre> export LANG="" java DemoScannerDouble Enter coordinates x and y: 12.34 56.78 Coordinates are (12.34 56.78) Coordinates are (12.340 56.780) </pre>	<pre> export LANG=cs_CZ.UTF-8 java DemoScannerDouble Enter coordinates x and y: 12,34 56,78 Coordinates are (12.34 56.78) Coordinates are (12,340 56,780) </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------

Vyzkoušejte chování programu s oddělovačem . a , na vstupu.

lec03/DemoScannerDouble.java

Příklad načítání vstupu 1/2

- Zadání: výpočet průměrné hodnoty posloupnosti celých čísel
- Postup řešení:
 - Postupně načítáme tokeny (celá čísla)
 - Inkrementujeme počet načtených hodnot
 - Zadaná čísla sčítáme
 - Při detekci konce vstupu vypočítáme průměr

```
1 Scanner sc = new Scanner(System.in);
2 int count = 0;
3 int sum = 0;
4 System.err.println("Enter a sequence of interger numbers:");
5 while(sc.hasNext()) {
6     sum += sc.nextInt();
7     count += 1;
8 }
9 System.out.println("Average of the " + count + " input numbers
    is " + (sum / count));
```

- Konec vstup je detekován zadáním znaku **Ctrl+D** (EOT – *End-of-transmission*)

`lec03/DemoScannerAvgSimple.java`

Příklad načítání vstupu 2/2

- Program doplníme o explicitní konverzi výsledku na typ double
- Dále program doplníme o detekci nulového vstupu

```
1 if (count > 0) {
2     System.out.println("Average of the " + count + " input
3         numbers is " + ((double)sum / count));
4 } else {
5     System.err.println("At least one number must be given");
6 }
```

lec03/DemoScannerAvg.java

- Vstup můžeme místo z klávesnice zadat programu ze souboru přesměrování standardní vstupu programu

```
1 for i in $( seq 1 10 ); do echo $[ ( $RANDOM % 20 ) +1 ]; done
2 > numbers.txt
3 java DemoScannerAvg < numbers.txt
4 Enter a sequence of interger numbers:
5 Average of the 10 input numbers is 8.5
```

- Přesměrování standardního chybového výstupu vytiskne na obrazovku pouze výsledek

```
1 java DemoScannerAvg < numbers.txt 2>err
2 Average of the 10 input numbers is 8.5
```

Část IV

Řídicí struktury

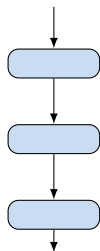
Přehled řídicích struktur

- Řídicí struktury mají obvykle formu strukturovaných příkazů
 - **Složený příkaz** – posloupnost příkazů
 - **Blok** – posloupnost deklarácí a příkazů vymezena složenými závorkami { a }
- Základní řídicí struktury
 - **Posloupnost** – předepisuje **postupné provedení** dílčích příkazů
 - **Větvení** – předepisuje provedení dílčích příkazů v závislosti na **splnění určité podmínky**
 - **Cyklus** – předepisuje **opakované provedení** dílčích příkazů v závislosti na splnění určité podmínky

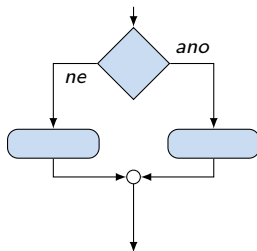
Připomínka

Typy řídicích struktur 1/2

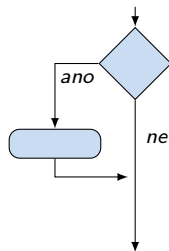
■ Sekvence



■ Podmínka If



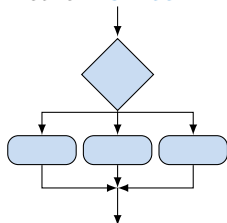
■ Podmínka If



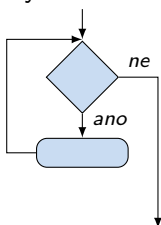
Přípomínka

Typy řídicích struktur 2/2

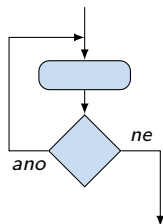
■ Větvení **switch**



■ Cyklus **for** a **while**



■ Cyklus **do**



Připomínka

Větvení `if`

- Příkaz `if` umožňuje větvení programu na základě podmínky
- Má dva základní tvary
 - `if (podmínka) příkaz1`
 - `if (podmínka) příkaz1 else příkaz2`
- podmínka je logický výraz, jehož hodnota je typu `boolean`
- příkaz je příkaz, složený příkaz nebo blok
- **Jaký je doporučený způsob zápisu příkazů?**

Příklad zápisu

Připomínka

```
1  if (x < y) {
2      int tmp = x;
3      x = y;
4      y = tmp;
5  }
```

```
1  if (x < y) {
2      min = x;
3      max = y;
4  } else {
5      min = y;
6      max = x;
7  }
```

Jaký je smysl těchto programů?

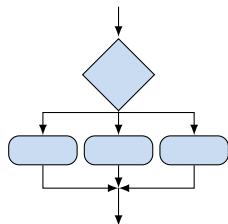
Příkaz větvení **switch**

- Příkaz **switch** (přepínač) umožňuje větvení programu do více větví na základě různých hodnot výrazu výčtového typu, jako jsou např. **int**, **byte**, **char**, ale také výrazy typu **String**
- Základní tvar příkazu

```
switch (výraz) {  
    case konstanta1: příkazy1; break;  
    case konstanta2: příkazy2; break;  
    ...  
    case konstantan: příkazyn; break;  
    default: příkazydef; break;  
}
```

kde *konstanty* jsou téhož typu jako *výraz* a *příkazy_i* jsou posloupnosti příkazů

Sémantika: vypočte se hodnota výrazu a provedou se ty příkazy, které jsou označeny konstantou s identickou hodnotou. Není-li vybrána žádná větev, provedou se příkazy_{def} (jso-li uvedeny).



Větvení `switch` – příklad 1/2

- Napište konverzní program, který podle čísla dnu v týdnu vytiskne na obrazovku jméno dne. Ošetřete případ, kdy bude zadané číslo mimo platný rozsah (1 až 7).

Příklad implementace

```
int dayOfWeek = 3;
if (dayOfWeek == 1) {
    System.out.println("
    Monday");
} else if (dayOfWeek == 2) {
    System.out.println("
    Tuesday");
} else ... {
} else if (dayOfWeek == 7) {
    System.out.println("
    Sunday");
} else {
    System.err.println("
    Invalid week");
}
```

```
int dayOfWeek = 3;
switch (dayOfWeek) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    ...
    case 7:
        System.out.println("Sunday");
        break;
    default:
        System.err.println("Invalid
        week");
        break;
}
```

DemoSwitchDayOfWeek.java

Oba způsoby jsou sice funkční, nicméně elegantněji lze vyřešit úlohu použitím datové struktury pole nebo ještě lépe `java.util.HashMap`.

Větvení `switch` – příklad 2/2

- Napište konverzní program pro převod textového řetězce obsahující kalendářní měsíc na číslo měsíce.

- Větvení příkazem `switch` lze také pro textový řetězec
- Hodnotu můžeme předat programu jako argument
- Text může obsahovat malá a velká písmena
- Měsíc lze identifikovat podle počátečních třech písmen
 - Přizpůsobujeme tím ale trošku zadání

```

public static void main(String args[]) {
    String monthStr =
        (args.length > 0) ? args[0] : "Jun";

    int month = -1;
    monthStr = monthStr.substring(0, 3);
    switch (monthStr.toLowerCase()) {
        case "jan":
            month = 1;
            break;
        ...
    }
    if (month >= 0 && month <= 12) {
        System.out.println("Parsed month '" +
            monthStr + "' is " + month + " month of
            the year");
    } else {
        System.err.println("Cannot parse '" +
            monthStr + "'");
    }
}

```

DemoSwitchMonth.java

Větvení **switch** – pokračování ve vykonávání dalších větví

- Příkaz **break** dynamicky ukončuje větev, pokud jej neuvedeme, pokračuje se v provádění další větve

Příklad volání více větví

```
int part = ?
switch(part) {
    case 1:
        System.out.println("Branch 1");
        break;
    case 2:
        System.out.println("Branch 2");
    case 3:
        System.out.println("Branch 3");
        break;
    case 4:
        System.out.println("Branch 4");
        break;
    default:
        System.out.println("Default branch");
        break;
}
```

- part ← 1
Branch 1
- part ← 2
Branch 2
Branch 3
- part ← 3
Branch 3
- part ← 4
Branch 4
- part ← 5
Default branch

DemoSwitchBreak.java

Cyklus **while** a **do-while**

- Základní příkaz cyklu **while** má tvar **while** (*podmínka*) příkaz
- Základní příkaz cyklu **do-while** má tvar **do** příkaz **while** (*podmínka*)

Příklad

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

```
q = x;
do {
    q = q - y;
} while (q >= y);
```

- Jaká je hodnota proměnné q po skončení cyklu pro hodnoty
 - $x \leftarrow 10$ a $y \leftarrow 3$
 - $x \leftarrow 2$ a $y \leftarrow 3$

while: 1, do-while: 1

while: 2, do-while: -1

DemoWhile.java

Cyklus **for**

- Základní příkaz cyklu **for** má tvar
for (*inicializace*; *podmínka*; *změna*) příkaz
- Odpovídá cyklu **while** ve tvaru:
inicializace;
while (*podmínka*) {
 příkaz;
 změna;
}
- Změnu řídicí proměnné lze zkráceně zapsat operátorem inkrementace nebo dekrementace **++** a **--**
- Alternativně lze též použít zkrácený zápis přiřazení, např. **+=**

Příklad

```
for (int i = 0; i < 10; ++i) {  
    System.out.println("i: " + i);  
}
```

Cyklus **for** – příklady

- Jak se změní výstup když použijeme místo prefixového zápisu `++i` postfixový zápis `i++`

```
for (int i = 0; i < 10; i++) {  
    System.out.println("i: " + i);  
}
```

- V cyklu můžeme také řídicí proměnou dekrementovat

```
for (int i = 10; i >= 0; --i) {  
    System.out.println("i: " + i);  
}
```

Kolik program vypíše řádků?

- A kolik řádků vypíše program:

```
for (int i = 10; i > 0; --i) {  
    System.out.println("i: " + i);  
}
```

- Řídicí proměnná může být také například typu **double**

```
for (double d = 0.5; d < Math.PI; d += 0.1) {  
    System.out.println("d: " + d);  
}
```

Shrnutí přednášky

Diskutovaná témata

- Znak, řetězec a standardní vstup programu
- Řídicí stuktury – větvení **if** a **switch**, cykly **while**, **do-while** a **for**
- Příště: Reprezentace základních typů, pole, funkce a procedury