Soubory a výjimky v Javě, třídní proměnné a výčtové typy

Soubory

- Soubor je množina údajů uložená ve vnější paměti - obvykle na disku.
- Pro soubor jsou typické tyto operace.
  - otevření souboru
  - čtení údaje
  - zápis údaje
  - uzavření souboru
- Přístup k údajům (čtení nebo zápis) může být:
  - sekvenční (proudový) - pouze postupné (sekvenční) čtení nebo zápis údajů
  - nahodilý (přímý) - náhodné čtení nebo zápis údaje (podobně jako pole)
- Způsob přístupu k údajům v souboru je dán formátem dat, programem a hardwarem
- Zde se budeme zabývat pouze sekvenčními soubory

Soubor jako posloupnost bytů

- V jazyku Java slouží pro práci se soubory třídy v balíku *java.io*
  Soubor jako posloupnost bytů reprezentují třídy:

FileInputStream	vstupní soubor (pro čtení)
FileOutputStream	výstupní soubor (pro zápis)

- Příklad: kopie souboru

```
import java.io.*;
public class Kopie1 {

 public static void main(String[] args) throws IOException {
  FileInputStream in = new FileInputStream("vstup.txt");
  FileOutputStream out = new FileOutputStream("vystup.txt");
  int b = in.read();
     while ( b!= -1 ) {
        out.write(b);
        b = in.read();
  }
  out.close();
  in.close();
 }
}
```

Soubor jako proud dat

Komentář k příkazům:

```
FileInputStream in = new FileInputStream("vstup.txt");
```

- vytvořený objekt ***in*** reprezentuje vstupní soubor uložený na disku v aktuálním adresáři pod názvem ***vstup.txt***; pokud takový soubor neexistuje, nastane chyba

```
FileOutputStream out = new FileOutputStream("vystup.txt");
```

- vytvořený objekt ***out*** reprezentuje výstupní soubor, který bude uložen do aktuálního adresáře pod názvem ***vystup.txt***; pokud by soubor nebylo možné vytvořit, nastane chyba

```
b = in.read();
```

- ze souboru ***in*** metoda přečte jeden byte a vrátí číslo v rozsahu ***0..255***, není-li v proudu žádný nepřečtený byte, vrátí ***–1***

```
out.write(b);
```

- do souboru ***out*** se zapíše nejnižší byte z ***int***

```
out.close();
in.close();
```

- způsobí uzavření souborů ***in*** a ***out***

Výjimky

- Při operacích se soubory (a i při jiných operacích) mohou nastat mimořádné stavy (např. chyby). Ty lze ošetřit a pokračovat dál. K ošetření slouží mechanismus výjimek (exceptions)
- Princip výjimek v jazyku Java:
  - metoda či konstruktor může skončit standardně nebo nestandardně tzv. vyhozením výjimky určitého typu, pokud nastala mimořádnost. Výjimka je objekt.
  - pokud vyhozena výjimka, další příkazy se neprovedou a řízení se předá konstrukci ošetřující výjimku daného typu (tzv. ***try-catch***)
  - pokud taková konstrukce v těle metody či konstruktoru není, skončí nestandardně a výjimka se šíří na dynamicky nadřazenou úroveň
  - není-li výjimka ošetřena ani ve funkci ***main()***, vypíše se a program skončí
  - pro rozlišení různých typů výjimek je v jazyku Java zavedena řada knihovních tříd

Výjimky při práci se soubory

- Příklady operací, které mohou vyhodit výjimky:

  `new FileInputStream(...);`

  pokud soubor neexistuje, vyhodí se výjimka typu:

  java.io.FileNotFoundException

  `new FileOutputStream(...);`

  pokud soubor nelze vytvořit (např. špatně zadaná cesta), vyhodí se výjimka:

  java.io.FileNotFoundException

  `in.read(); out.write(b);`

  nepodaří-li se data přečíst nebo zapsat, vyhodí se výjimka:

  java.io.IOException

- Obsahuje-li metoda deklarace a příkazy, které mohou vyhodit výjimky a tyto výjimky nejsou v metodě ošetřeny, musí být seznam typů výjimek, které se mohou z metody šířit, uveden v hlavičce metody:

```
public static void main(String[] args) throws
FileNotFoundException, IOException { }
```

Ošetření výjimek

- Chceme-li ošetřit výjimku, která může vzniknout při provádění určité operace, je třeba tuto operaci:
  - vložit do bloku *try*, ke kterému je připojen
  - blok (klauzule) *catch* ošetřující (zachytávající) výjimky specifikovaných typů
- Příklad: funkce si vyžádá zadání jména vstupního souboru z klávesnice a pokud najde takový soubor, otevře jej.

```
static FileInputStream vstup() {
   Scanner sc = new Scanner(System.in);
   System.out.println("zadejte vstupni soubor: ");
   String jmeno = sc.readLine();
   if (jmeno.equals("")) System.exit(0);
   try {
       FileInputStream in = new FileInputStream(jmeno);
       return in;
   } catch (FileNotFoundException ex) {
       System.out.println("soubor neexistuje");
}}}
```

Ošetření výjimek

- Nové řešení kopírování souboru
- V metodě *main()* ošetříme výjimku typu *IOException*, která mohou vyhodit metody *read()*, *write()* a *close()*

```
public static void main(String[] args) {
  FileInputStream in = vstup();
  FileOutputStream out = vystup();  // předpoklad
  try {
    int b = in.read();
    while (b != -1) {
      out.write(b);
      b = in.read();
    }
    in.close();
    out.close();
  } catch (IOException ex) {
     ex.printStackTrace();
  }
}
```

Argumenty příkazového řádku

- Provedení programu v jazyku Java lze zadat po překladu do bytekódu příkazovým řádkem

```
java Program
```

ten může obsahovat další argumenty

```
java Program arg0 arg1 ...
```

- Argumenty jsou přístupné metodě main pomocí jejího parametru, což je reference na pole řetězců

```
public static void main(String[] args)
```

- Příklad: spustíme-li program takto:

```
java Program prvni 2222 Treti
```

pak v metodě main třídy Program platí:

  - hodnotou args[0] je řetěz “prvni”
  - hodnotou args[1] je řetěz “2222”
  - hodnotou args[2] je řetěz “Treti”
  - hodnotou args.length je 3

- Pozn: nedochází ke konverzi řetezce na jiný typ

Zadání souborů příkazovým řádkem

```
public class Kopie {
 public static void main(String[] ar) {
  if (ar.length<2) {
   System.out.println("použití: Kopie <vstup>
   <výstup>");
   return;
  }
  try {
   FileInputStream in = new FileInputStream(ar[0]);
   FileOutputStream out = new FileOutputStream(ar[1]);
   int b = in.read();
   while (b! = -1) {
    out.write(b);
    b = in.read();
   }
   out.close();
   in.close();
  } catch (Exception ex) {
      ex.printStackTrace();
  }}
```

Další verze kopie souboru – Kopie4

Metoda ***available()*** - vrací počet nepřečtených bytů souboru

```
public class Kopie {
  public static void main(String[] ar) {
    if (ar.length<2) {
      System.out.println("use: Kopie <vstup> <výstup>");
      return;
    }
    try {
      FileInputStream in = new FileInputStream(ar[0]);
      FileOutputStream out = new FileOutputStream(ar[1]);
      while (in.available()>0) out.write(in.read());
      out.close();
      in.close();
    } catch ( Exception ex ) {
      ex.printStackTrace();
    }
  }
}
```

Soubor jako posloupnost primitivních typů

- Třídy FileInputStream, FileOutputStream zpřístupňují soubor jen jako proud bytů.
- Soubor jako posloupnost primitivních typů umožňují třídy DataOutputStream a DataInputStream
- Příklad: program vytvoří soubor 100 náhodných čísel typu double a pak soubor přečte a vypíše součet čísel (jiné zpracování výjimky):

```
public class Cisla {
  public static void main(String[] ar) throws Exception {
    DataOutputStream out = new DataOutputStream(
      new FileOutputStream("temp.bin"));
    for (int i=0; i<100; i++) out.writeDouble(Math.random());
    out.close();

    DataInputStream in = new DataInputStream(
      new FileInputStream("temp.bin"));
    double soucet = 0;
    while (in.available()>0)
      soucet = soucet + in.readDouble();
    System.out.println(soucet);
  }
```

Operace se souborem primitivních typů

- DataOutputStream

  void writeTyp(Typ x) throws IOException

  – do souboru zapíše hodnotu *x* primitivního typu *Typ*

  Příklad:

  void writeInt(int x) throws IOException
  void writeDouble(double x) throws IOException
  ...

- DataInputStream

  Typ readTyp() throws EOFException, IOException

  – ze souboru přečte hodnotu primitivního typu *Typ* a vrátí ji jako výsledek
  – výjimka *EOFException* vznikne při pokusu číst za konec proudu

  Příklad:

  int readInt() throws EOFException, IOException
  double readDouble() throws EOFException, IOException
  ...

- Při čtení souboru je třeba dodržet pořadí typů obsahu

Soubor primitivních typů a objektů

- Příklad: program, který vytvoří soubor obsahující dvě hodnoty typu *int* a dva řetězce, a pak soubor přečte a vypíše

```
public class CislaRetezce {
  public static void main(String[] ar) throws Exception {
    ObjectOutputStream out= new ObjectOutputStream(
     new FileOutputStream("x.bin"));
    out.writeInt(1);
    out.writeInt(2);
    out.writeObject("prvni retez");
    out.writeObject("druhy retez");
    out.close();

    ObjectInputStream in=new ObjectInputStream(
     new FileInputStream("x.bin"));
    System.out.println(in.readInt()+ " " +in.readInt());
    String s1 = (String)in.readObject();
    String s2 = (String)in.readObject();
    System.out.println(s1+ " " +s2);
  }}
```

Operace se souborem primitivních typů a objektů

- Pro zápis hodnot primitivních typů do proudu typu *ObjectOutputStream* slouží stejné metody jako pro *DataOutputStream*
- Pro čtení hodnot primitivních typů ze proudu typu *ObjectInputStream* slouží stejné metody jako pro *DataInputStream*

- Pro zápis objektu do souboru *ObjectOutputStream* slouží metody writeObject(Object o) a další.
- Pro čtení objektu ze souboru *ObjectInputStream* slouží readObject() a další. Ježto návratový typ metody je *Object*, je nutné znát skutečný typ přečteného objektu a dle toho vrácenou referenci přetypovat.
  - Objekty jsou čteny v témž pořadí jak byly zapsány

- Pozn.: Těmito metodami lze zapisovat a číst pouze tzv. serializovatelné objekty - patří mezi ně např. řetězce, pole primitivních typů, pole řetězců atd.

Textové soubory

Textové soubory

- Soubory určené pro ukládání čitelného textu
- Textový soubor je posloupnost znaků členěná na řádky
  - každý znak je v souboru reprezentován jedním či více byty, jejichž obsah je dán použitým kódováním znaků
  - členění na řádky je závislé na platformě a obvykle je dáno jedním nebo dvěma řídicími znaky (*CR*, *CR LF* , *0x0d 0x0c*, *"\r\n"*)
- Java však pracuje se znaky prostřednictvím primitivního typu *char*, které jsou zakódovány v 16-ti bitovém kódu Unicode (www.unicode.org). Proto jsou zapotřebí transformace mezi 16-ti bitovými hodnotami typu *char* a hodnotami znaků v textového souboru v daném kódování (např. 8b v *Cp1250*)

- Vytvoření textového souboru z posloupnosti hodnot typu *char* s implicitním kódováním umožňuje třída *FileWriter*.
- Čtení textového souboru jako posloupnosti hodnot typu *char* s implicitním kódováním umožňuje třída *FileReader*

- Pozn: Pro OS Windows 10 je implicitní kódování *UTF-8* (x *Cp1250*)

Vnitřní reprezentace x kódování

- znak "*č*"
  - vnitřní reprezentace 16b hodnotou typu *char*
    - dekadicky: 269
    - hexadecimálně: 010D
    - bitově: 0000|0001|0000|1101
  - kódování v souboru:
    - UTF8: C48D (2 bajty) 1100|0100|1000|1101
    - CP1250: E8 (1 bajt) 1110|1000

Příklad textového souboru

- Program zapíše znaky dané řetězcem a pak soubor přečte do pole prvků typu *char* a pole vypíše na obrazovku
  - soubor *text1.txt* je čitelný textovým editorem

```
package text1;
import java.io.*;

public class Text1 {
 public static void main(String[] args) throws Exception {
  FileWriter out = new FileWriter("text1.txt");
  out.write("čau, nazdar ");
  out.close();

  FileReader in = new FileReader("text1.txt");
  char znaky[] = new char[20];
  int pocet = in.read(znaky);
  for ( int i=0; i<pocet; i++ )
     System.out.print(znaky[i]);
 }}
```

Metody tříd *FileWriter* a *FileReader*

- Uveďme některé jejich metody s tím, že všechny mohou vyhodit kompilátorem kontrolovanou výjimku *IOException*

- FileWriter
  void write(int c)
    - zapíše znak c

  void write(char[] cbuf)
    - zapíše znaky z pole cbuf

  void write(String str)
    - zapíše znaky tvořící řetěz str

- FileReader
  int read()
    - vrátí přečtený znak, na konci souboru vrátí -1

  int read(char[] cbuf)
    - přečte znaky do pole cbuf a vrátí jejich počet, na konci souboru vrátí -1

Výstupní konverze

- Do textového souboru často potřebujeme zapisovat posloupnosti znaků tvořící zápisy čísel
- Řetězy tvořící dekadické zápisy čísel lze získat statickými metodami ***valueOf*** třídy ***String.*** Variantou je využití automatické konverze pomocí metody ***toString().***
- **Příklad:**

```
public class Text2 {
 public static void main(String[] args) throws Exception {
  FileWriter out = new FileWriter("text2.txt");
  int a = 10, b = 20;
  out.write(String.valueOf(a));
  out.write( "+" );
  out.write(String.valueOf(b));
  out.write( "=" );
  out.write(String.valueOf(a+b));
  out.close();
 }}
```

  - Program vytvoří textový soubor obsahující text: ***10+20=30***

Třída *PrintWriter*

- Třída ***PrintWriter*** umožňuje vytvářet textový soubor metodami, které:
  - konvertují primitivní typy do znakového tvaru
  - oddělují řádky oddělovačem dle platformy (CR LF ve Windows, CR v Unixu)
- **Příklad**:

```
public class Text3 {
 public static void main(String[] args) throws Exception {
 PrintWriter out = new PrintWriter(
          new FileWriter("text3.txt"));
  int a = 10, b = 20;
  out.print(a);
  out.print( "+" );
  out.print(b);
  out.print("=");
  out.println(a+b);
  out.println("konec programu");
  out.close();
 }}
```

  - Program vytvoří soubor obsahující text:

10+20=30
konec programu

Metody třídy PrintWriter

void print(T x)

- metoda je definovaná pro všech osm primitivních typů ***T***, do souboru zapíše textovou reprezentaci hodnoty ***x***

void print(String str)

- zapíše znaky řetězu ***str***; je-li ***str*** rovno ***null***, zapíše ***null***

void print(Object o)

- zapíše textovou reprezentaci objektu ***o***, kterou vrátí metoda ***o.toString()***; obsahuje-li o ***null***, zapíše se ***null***

void println()

- zapíše oddělovač řádků (***CR*** nebo ***CR LF*** podle platformy)

void println(T x)

- zapíše textovou reprezentaci hodnoty ***x*** a zakončí řádek

a další ...

- Pozn: Tyto metody třídy *PrintWriter* nekončí vyhozením výjimky

Čtení textového souboru po řádcích

- Třída ***FileReader*** umožňuje čtení po jednotlivých znacích nebo po skupinách znaků (ukládaných do pole znaků), oddělovač řádků je závislý na platformě.
- Čtení po řádcích nezávisle na platformě umožňuje třída ***BufferedReader***
- Příklad: program který přečte textový soubor daný parametrem z příkazového řádku a vypíše jej na obrazovku:

```
public class Text4 {
 public static void main(String[] args)throws Exception {

  if (args.length==0) {
   System.out.println("použití: Text4 <vstup>");
   return;
  }
  BufferedReader in = new BufferedReader(
    new FileReader(args[0]));
  String radek = in.readLine();
  while (radek!=null) {
     System.out.println(radek);
    radek = in.readLine();
  }}}
```

Metody třídy *BufferedReader*

int read()

- vrátí přečtený znak do proměnné typu ***int***, na konci souboru vrátí ***-1***

int read(char[] cbuf)

- přečte znaky do pole ***cbuf*** a vrátí jejich počet, na konci souboru vrátí ***-1***

int read(char[] cbuf, int off, int len)

- přečte ***len*** znaků do ***cbuf*** počínaje indexem ***off***, výsledkem je počet přečetných znaků, na konci souboru vrátí ***-1***

String readLine()

- přečte znaky až do oddělovače řádků a vytvoří z nich řetěz (bez oddělovače řádků), a ten vrátí. Na konci souboru vrátí ***null***.

a další ...

- Pozn: Všechny metody mohou vyhodit výjimku *IOException*

Čtení textu po lexikálních elementech

- Textový soubor se často skládá z lexikálních elementů tvořených posloupnostmi znaků s určitou syntaxí

- Čtení textového souboru po lexikálních elementech umožňuje třída ***StreamTokenizer***, která rozlišuje 4 druhy lexikálních elementů (*tokens*):
  - **číslo**, jako posloupnost dekadických číslic příp. začínající znaménkem a příp. obsahující desetinnou tečku
  - **slovo**, jako posloupnost písmen a číslic začínající písmenem
  - **oddělovač řádku**
  - **konec souboru**

Čtení textu po lexikálních elementech

- Příklad: čtení souboru pomocí třídy *StreamTokenizer:*

```
public class Text5 {
public static void main(String[] args) throws Exception {
   StreamTokenizer st = new StreamTokenizer(new
       FileReader("text5.txt"));
   st.eolIsSignificant(true);
   while (true) {
       switch (st.nextToken()) {
              case StreamTokenizer.TT_NUMBER:
                    System.out.print(st.nval + " ");
                    break;
              case StreamTokenizer.TT_WORD:
              System.out.print(st.sval + " ");
       break;
              case StreamTokenizer.TT_EOF:  return;
              case StreamTokenizer.TT_EOL:
                    System.out.println(); break;
}}}}
```

Změna kódování textového souboru

Příklad : převeďte textový soubor z kódování CP1250 na UTF-8

- využijeme třídy: *InputStreamReader*, *OutputStreamWriter*

v konstruktoru je navíc parametr "***encoding***"

```
FileInputStream is = new FileInputStream("in-cp1250.txt");
InputStreamReader isr = new InputStreamReader(is, "Cp1250");
System.out.println("InputStreamReader: "+isr.getEncoding());
...
FileOutputStream os = new FileOutputStream ("out.txt");
OutputStreamWriter osw = new OutputStreamWriter(os, "UTF-8");
System.out.println("OutputStreamWriter: "+ osw.getEncoding());
...
```

Změna kódování textového souboru

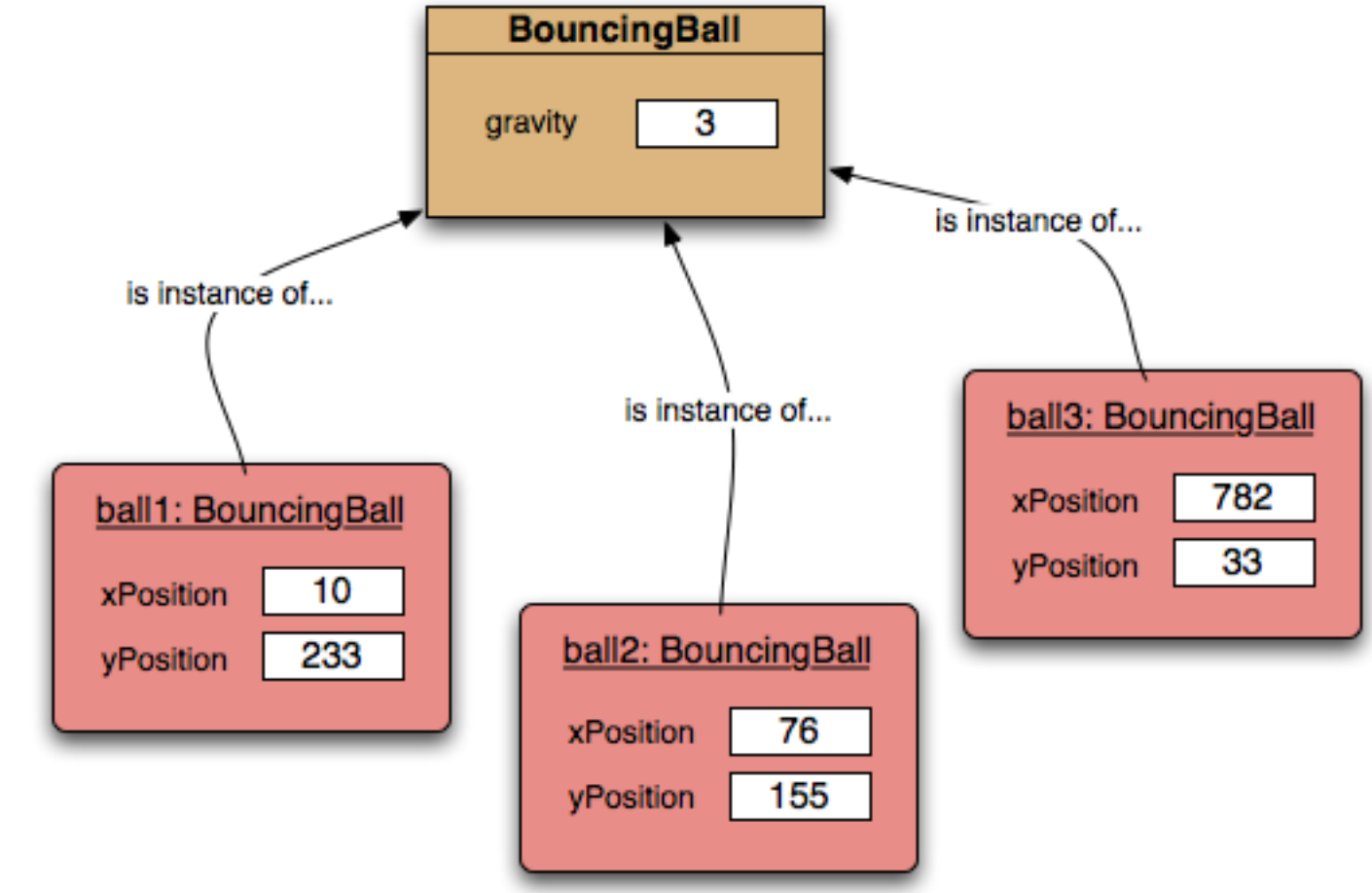
```
public class ChangeCoding {
 public static void main(String[] args) throws
 FileNotFoundException, IOException {
      FileInputStream is = new FileInputStream("in-cp1250.txt");
   InputStreamReader isr = new InputStreamReader(is, "Cp1250");
   System.out.println("InputStreamReader: "+isr.getEncoding());
   FileOutputStream os = new FileOutputStream("out.txt");
OutputStreamWriter osw = new OutputStreamWriter(os, "UTF-8");
System.out.println("OutputStreamWriter: " + osw.getEncoding());
   char[] radek = new char[100];
   int l = isr.read(radek);
   while (l > 0) {
     System.out.println("délka :" + l + " text: ");
     for (int i = 0; i < l; i++) {
        System.out.print(radek[i]);
        osw.write(radek[i]);
      }
      System.out.println();
      l = isr.read(radek);
   }
   isr.close();
   osw.close();
}}
```

Třídní proměnné a metody

Třídní proměnné

- Třídní proměnná je sdílena všemi instancemi třídy.
- Ve skutečnosti náleží třídě a existuje nezávisle na kterékoliv instanci.
- V deklaraci je označena klíčovým slovem **`static`**.
- Veřejné statické proměnné jsou přístupné prostřednictvím jména třídy.
  - **`např. Thermometer.boilingPoint`**

Třídní proměnné



Třídní metody

- Java podporuje třídní (statické) metody.
- V deklaraci se uvádí klíčové slovo **`static`**.
- Třídní metody mohou přímo přistupovat k třídním proměnným a volat přímo třídní metody.
- Třídní metody nemohou přímo přistupovat k instančním proměnným ani nemohou volat přímo instanční metody.

Konstantní proměnné

- Proměnná, která má nastavenou hodnotu, může mít tuto hodnotu fixovanou.
- Taková proměnná se v deklaraci označuje klíčovým slovem **`final`**.
  - **`final int max = list.size();`**
- Inicializace konstantní instanční proměnné se musí provést buď v deklaraci nebo v konstruktoru.

Třídní konstantní proměnné

- **`static`**: třídní proměnná
- **`final`**: konstanta

```
private static final int GRAVITY = 3;
```

- Použití klíčového slova public je v případě třídních konstant obecně neproblematické a někdy užitečné.
- Identifikátory tvořené velkými písmeny se často používají pro třídní konstanty.
  - `public static final int BOILING_POINT = 100;`

Výčtové typy

Výčtové typy

- Rys programovacího jazyka.
- K zavedení jména typu se používá `enum` místo `class`.
- Nejjednodušší použití spočívá v definici množiny platných jmen.
  - Alternativa k statickým konstantám typu int.
  - Když jsou hodnoty konstant libovolného typu.
- Použitelné při znalosti konečné množiny jmen již při překladu.

Základní výčtový typ

```
public enum CommandWord
{
    // Hodnota pro každé příkazové slovo,
    // plus jedna pro nerozpoznatelné příkazy.
    GO, QUIT, HELP, UNKNOWN;
}
```

- Každé jméno reprezentuje jeden objekt výčtového typu, např. CommandWord.HELP.
- Objekty výčtového typu nejsou přímo vytvářeny programátorem.

Základní výčtový typ

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY,
  WEDNESDAY, THURSDAY, FRIDAY,
  SATURDAY
}
```

Výčtové typy

```
public class EnumTest {
    private Day day;
    public EnumTest(Day day) {
        this.day = day;
    }

    public void tellItLikeItIs() {
        switch (day) {
                case MONDAY: System.out.println("Mondays are bad.");
                             break;

                case FRIDAY: System.out.println("Fridays are better.");
                             break;

                case SATURDAY:
                case SUNDAY: System.out.println("Weekends are best.");
                             break;

                default: System.out.println("Midweek days are so-so.");
                             break;
        }
    }
```

Výčtové typy

```
public static void main(String[] args) {
        EnumTest firstDay = new EnumTest(Day.MONDAY);
        firstDay.tellItLikeItIs();
        EnumTest thirdDay = new EnumTest(Day.WEDNESDAY);
        thirdDay.tellItLikeItIs();
        EnumTest fifthDay = new EnumTest(Day.FRIDAY);
        fifthDay.tellItLikeItIs();
        EnumTest sixthDay = new EnumTest(Day.SATURDAY);
        sixthDay.tellItLikeItIs();
        EnumTest seventhDay = new EnumTest(Day.SUNDAY);
        seventhDay.tellItLikeItIs();
}
```

Výstup

Mondays are bad.

Midweek days are so-so.

Fridays are better.

Weekends are best.

Weekends are best.

Třídy výčtového typu

- Java definuje výčtové typy jako třídy
  - mohou mít konstruktory, položky, metody
  - vytváří se jedna instance třídy pro každý prvek výčtového typu

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7);
}
```

Příklad třídy výčtového typu

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7);
-------------------------------------------------
    private final double mass;   // in kilograms
    private final double radius; // in meters

    // universal gravitational constant  (m3kg-1s-2)
    public static final double G = 6.67300E-11;

    private Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }

    private double getMass()   { return mass; }

    private double getRadius() { return radius; }

    public double surfaceGravity() {
        return G * mass / (radius * radius);
    }

    public double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
}
```

Příklad třídy výčtového typu

Poměrná váha na různých planetách

```
public class PlanetTest
{
    public static void main(String[] args)
    {
        double earthWeight = Double.parseDouble(args[0]);

        double mass = earthWeight/Planet.EARTH.surfaceGravity();

        for (Planet p : Planet.values())

                System.out.printf("Your weight on %s is %f%n",
                                  p, p.surfaceWeight(mass));
     }

}
```

Spuštění mimo vývojové prostředí a výstup programu

java PlanetTest 175 – spuštění z příkazové řádky

```
Your weight on MERCURY is 66.107583
Your weight on VENUS is 158.374842
Your weight on EARTH is 175.000000
Your weight on MARS is 66.279007
Your weight on JUPITER is 442.847567
Your weight on SATURN is 186.552719
Your weight on URANUS is 158.397260
Your weight on NEPTUNE is 199.207413
```