# Teoretické minimum z PJV

Pozn.: následující text popisuje vlastnosti jazyka Java zjednodušeně pouze pro potřeby výuky.

## *Třída*

Zavádí se v programu deklarací třídy což je část programu od klíčových slov „public class“ do závěrečné závorky: }.

Třída:

1. Je prostředek pro členění programu do menších celků – umožňuje sdružovat dohromady související třídní (static) proměnné a třídní (static) metody.
2. Deklarací instančních proměnných definuje strukturu objektů tj. z jakých položek (fields) se budou skládat objekty této třídy.
3. Deklarací instančních metod určuje jaké služby budou objekty této třídy poskytovat.
4. Definicí konstruktorů určuje, jak je možno objekty po jejich vzniku inicializovat.

Příklad jednoho souboru (Vektor.java) zdrojového textu programu (tzv. kompilační jednodnotka):

```
// deklarace balíčku, ve kterém je třída umístěna:
package vektory;

// import klauzule
import java.lang.*;

// začátek deklarace třídy:
public class Vektor extends Object {

// Konstanta, která může být deklarována v kterékoliv třídě,ale logicky
// patří do třídy Vektor:
     public static final vektory.Vektor NULOVY_VEKTOR
                = new vektory.Vektor();

// Třídní (statická) metoda, která může být deklarována v kterékoliv
třídě,
// ale logicky patří do třídy Vektor:
     public static vektory.Vector soucetVectoru(vektory.Vector a,
                                                vektory.Vector b) {
          return new vektory.Vector(a.x + b.x, a.y + b.y);
     }

// Statická proměnná, která může být deklarována v kterékoliv třídě,
// ale logicky patří do třídy Vektor:
     public static double delkaVsechVytvorenychVektoru;

// Instanční proměnné tj. položky (fields) objektů třídy Vektor:
     double x;
     double y;

// Instanční metody (tj. služby poskytované objekty třídy Vektor):
     public double delka() {
          return Math.sqrt(this.x * this.x + this.y * this.y);
```

```
      }
      public void inverze() {
            this.x = -this.x;
            this.y = -this.y
      }

// Konstruktory, které inicializují objekty třídy Vektor – obvykle
// hodnotami svých argumentů:
      Vektor(double x, double y) {
            super();
            this.x = x; this.y = y;
            Vektor.delkaVsechVytvorenychVektoru += this.delka();
      }
      Vektor() {
            this(0, 0);
      }
} // konec deklarace třídy Vektor
```

## *Objekt*

(instance třídy) je souvislý paměťový prostor, ve kterém jsou umístěny instanční proměnné deklarované ve třídě. Prostor je alokován v paměťové oblasti nazývané heap. Objekt je jednoznačně identifikován referenční hodnotou (zkráceně referencí).

Objekt vzniká operátorem new. Vznikem objektu rozumíme alokaci paměťového místa na heapu a dále inicializaci tohoto místa některým konstruktorem. Při vzniku objektu současně automaticky vzniká reference na tento objekt. V programu může současně existovat více referencí na jeden objekt, protože přiřazováním se reference duplikuje.

Pomocí reference lze:

- přistupovat k položkám objektu,
- volat instanční metody objektu

Objekt je v programu dostupný pouze tehdy, pokud nezanikly všechny jeho reference. Zánikem poslední reference na objekt přestane být objekt z programu dostupný a jeho alokovaný paměťový prostor může být uvolněn pro další použití (garbage collecting).

Objekt:

- není proměnná, takže se nedeklaruje a nemá jméno a typ (v C++ může být objekt proměnná),
- není hodnota, takže jej nelze přiřazovat do proměnných, předávat jako parametr nebo vracet jako výsledek funkce – naproti tomu referenci na objekt lze (v C++ může být objekt hodnota),
- uchovává hodnoty ve svých položkách (instančních proměnných),
- poskytuje služby (operace) deklarované jako instanční metody třídy objektu.

## *Operátor this*

Služby, které objekt poskytuje, jsou definovány instančními metodami třídy objektu. Instanční metody tedy musí mít k tomuto objektu přístup a tedy musí mít k dispozici jeho referenci. Při volání instanční metody se před její jméno uvádí referenční výraz, jehož hodnotou je právě tato

požadovaná reference:

```
Vektor v  = new Vektor(1,2);
System.out.println(v.delka());  // v je výraz referenčního typu
```

Reference na objekt se předá instanční metodě stejným mechanismem jako ostatní skutečné parametry (argumenty). Tento speciální "skrytý parametr" však není deklarovaný, a proto musí být jeho atributy (tj. jméno a typ) definovány implicitně:

- je referenčního typu třídy, ve kterém je instanční metoda deklarovaná
- jeho "jméno" je tvořeno klíčovým slovem this:

```
public double delka( /*jakoby zde bylo deklarováno: Vektor this */)
{
      return Math.sqrt(this.x * this.x + this.y * this.y);
}
```

Obdobným způsobem jako instančním metodám se také konstruktoru předává reference na objekt, který má být inicializován. V tomto případě je reference výsledkem operátoru new, který objekt vytvořil. Reference, kterou vrací operátor new se tedy využije dvojím způsobem:

- předá se dovnitř konstruktoru jako hodnota "skrytého parametru" this,
- vrátí se jako výsledek, který je možno dále např. uložit do proměnné:

```
Vektor v = new Vektor(/* reference vrácená oprátorem new */ 1,2);
```

## *Dědičnost*

Dědění není operace, která by se prováděla v nějakém okamžiku. Dědičnost je vztah mezi dvěma třídami (resp. rozhraními). V tomto vztahu je vždy jedna třída (resp rozhraní) tzv. nadtřídou a druhá tzv. podtřídou.  Jsou možné všechny kombinace s výjimkou, že rozhraní nemůže být podtřídou třídy.

Třída (resp. rozhraní) obsahuje všechny proměnné a metody které:

- jsou ve třídě přímo deklarovány,
- jsou deklarovány v nadtřídě a nejsou označeny private (říkáme nepřesně, že jsou do třídy zděděny).

Každá třída s výjimkou třídy java.lang.Object  má vždy právě jednu nadtřídu, která je třída, a žádnou nebo více nadtříd které jsou rozhraní.

## *Abstrakce a překrývání*

Deklarace instanční metody se skládá z hlavičky (header) a těla (body). Hlavička vyjadřuje kontrakt (smlouvu) mezi uživatelem metody (tj. kódem, kde se metoda volá) a poskytovatelem metody (tj. třídou, kde je deklarovaná). Tělo metody definuje implementaci (tj. realizaci) tohoto kontraktu.

Tělo metody může být v deklaraci vynecháno. Takovou metodu nazýváme abstraktní (neúplnou) a musí být označena klíčovým slovem abstract.

Pokud je ve třídě alespoň jedna abstraktní metoda, nazýváme celou třídu abstraktní a musí být sama označena klíčovým slovem abstract. Od takové třídy nemůžeme vytvářet objekty, protože tyto objekty by nemohly plnit svůj kontrakt.

Zděděnou instanční metodu je možno tzv. překrýt (override).  Přitom,

- pokud je zděděná metoda abstraktní, je mechanismem překrytí možno doplnit chybějící tělo,
- pokud není abstraktní, je možno změnit původní implementaci (tělo) za jinou.

Mechanismem překrýváním vzniká skupina metod, které mají stejnou hlavičku (tedy stejný kontrakt), avšak liší se svými těly (tedy implementacemi kontraktu). Tato skupina metod reprezentuje nějakou obecnou vlastnost (delka), operaci (inverze), která je společná pro všechny třídy, ve kterých jsou jednotlivé překrývající se metody deklarovány.

## *Rozhraní*

Rozhraní je speciální třída, ve které jsou deklarovány pouze kontrakty – tedy pouze abstraktní metody. Nelze tedy – stejně jako u abstraktních tříd – vytvářet od rozhraní objekty.

## *Polymorfismus a pozdní vázání*

Hodnotou výrazu referenčního typu může být reference na objekt třídy referenčního typu, ale také reference na objekt její libovolné podtřídy. V tomto smyslu je referenční typ polymorfní (mnohotvarý), protože obsahuje reference na objekty různého "tvaru":

```
// do o se může přiřadit reference na objekt libovolné třídy
     Object o = ...;
```

Vzhledem k polymorfismu referenčních typů a překrývání metod nelze při kompilaci stanovit, která konkrétní instanční metoda bude v daném místě volána. Proto se konkrétní metoda určuje až při výpočtu podle skutečné třídy referencovaného objektu. Tato vlastnost se nazývá pozdní vázání:

```
     //která konkrétní metoda toString() se bude volat?
     String s = o.toString();
```

## *Hodnota*

Elementární část informace (celá čísla, desetinná čísla, logické hodnoty, znaky, reference).

1. Hodnoty jsou:
   a) Zapsány přímo ve zdrojovém kódu programu jako literály (znak: 'a', celé číslo: 99, celé číslo v hexadecimálním tvaru: 0xAA, desetinné číslo: 2.72, logická hodnota pravda: true, ...).
   b) Načteny ze vstup některou ze vstupních operací (readInt, readDouble, ...)
   c) Vytvořeny výpočtem operace ve výrazu.
2. Hodnota je vždy nějakého typu.
3. Mezi provedením dvou po sobě jdoucích příkazů jsou hodnoty uloženy pouze v proměnných. V proměnné může být uložena pouze hodnota takového typu, který je proměnné přiřazen deklarací.
4. Přiřazením jedné proměnné do druhé vznikne kopie hodnoty.
5. Hodnoty zanikají:
   a) Přepsáním hodnoty v proměnné jinou hodnotou.
   b) Zánikem proměnné, ve které je hodnota uložena.

## Reference

Speciální hodnota, jejímž významem je identifikace objektu (jako je rodné číslo identifikace občana).

Vzniká obvykle jako výsledek operace new (také readObject, newInstance a operací + nad řetězy).

Můžeme si ji představovat jako adresu (32 bitové nezáporné číslo) prvního bytu paměťového prostoru objektu. Protože jejím významem není numerická hodnota, nesmí se s ní provádět numerické operace +, * atd. (stejně jako nemá smysl sčítat resp. násobit rodná čísla). Povolené operace s referencemi jsou:

- přiřazení
- porovnání na rovnost a nerovnost
- přístup k položce objektu
- volání instanční metody

Pozn.: Jazyk C/C++ dovoluje provádět s referencemi některé aditivní operace. Tím se ale podstatně sníží bezpečnost programování a tedy spolehlivost programů.

## *Typ*

Java je silně typový jazyk, což znamená, že o každé hodnotě, proměnné a výrazu je v čase kompilace programu znám typ. Typ:

- určuje množinu hodnot, které mohou být uloženy v proměnné daného typu, resp. mohou být výsledkem výrazu daného typu (o každé hodnotě z této množiny říkáme, že je daného typu),
- určuje operace povolené s hodnotami tohoto typu,
- určuje defaultní hodnotu pro inicializaci proměnných.

Dělení typů:

1. Primitivní typy
   a) množina hodnot je předem přesně dána definicí jazyka Java,
   b) v textu programu se hodnoty dají přímo zapisovat pomocí literálů,
   c) označují se klíčovými slovy:
      - numerické (byte, short, char, int, long, float, double) – pro jednotlivé hodnoty je přesně stanovena reprezentace (tj. způsob zápisu) hodnot v paměti počítače (bit po bitu).
      - logický (boolean) – reprezentace hodnot (true, false) je daná konkrétní implementací Javy.
   d) Pro hodnoty typu char, int, long, float, double a boolean jsou definovány literály tj. přímý zápis hodnoty v programu:

```
99 // hodnota typu int
24L  // hodnota typu long
1.1E10// hodnota typu double
3.14F // hodnota typu float
'a'  // hodnota typu char
true // hodnota typu boolean
```

1. Referenční typy:
   a) označují se jménem třídy (resp. rozhraní) nebo typem pole,
   b) existuje pouze jediný literál pro vyjádření referenční hodnoty: null, který označuje hodnotu „prázdná reference“.
   c) množina hodnot daného referenčního typu není předem definována specifikací Javy a mění se během výpočtu programu. Na začátku obsahuje typ pouze hodnotu null. Při výpočtu se rozrůstá o nové reference, které jsou přidělovány nově vznikajícím objektům. Množina typu se také může zmenšovat zánikem referencí.

## *Proměnná*

Paměťové místo, ve kterém je uložena hodnota příslušného typu. Proměnná má atributy jméno a typ, který se určují deklarací proměnné. Proměnná nemá referenci (V jazyku C/C++ proměnná referenci má). Pravidla Javy zaručují, že proměnná použitá ve výrazu má vždy definovanou hodnotu (defaultní, danou inicializací, přiřazením nebo předáním skutečného parametru). Proměnné jsou:

- třídní (statické) – paměťové místo je alokováno v globálním adresním prostoru. K dané deklaraci existuje vždy právě jedno paměťové místo. Třídní proměnná existuje od začátku do konce běhu programu. Inicializuje se při spuštění programu a pokud není v deklaraci uvedena inicializace, je jí automaticky přiřazena defaultní hodnota.
- instanční – paměťové místo je alokováno v rámci objektu. K dané deklaraci instanční proměnné existuje v daném okamžiku tolik paměťových míst, kolik existuje objektů příslušné třídy. Proměnná zaniká ukončením existence objektu. Inicializuje se při inicializaci objektu a pokud není v deklaraci uvedena inicializace, je jí automaticky přiřazena defaultní hodnota.
- lokální proměnná – paměťové místo je alokováno na zásobníku provedením deklaračního příkazu. Lokální proměnná se neinicializuje na defaultní hodnotu (na rozdíl od třídních a instančních). Proměnná zaniká ukončením bloku, ve kterém je deklarační příkaz uveden. K dané deklaraci lokální proměnné existuje tolik paměťových míst, kolikrát je v daném okamžiku blok rekurzivně zavolán.
- parametr – paměťové místo je alokováno na zásobníku při volání metody. Parametr je inicializován hodnotou skutečného parametru (argumentu) tj. výrazu uvedeného ve volání metody. K dané deklaraci parametru existuje tolik paměťových míst, kolikrát je v daném okamžiku metoda rekurzivně zavolána. Parametr zaniká ukončením metody.

## *Jméno*

Některé programové entity se označují jmény a jiná jsou bezejmenná. Jména mají:

- balíčky
- třídy a rozhraní
- proměnné (včetně parametrů)
- metody

Bezejmenné jsou:

- hodnoty

- výrazy
- příkazy
- objekty

Jména rozlišujeme na jednoduchá a kvalifikovaná (obdoba jména souboru a cesty k souboru). Jednoduchá jména jsou tvořena identifikátorem a v programu se zavádějí deklaracemi.

## Identifikace pojmenovaných entit v programu

Protože se stejný identifikátor může použít ve více různých deklaracích, musí existovat mechanismus, který jednoznačně určí význam identifikátoru použitého v programu (tj. kterou entitu chceme v daném místě použít):

- Balíčky, třídy a rozhraní, třídní metody a třídní proměnné se v programu identifikují (určují) kvalifikovanými jmény:
  - Kvalifikovaná jména tříd a rozhraní jsou tvořena tak, že se před jednoduché jméno třídy uvede jméno balíčku, ve kterém je třída umístěna:

    `vektory.Vektor`
  - Kvalifikovaná jména třídních proměnných a třídních metod jsou tvořena tak, že se před jednoduché jméno proměnné resp. metody uvede kvalifikované jméno třídy ve kterém je deklarace proměnné resp. metody umístěna:

    `vectory.Vector.NULOVY_VEKTOR`

  Pro určení těchto entit v programu lze vždy použít kvalifikované jméno. Často je ale možno, tam kde nemůže dojít k nejednoznačnosti, použít jméno jednoduché.
- Instanční proměnné, instanční metody, lokální proměnné a parametry se identifikují svými jednoduchými jmény.
  - Lokální proměnné a parametry existují pouze v bloku, ve kterém jsou nadeklarovány, takže pro jejich identifikaci stačí jejich jednoduché jméno.
  - Před voláním instanční metody resp. přístupem k instanční proměnné musí být uveden referenční výraz, jehož typ určí význam následujícího jednoduchého jména:

```
Vektor v = new Vektor(1,2);
// typ proměnné v určí význam identifikátorů x a delka
System.out.println(v.x);
System.out.println(v.delka());
```

## *Konverze*

Každý výraz v Javě má čase kompilace přiřazen atribut typ. Implicitně je tento typ dán typy operandů, které jsou ve výrazu použity. Konverze je změna tohoto implicitního typu. Konverze se v programu zapisuje operací přetypování:

```
public static int CELA_CAST_PI = (int)Math.PI;
```

Přetypování je většinou možno vynechat (provede se automaticky).

Konverze:

- Přiřazovací konverze – pokud je proměnná jiného typu než výraz, jehož hodnota se do ní

přiřazuje, musí se výraz zkonvertovat na typ proměnné.

- Unární numerické povýšení – pokud je operandem operace výraz, jehož typ je byte, short nebo char, automaticky se zkonvertuje na typ int.
- Binární numerické povýšení – pokud jsou operandy binární operace (s výjimkou posuvů) výrazy nestejných typů, tak se operand „menšího“ typu automaticky zkonvertuje na hodnotu většího typu:
  int → long → float → double
  Tento typ je pak také typem celého výrazu.

Konverze výrazu znamená v době výpočtu:

- prázdnou operaci
- skutečnou změnu reprezentace hodnoty
- test, zda je konverze přípustná a v opačném případě hození výjimky

## Konverze hodnot primitivních typů

Hodnoty typu boolean nelze konvertovat. Hodnoty všech numerických typů lze konvertovat na libovolný jiný numerický typ. Rozlišujeme konverze:

- **rozšiřující** – výraz se konvertuje na typ, který má „širší“ reprezentaci, takže nedojde ke ztrátě hodnoty (může dojít ke ztrátě přesnosti při konverzi int → float):

```
byte b = 10;
int i = (int)b;
```

- **zužující** – výraz se konvertuje na typ, který má „užší“ reprezentaci, takže může dojít ke ztrátě hodnoty, pokud se do nové reprezentace nevejdou významné číslice původní hodnoty:

```
int i = 266;
byte b = (byte)i; // v b je hodnota 10;
```

## Konverze referencí

Při konverzi referencí nedochází ke změně reprezentace. Konvertovat lze pouze referenční typy, jejichž třídy jsou ve vztahu nadtřída – podtřída:

```
String s  = "999";
Integer i = (Integer)s; // chyba při kompilaci
```

Rozlišujeme konverze:

- **rozšiřující** – výraz se konvertuje na takový referenční typ, jehož třída je nadtřídou třídy původního typu. Při této konverzi nemůže dojít k chybě:

```
String s = "XYZ";
Object o = (Object)o;
```

- **zužující** - výraz se konvertuje na takový typ, jehož třída je podtřídou původního typu. V takovém případě se v době běhu kontroluje, zda konvertovaná reference patří do nového typu a v opačném případě je generována výjimka ClassCastException:

```
try {
     Object o = ...;
     String s = (String)o; // je v o skutečně reference na String?
} catch (ClassCastException e) { ... }
```

## *Struktura programu*

Struktura programu je odvozena od struktury souborového systému a skládá se z:

- balíčků – odpovídají adresářům,
- podbalíčků – odpovídají podadresářům,
- tříd – odpovídají souborům.

Zdrojový kód programu se zapisuje do jednotlivých textových souborů, které se nazývají kompilační jednotky. Kompilační jednotka obsahuje:

- deklaraci package – deklarace balíčku do kterého patří následující deklarovaná třída
- klauzule import
- deklaraci třídy

Kompilační jednotka (zdrojový soubor) se jmenuje jednoduchým jménem deklarované třídy s příponou .java (Vektor.java).

## *Exekutivní tvar programu*

Aby bylo možno program vykonat musí se převést do vykonavatelné formy, která se nazývá jarfile (obdoba exe formátu programů pro Windows). Postup převodu je následující:

- Založí se pomocný adresář (obvykle se jménem class)
- Pro každý balíček javovského programu se v adresáři class vytvoří podadresář, jehož relativní cesta odpovídá kvalifikovanému jménu balíčku (class/vektory)
- Každá kompilační jednotka se přeloží programem javac, čímž vznikne stejnojmenný soubor avšak s příponou .class (takzvaný classfile). V tomto souboru je deklarace třídy převedená do vnitřní binární formy nazývané byte-code. Classfile se uloží do takového podadresáře adresáře class, který odpovídá balíčku přeložené třídy: class/vektory/Vektor.class.
- Pomocný adresář class se zkomprimuje do formátu zip a výslednému souboru se dá libovolné jméno s příponou .jar (např. JavaVektory.jar)

Spuštění programu se pak provede voláním interpretu java, kterému se dá jméno jarfilu jako parametr:

```
java –jar JavaVektory.jar
```

## *Co je možno vynechávat*

V předcházejícím textu jsme záměrně pro dosažení co největší ilustrativnosti používali nezkrácený zápis programu. V běžných programech se ovšem obvykle používá zkrácený zápis a vynechávají se ty části, které kompilátor implicitně doplňuje:

- u tříd, které mají přímo nadtřídu Object klauzuli extends:

  ```
  class Vektor // extends Object
  {
  ```

- import klauzuli pro základní balíček:

  ```
  // import java.lang.*;
  ```

- kvalifikace jména třídy v balíčku, kde je deklarovaná:

```
public static final Vektor NULOVY_VEKTOR =
     new   /* vektory.*/ Vektor();
```

- kvalifikaci jména třídní proměnné a třídní metody ve třídě kde jsou deklarovány resp. zděděny:

```
Vektor(double x, double y) {
     this.x = x; this.y = y;
     /* Vektor. */ delkaVsechVytvorenychVektoru += this.delka();
}
```

- operátor this při přístupu k instančním proměnným a volání instančních metod v instančních metodách a konstruktorech:

```
Vektor(double xx, double yy) {
     /* this. */ x = xx;
     /* this. */ y = yy;
     delkaVsechVytvorenychVektoru += /* this. */ delka();
}
```

- volání konstruktoru bez parametrů z nadtřídy jako první příkaz v těle konstruktoru:

```
Vektor(double x, double y) {
     // super();
     this.x = x; this.y = y;

}
```

- přetypování v rozšiřující konverzi:

```
int x = 10;
long l = /* (long) */ x;
```

- přetypování při přiřazování literálu typu int a v unárním a binárním povýšení:

```
byte b = /* (byte) */ 34
double x = 10.0 * /* (double) */ - /* (int) */ b;
```