Společná část - otázka č. 5

Imperativní programování, software, překladač, interpret, vnitřní forma, programovací jazyky, syntaxe, sémantika, proměnné, výrazy, vstup, výstup, řídící struktury, jednoduché datové typy, přiřazení, funkce, procedury, parametry, rozklad problému na podproblémy, princip rekurze a iterace (A0B36PR1)

June 2, 2012

1 Základní pojmy

1.1 Informatika

Informatika:
- zabývá se zpracováním informací nejen na počítačích
- studuje výpočetní a informační procesy z hlediska hardware i software – „technická informatika“
- je součástí teorie informací, věda spojující aplikovanou matematiku a elektrotechniku za účelem kvantitativního vyjádření informace

1.2 Software

V informatice sada všech počítačových programů v počítači. Software zahrnuje:
– operační systém (zajišťuje běh programů)
– aplikační software (pracuje s ním uživatel)
– další (knihovny, middleware, BIOS, firmware apod.)

1.2.1 Knihovna

Knihovna je v informatice označení pro soubor funkcí a procedur (v objektovém programování též objektů, datových typů a zdrojů), který může být sdílen více počítačovými programy. Knihovna usnadňuje programátorovi tvorbu zdrojového kódu tím, že umožňuje použít již vytvořený kód i v jiných programech. Knihovna navenek poskytuje své služby pomocí API (aplikační rozhraní), což jsou názvy funkcí (včetně popisu jejich činnosti), předávané parametry a návratové hodnoty. Knihovny lze rozdělit podle vazby na program, který je používá, na statické a dynamické.

1.2.1.1 Statické knihovny

Statická knihovna tvoří s přeloženým programem kompaktní celek. V závěrečné fázi překladu programu ze zdrojového kódu do strojového kódu jsou volané funkce (a další jimi kaskádově volané funkce) připojovány linkerem (odtud označení linkování) do výsledného spustitelného souboru. V horším případě je k programu připojena celá knihovna bez ohledu na to, jaké části program skutečně využívá. Výsledný spustitelný soubor tak v sobě obsahuje všechny části, které jsou nutné pro jeho běh. Staticky slinkovaný spustitelný soubor proto typicky při spuštění nepotřebuje žádné další soubory, takže je ho možné překopírovat na jiný systém a jednoduše spustit (tj. bez instalace, tak, jak je známá například pro běžné programy v Microsoft Windows). Př. .lib, .a.

1.2.1.2 Dynamické knihovny

Dynamické knihovny nejsou (na rozdíl od statických knihoven) k výslednému spustitelnému souboru přidávány. V závěrečné fázi překladu programu ze zdrojového kódu do strojového kódu jsou odkazy na volané knihovní funkce pomocí linkeru (odtud označení linkování) zapsány do speciální tabulky symbolů, která je připojena k výslednému spustitelného souboru. Pro chod programu (tj. spuštění výsledného spustitelného souboru) je pak nutné mít k dispozici též příslušné dynamické knihovny. Pokud dynamická knihovna využívá ke své činnosti jiné dynamické knihovny, vzniká řetězec závislostí a všechny potřebné knihovny musí být při spuštění programu přítomny. Př. .dll, .o, .so.

1.3 Hardware

Zahrnuje všechny fyzické součásti počítače

- čistě elektronická zařízení (procesor, paměť, display)
- elektromechanické díly (klávesnice, tiskárna, diskety, disky, jednotky CD-ROM, páskové jednotky, reproduktory) pro vstup, výstup a ukládání dat.
- Počítač se skládá z procesoru, operační paměti a vstupně-výstupních zařízení.

1.4 Algoritmus

Postup při řešení určité třídy úloh, který je tvořen seznamem jednoznačně definovaných příkazů a zaručuje, že pro každou přípustnou kombinaci vstupních dat se po provedení konečného počtu kroků dospěje k požadovaným výsledkům.

- **hromadnost** - měnitelná vstupní data
- **determinovanost** - každý krok je jednoznačně definován
- **konečnost a resultativnost** - pro přípustná vstupní data se po provedení konečného počtu kroků dojde k požadovaným výsledkům

1.5 Program

Program je předpis (zápis algoritmu) pro provedení určitých akcí počítačem zapsaný v programovacím jazyku.

1.6 Programovací jazyk

Programovací jazyk je prostředek pro zápis algoritmů, jež mohou být provedeny na počítači.

1. **strojově orientované**

- **strojový jazyk** = jazyk fyzického procesoru

- **asembler** = jazyk symbolických adres

1. **vyšší jazyky**

- **imperativní** (příkazové, procedurální)
- **neimperativní** (např. funkcionální)

1.6.1 Imperativní programování

Hlavní rysy imperativních jazyků (např. C, C++, Java, Pascal, Basic, ...)
- zpracovávané údaje mají formu datových objektů různých typů, které jsou v programu reprezentovány pomocí proměnných resp. konstant
- program obsahuje deklarace a příkazy
- deklarace definují význam jmen (identifikátorů)
- příkazy předepisují akce s datovými objekty nebo způsob řízení výpočtu

1.7 Syntaxe

Souhrn pravidel udávajících přípustné tvary dílčích konstrukcí a celého programu.

1.8 Sémantika

Udává význam jednotlivých konstrukcí.

1.9 Interpret

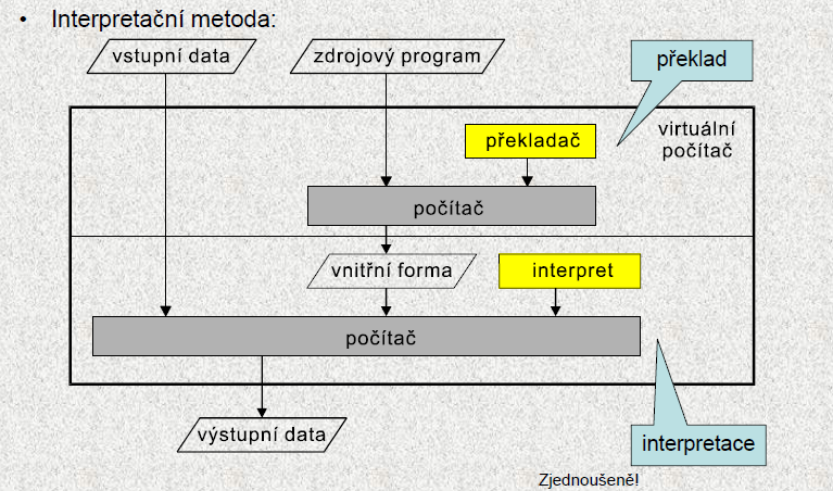
Interpret je v informatice speciální počítačový program, který umožňuje **přímo vykonávat (interpretovat)** zápis jiného programu v jeho zdrojovém kódu ve zvoleném programovacím jazyce. Program proto není nutné převádět do strojového kódu cílového procesoru, jako je tomu v případě překladače. Interpret tak umožňuje programování kódu, který je snadno přenositelný mezi různými počítačovými platformami.

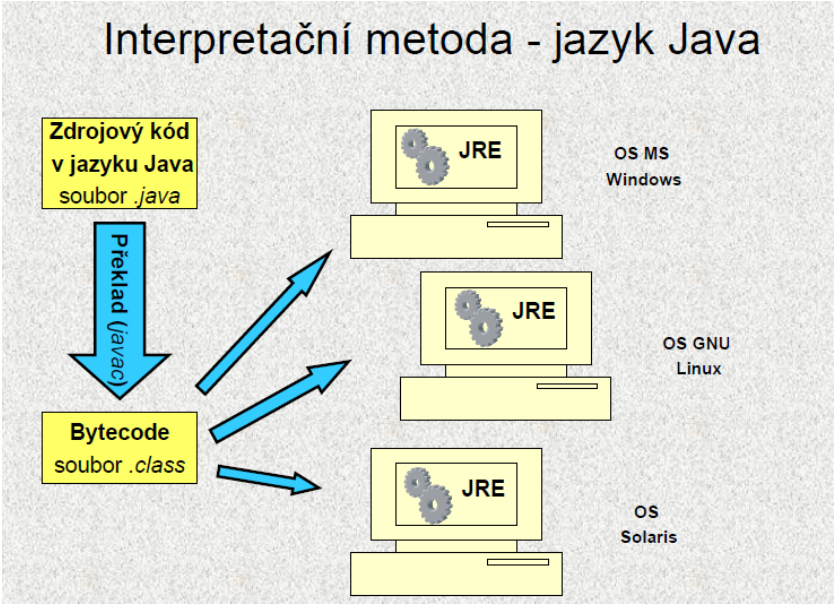
1. provádějí přímo zdrojový kód (unixový shell, COMMAND.COM nebo interprety jazyka BASIC)

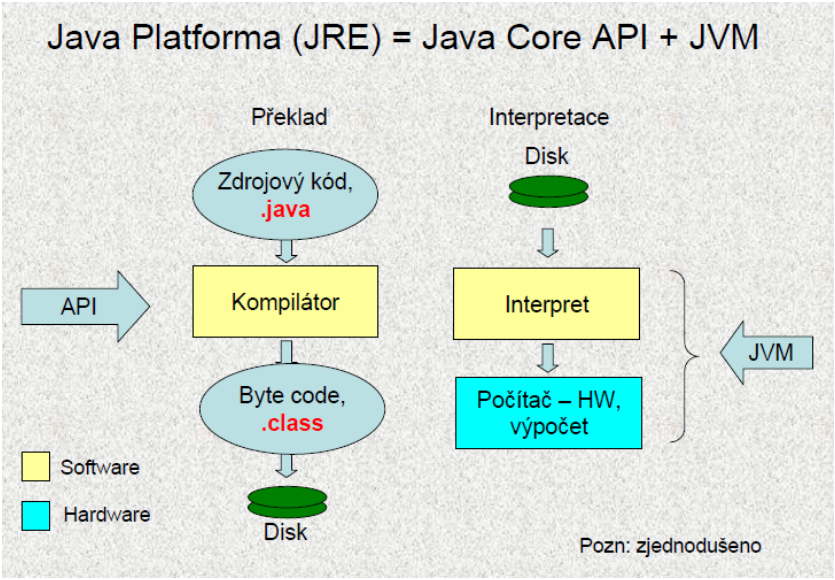
2. přeloží zdrojový kód do efektivnějšího mezikódu, který následně spustí (Perl, Python nebo MATLAB)

3. přímo spustí předem vytvořený předkompilovaný mezikód, který je produktem části interpretu (UCSD Pascal a Java - zdrojové kódy jsou kompilovány předem, uloženy ve strojově nezávislém tvaru, který je po spuštění linkován a interpretován nebo kompilován v případě použití JIT).

- Interpretační metoda:





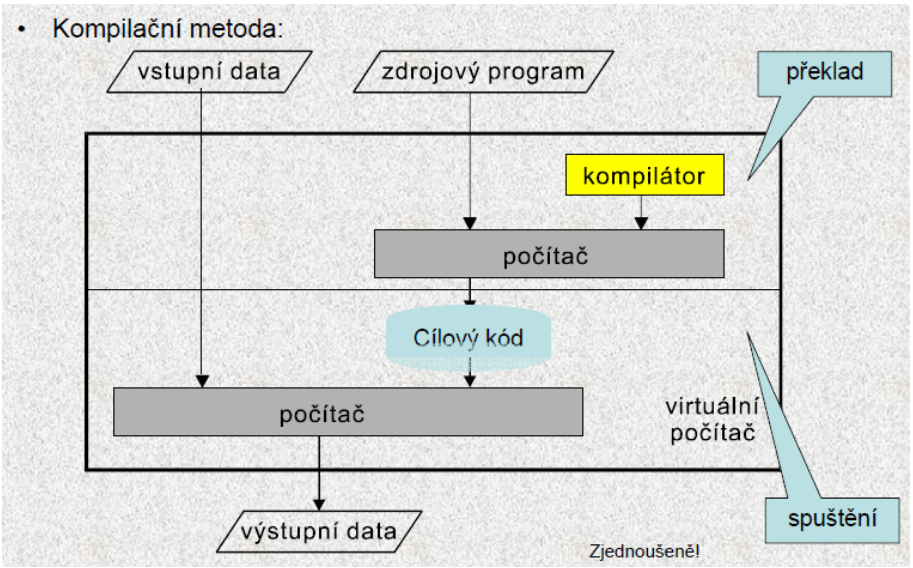


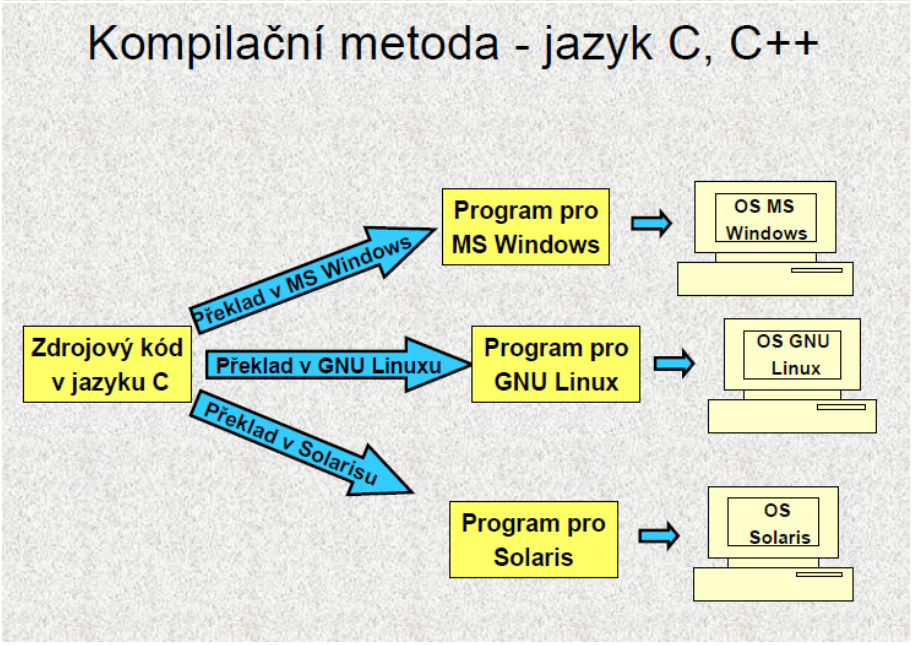
1.9.1 Vnitřní forma

Při interpretaci se zdrojový program přeloží do vnitřní formy. Ta není strojovým jazykem fyzického procesoru, ale je jazykem virtuálního počítače. Provedení programu ve vnitřní formě na konkrétním počítači zajistí interpretr.

1.10 Kompilátor = překladač

Překladač (též kompilátor) je v nejčastějším smyslu slova softwarový nástroj používaný programátory pro vývoj softwaru. Kompilátor slouží pro překlad algoritmů zapsaných ve vyšším programovacím jazyce do jazyka strojového, či spíše do strojového kódu. Z širšího obecného hlediska je kompilátor stroj, respektive program, provádějící překlad z nějakého vstupního jazyka do jazyka výstupního.


• Kompilační metoda:
vstupní data
zdrojový program
překlad
kompilátor
počítač
Cílový kód
počítač
virtuální počítač
výstupní data
spuštění
Zjednodušeně!


Kompilační metoda - jazyk C, C++
Zdrojový kód v jazyku C
Překlad v MS Windows
Překlad v GNU Linuxu
Překlad v Solarisu
Program pro MS Windows
Program pro GNU Linux
Program pro Solaris
OS MS Windows
OS GNU Linux
OS Solaris

2 Části programovacích jazyků

2.1 Proměnná

V imperativním programování je proměnná „úložiště“ informace (tedy vyhrazené místo v paměti - v některých jazycích se ovšem v průběhu výpočtu může místo, kde je proměnná uložena, měnit). Proměnná nebo (v beztypových jazycích) její hodnota má typ.

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

2.1.1 Přiřazení

Pro přiřazení hodnoty proměnné slouží přikazovací příkaz. Má tvar <proměnná> = <výraz>;

V jazyku Java zavedeme proměnnou deklarací int promenna = 0;

Přiřazovací příkaz: promenna = 37;

Konstantě nelze přiřadit hodnota: MAX = 32 -> CHYBA!!

Jazyk JAVA má **silnou typovou kontrolu**, následující nelze: boolean b; b = 1; -> CHYBA!!

2.1.1.1 Přiřazovací operátory

- <proměnná> = <proměnná> <OP> <výraz> e.g. x = x +1
- <proměnná> <OP>= <výraz> x += 1

2.1.2 Přidělení paměti proměnným

Přidělením paměti proměnné rozumíme určení adresy umístění proměnné v paměti počítače.

2.1.3 Typy proměnných

- lokální proměnné funkcí - pamět přidělena při volání funkce, po jejím skončení uvolněna, pamět na zásobníku
- statické proměnné tříd - pamět přidělena při zavedení kódu třídy do paměti (JVM), až do konce programu

2.2 Primitivní datové typy

Datový typ (zkráceně jen typ) specifikuje:

• **množinu hodnot** - pro int: -2147483648 .. 2147483647

• **množinu operací,** které lze s hodnotami daného typu provádět (pro int je to například: relační operace ==, !=, >, >=, <, <=, jejichž výsledkem je hodnota typu boolean)

V jazyce JAVA je třeba deklarovat, s jakým typem dat budeme pracovat. Překladač tuto deklaraci hlídá. V důsledku reprezentace dat v počítači vznikají nepřesnosti (1.00 + 2.00 se nemusí rovnat 3.00). Čísla jsou aproximována. Čím větší exponent (tzn. čím dále od nuly), tím je větší mezera mezi aproximacemi. Kolem nuly je čísel nejvíce. Reprezentace dat v počítači (int, double apod.) bude podrobněji rozebrána pravděpodobně v nějaké otázce z APO.

2.2.1 JAVA - Primitivní datové typy

1. **byte:** Datový typ byte obsahuje 8bitové dvakrát doplněné celé číslo se znaménkem. Jeho minimální hodnota je −128 a maximální 127. Datový typ byte může být použit pro ušetření paměti ve velkých polích, kde se může ve speciálních případech hodit každý bit. Taktéž může být náhradou za typ int, kde mohou jeho limity pomoct objasnit váš kód; takto je hodnota limitována víc než údajem v dokumentaci.

2. **short:** Datový typ short je 16bitové dvakrát doplněné celé číslo. Jeho minimální hodnota je −32 768 a maximální 32 767. Stejně jako u typu byte jsou zde stejné případy použití: typ short můžete použít k ušetření paměti při vytváření velkých polí u náročných aplikací.

3. **int:** Datový typ int je 32bitové dvakrát doplněné celé číslo. Jeho minimální hodnota je −2 147 483 648 a maximální 2 147 483 647 (2^31). Pro celočíselné hodnoty je tento typ tou základní volbou, pokud se nedostanete do výše popsaných situací. Tento datový typ bude dostatečně velký pro většinu čísel ve vaší aplikaci, pokud však budete potřebovat širší rozsah hodnot, použijte místo něj typ long.

4. **long:** Datový typ long je 64bitové dvakrát doplněné celé číslo. Jeho minimální hodnota je −9 223 372 036 854 775 808 a maximální 9 223 372 036 854 775 807 (2^63). Používejte tento datový typ, pokud vám nebude stačit rozsah hodnot poskytovaný datovým typem int.

5. **float:** Datový typ floatje 32bitové IEEE 754 číslo s pohyblivou desetinou čárkou. Stejně jako u typu byte a short používejte typ float (místo double), pokud potřebujete ušetřit paměť ve velkých polích. Tento datový typ byste nikdy neměli používat pro přesné hodnoty, jako jsou finanční částky. Pro tento případ budete muset použít třídu java.math.BigDecimal.

6. **double:** Datový typ double je 64bitové IEEE 754 číslo s pohyblivou desetinou čárkou. Pro desetinná čísla je tento datový typ tou základní volbou. Stejně jako typ float se double nehodí pro přesné hodnoty, jako jsou finanční částky.

7. **boolean:** Datový typ boolean má povoleny pouze 2 hodnoty: true a false. Používejte tento datový typ, abyste specifikovali, zda je podmínka pravdivá (true) nebo nepravdivá (false). Tento datový typ obsahuje 1 bit informací, ale jeho velikost není přesně definována.

8. **char:** Datový typ char je jednoduchý 16bitový Unicode znak (proto jsou odstraněny problémy s různými jazyky). Jeho minimálni hodnou je u0000 (nebo 0) a maximální uffff (nebo 65535). Konverze znaků probíhá v JAVA většinou automaticky podle nastavení jazyka OS.

Primitivní či základní datové typy

Typ	Bitů	Rozsah		Obal.třída
Celočíselný typ				
byte	8	-128 ... 127		Byte
short	16	-32768 ... 32767		Short
int	32	-2147483648 ... 2147483647		Integer
long	64	-9223372036854775808 ... 9223372036854775807		Long
Reálný typ, IEEE 754 (NaN, infinity)				
float	32	2^{-149} ... $(2-2^{-23})*2^{127}$		Float
double	64	2^{-1074} ... $(2-2^{-52})*2^{1023}$		Double
Znaky, UCS2				
char	16	'\u0000' to '\uffff' 0 ... 65535		Character
Logický typ				
boolean	1/8	true false		Boolean
Pomocný prázdný typ				
void				

byte +-*/ byte ... int
short +-*/ short ... int
int +-*/ int ... int (pozor na přetečení)
long +-*/ long ... long

2.2.2 Typové konverze

Typová konverze je operace, která hodnotu nějakého typu převede na hodnotu jiného typu.

- implicitní - např. int na double (když se očekává hodnota double, ale je tam INT, dojde k implicitní konverzi) - implicitní konverze je bezpečná

- explicitiní - programátor musí explicitně označit, je potenciálně nebezpečná (může dojít ke ztrátě informace) - eg. double -> int, double d = 1E30; int i = (int)d; // i je 2147483647 asi 2E10

2.3 Výrazy

Výraz se skládá z operandů a operátorů. **Operandem** může být konstanta, proměnná, volání metody nebo opět výraz. **Operátory** udávají, co se má provést s jednotlivými hodnotami operandů.

Operátory a jejich priorita

priorita	operátor	typ operandu	asociativita	operace
1	++	aritmetický	P	pre/post inkremetace
	--	aritmetický	P	pre/post dekrementace
	-	aritmetický	P	unarní plus/minus
	~	celočíselný	P	bitová inverze
	!	logický	P	logická negace
	(typ)	libovolný	P	přetypování
2	*, / ,%	aritmetický	L	násobení, dělení, zbytek
3	-	aritmetický	L	odečítání
	+	aritmetický, řetězový	L	sčítání, zřetězení
4	<<	celočíselný	L	posun vlevo
	>>	celočíselný	L	posun vpravo
	>>>	celočíselný	L	posun vpravo s doplňováním nuly
	<,>, <=, >=	aritmetický	L	porovnání
	instanceof	objekt	L	test třídy
	==, !=	primitivní	L	rovno, nerovno
7	&	celočíselný nebo logický	L	bitové nebo logické AND
8	^	-"-	L	bitové nebo logické XOR
9	|	-"-	L	bitové nebo logické OR
10	&&	logický	L	logické AND vyhodnocované zkráceně
11	||	logický	L	logické OR vyhodnocované zkráceně
12	? :	logický	P	podmíněný operátor
13	=, -=, *=, /=, %=, ==, &=, ^=, |=	libovolný	P	přiřazení

Pozor na asociativitu. Odčítání je asociativní zleva, mocnění zprava.
Přikazovací příkaz může být výraze: y = x = x + 6, tj. y = (x = (x + 6));
Operace && a || se vyhodnocují zkráceným způsobem, druhý operand se nevyhod-

nocuje, když lze výsledek určit již z prvního operandu.

2.4 Výstup

Pro výpis dat na obrazovku se v Javě používá příkaz System.out.println(parametr); = výpis na standardní výstup. Metoda je přetížená, jako její parametr lze zadat všechny primitivní datové typy i String.

2.4.1 Formátovaný výstup

Př. System.out.printf("Cislo Pi = %6.3f %n ", Math.PI);
Specifikace 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.

Formátovaný výstup

```
package pr1_3;
public class FormatovanyVystup {
public static void main(String[] args) {
System.out.printf("Cislo Pi = %6.3f %n", Math.PI);
System.out.printf("Cislo Pi = %8.3f %n", Math.PI);
System.out.printf("Cislo Pi = %6.5e %n", Math.PI);
System.out.printf("Cislo Pi = %6.5g %n", Math.PI);
System.out.printf("Cislo Pi = %6d %n", 33);
System.out.printf("Cislo Pi = %4d %n", -33);
    }
}
```

```
Cislo Pi =  3,142
Cislo Pi =    3,142
Cislo Pi = 3.14159e+00
Cislo Pi = 3.1416
Cislo Pi =     33
Cislo Pi = -33
```

2.5 Vstup

Pro vstup dat zadaných na klávesnici poslouží třída Scanner. Je třeba vytvořit objekt třídy Scanner a napojit jej na standardní vstupní proud: Scanner sc = new Scanner(System.in);

- sc.nextInt() : přečte celé číslo z řádku zadaného klávesnicí (řádek je zakončen klávesou Enter, číslo je zakončeno mezerou nebo Enter) a vrátí je jako funkční hodnotu typu int
- sc.nextDouble() : přečte číslo z řádku zadaného klávesnicí a vrátí je jako funkční hodnotu typu double, jako oddělovač použijte . nebo čárku v závislosti na definovaném jazyku OS. Lze změnit pomocí před vytvořením Scanneru Locale.setDefault(Locale.ENGLISH);
- sc.nextLine() : přečte zbytek řádku zadaného klávesnicí a vrátí je jako funkční hodnotu typu String

2.6 Řídicí struktury

Řídicí struktura je programová konstrukce, která se skládá z dílčích příkazů a předepisuje pro ně způsob provedení. Tři druhy řídicích struktur:

1. 1. posloupnost, předepisující postupné provedení dílčích příkazů
2. 2. větvení, předepisující provedení dílčích příkazů v závislosti na splnění určité podmínky
3. 3. cyklus, předepisující opakované provedení dílčích příkazů v závislosti na splnění určité podmínky

2.6.1 Podmínka

Příkaz if (podmíněný příkaz) umožňuje větvení na základě podmínky.

- if (podmínka) příkaz1 else příkaz2
- if (podmínka) příkaz1

2.6.2 Cyklus

Základní příkaz cyklu, který má tvar: while (podmínka) příkaz (blok příkazů).

Příkaz cyklu do se od příkazu while liší v tom, že podmínka se testuje až za tělem cyklu. Tvar příkazu: do příkaz while (podmínka);

Poznámka k sémantice: příkaz do provede tělo cyklu alespoň jednou, nelze jej tedy použít v případě, kdy lze očekávat ani jedno provedení těla cyklu

Cyklus For: je často řízen proměnnou, pro kterou je stanoveno:

- jaká je počáteční hodnota

- jaká je koncová hodnota
- jak změnit hodnotu proměnné po každém provedení těla cyklu

2.6.2.1 Konečnost cyklů

Vstupní podmínku konečnosti cyklu lze určit téměř ke každému cyklu (někdy je to velmi obtížné, až nespočetné). Splnění vstupní podmínky konečnosti cyklu musí zajistit příkazy předcházející příkazu cyklu.

2.6.3 Switch

Příkaz switch (přepínač) umožňuje větvení do více větví na základě různých hodnot výrazu (nejčastěji typu int nebo char).

Sémantika (zjednodušeně):

- vypočte se hodnota výrazu a pak se provedou ty příkazy, které jsou označeny konstantou označující stejnou hodnotu
- není-li žádná větev označena hodnotou výrazu, provedou se příkazydef

2.7 Funkce

Funkce v programování je část programu, kterou je možné opakovaně volat z různých míst kódu. Funkce může mít argumenty (též parametry) – údaje, které jí jsou předávány při volání – a návratovou hodnotu, kterou naopak vrací.

Deklaraci funkce tvoří hlavička funkce a 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

2.7.1 Parametry

Parametry funkce slouží pro předání vstupních dat algoritmu, který je funkcí realizován. Parametr může být výraz.

2.8 Procedura

Funkce, jejíž typ výsledku je void, nevrací žádnou hodnotu.

2.9 Rozklad problému na podproblémy

Postupný návrh programu rozkladem problému na podproblémy:

- zadaný problém rozložíme na podproblémy
- pro řešení podproblémů zavedeme abstraktní příkazy
- s pomocí abstraktních příkazů sestavíme hrubé řešení
- abstraktní příkazy realizujeme pomocí procedur (void)

2.10 Rekurze

V imperativním programování rekurze představuje opakované vnořené volání stejné funkce (podprogramu).

2.10.1 Rekurzivní algoritmus

Rekurzivní algoritmus předepisuje výpočet „shora dolů“ v závislosti na velikosti (složitosti) vstupních dat:

- pro nejmenší (nejjednodušší) data je výpočet předepsán přímo
- pro obecná data je výpočet předepsán s využitím téhož algoritmu pro menší (jednodušší) data

Výhoda: jednoduchost a přehlednost (to je diskutabilní).
Nevýhoda: Nevýhodou může být časová náročnost způsobená např. zbytečným opakováním výpočtu.

2.10.2 Rekurzivní funkce

Rekurzivní funkce (procedury) jsou přímou realizací rekurzivních algoritmů. Použití: faktoriál, fibonacciho posloupnost (začíná 1,1,2,3,5,8...). Fibonacci rekurzí (složitost exponenciální 2^n):

```
static int fib(int i) {
   if (i<2) return 1;
   return fib(i-1)+fib(i-2);
}
```

Fibonacci iteračně (složitost 3n):

```
static int fib(int n) {

    int i, fibNMinus2=1;
    int fibNMinus1=1, fibN=1;
    for (i=2; i<=n; i++) {
```

```
            fibNMinus2 = fibNMinus1;
            fibNMinus1 = fibN;
            fibN = fibNMinus1 + fibNMinus2;
        }

    return fibN;
    }
```

2.11 Iterace

Iterace v programování znamená opakované volání funkce v počítačovém programu. Zvláštní formou iterace je rekurze. Pro naše účely lze chápat iteraci jako opakované provádění bloku příkazu, typicky pomocí cyklu.