OOPR_02

Osnova přednášky

- Třída
- Konstruktor
- Volání konstruktorů z konstruktorů
- Kopírovací konstruktor
- Vstup dat pomocí třídy Scanner
- Vstup dat s využitím třídy JOptionPane
- Objekt, zpráva, metoda
- Třídně – instanční model OOP
- Formátovaný výstup a metoda printf
- Třída ArrayList

Třída

- Třída popisuje implementaci množiny objektů, které všechny reprezentují stejný druh systémové komponenty (složky).
- Třídy se zavádějí především z důvodů *datové abstrakce, znalostní abstrakce, efektivnímu sdílení kódu* a *zavedení taxonomie do popisu programové aplikace*.
- V systémech se třídami jsou *objekty* chápány jako *instance tříd*.
- Třída *OsobníAuto* má pak např. instance *Fabia, Seat, Audi*.

Konstruktory

- Konstruktor je speciální metoda, pro vytváření a inicializaci nových objektů (instancí).
- Název metody je totožný s názvem třídy.

```
b1 = new Bod(); // new klíčové slovo
```

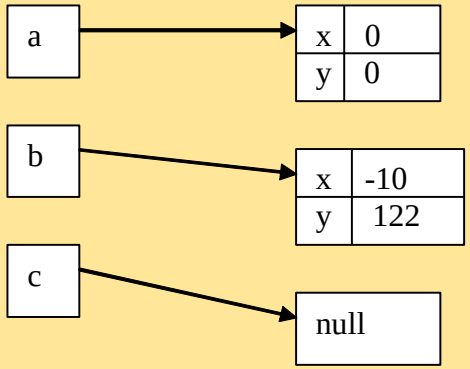
- Konstruktor vytvoří požadovanou instanci a vrátí odkaz, jehož prostřednictvím se na objekt odkazujeme.
- Konstruktor musí mít každá třída. Pokud třída nemá deklarovaný žádný konstruktor, doplní překladač nejjednodušší konstruktor (bez parametrů) a ten se označuje jako implicitní.

```
public class Point
{
   private int x;
   private int y;

   public Point() {
        x = 0; y=0;
   }

   public Point(int c) {
     x = c; y = c;
    }

    public Point(int x, int y) {
       this.x = x; this.y = y;
    }

    public Point getPoint() {
      return this;
    }

    public void setX(int x) {
      this.x = x;
     }
    public void setY(int y) {
      this.y = y;
     }
  }
```

Poznámky

Konstruktory a kopírovací konstruktor

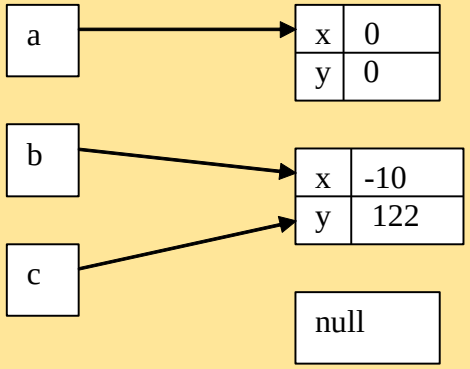
```
public class PointRun{
  public static void main(String[] args){

     Point a = new Point();
     Point b = new Point(-10, 22);
     Point c;

     . . .

  }
}
```



Poznámky

BodTest

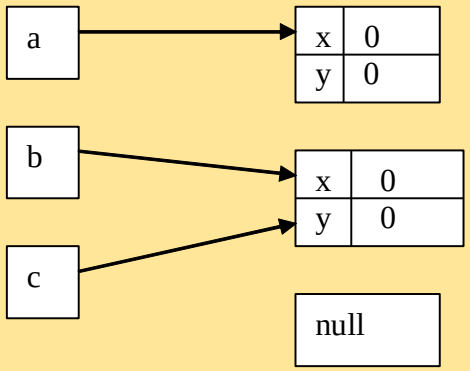
```
public class PointRun{
  public static void main(String[] args){

     Point a = new Point();
     Point b = new Point(-10, 22);
     Point c;
     . . .

     c = b;  // c = b.getPoint();

  }
}
```



Poznámky

BodTest

```
public class PointRun{
  public static void main(String[] args){

     Point a = new Point();
     Point b = new Point(-10, 22);
     Point c;
     . . .

     c = b;  //c = b.getPoint();
     . . .
     c.setX(0);
     b.setY(0);

  }
}
```



Poznámky

BodTest

Konstruktory

- Existuje-li pro třídu alespoň jeden programátorem deklarovaný konstruktor (explicitní), překladač žádný implicitní konstruktor nepřidává.
- Konstruktory dané třídy se mohou lišit počtem (typem) parametrů.
- Definice několika verzí konstruktorů s různými sadami parametrů se označuje jako přetěžování (overloading) daného konstruktoru.

Volání konstruktorů z konstruktorů

- Tvorba jedné třídy s několika konstruktory, tehdy voláme jeden konstruktor z těla jiného konstruktoru.
- používáme k tomu klíčové slovo this
- this odkazuje na „tento objekt“ – je-li za ním seznam argumentů konstruktoru, zajistí explicitní volání konstruktoru.
- Seznam volání argumentů konstruktoru a jejich pořadí se musí shodovat se seznamem a pořadím argumentů konstruktoru dané třídy.

Kopírovací konstruktor
(copy constructor)

- Potřebujeme vytvořit objekt (instanci), jehož datové atributy jsou shodné s datovými atributy argumentu konstruktoru.
- Prakticky se jedná o zkopírování datových atributů instance argumentu do daného objektu.
- Vznikne další objekt, jehož datové atributy jsou totožné s jiným objektem.

```
public Point(Point b) { };
public Square(Square square) { };
```

```
public class Point
{
   private int x;
   private int y;

   public Point() {
     this(0 , 0);
     //x = 0; y=0;
   }

   public Point(int c) {
      this(c ,c);
      //x = c; y = c;
    }

    public Point(int x, int y) {
       this.x = x; this.y = y;
    }
    // copy constructor
    public Point(Point p){
           x = p.getX();
           y = p.getY();
    }
```

Poznámky

Konstruktory a kopírovací konstruktor

```
  public void setPoint(Point a){
    x = a.getX();
    y = a.getY();
  }

  public Bod getBod(){
     return this;
   }
}
```

Poznámky

```
public class PointRun{
  public static void main(String[] args){

     Point a = new Point(12, -25);
     Point b = new Point(44);
     Point c;
     Point d = new Point(b); // copy constructor
     c = a.getBod(); // c = a;
     Point f;
     f.setPoint(b);
  }
}
```

Poznámky

BodTest

Uživatelské zadávání datových atributů

- Datové atributy se zadávají až ve třídě, která vlastní main metodu
- můžeme použít:
  - třídu *Scanner* z balíčku **java.util** – konzolový (dosovský) vstup
  - třídu *JOptionPane* z balíčku **javax.swing** – dialogová okna

```
import java.util.Scanner;
public class Scanner_M {

  public static void main(String[] args) {
          Scanner input = new Scanner(System.in);
          String text, line, buffer;
          Double db; int number;

          System.out.println(“Enter string: ");
          text = input.next();
          System.out.println(“Result: “ + text);
          buffer = input.next();
          System.out.println(“Result buffer: “ + buffer);

          line = input.nextLine();
          System.out.println(“Result: “ + line);
          System.out.println(“Enter string: ");
          text = input.next(); // just reading

          System.out.println(“Enter double: ");
          // zadava se s desetinnou carkou
          db = input.nextDouble();
          System.out.println("double number: "+db);
          System.out.println(“Enter integer: ");
          number = input.nextInt();
          System.out.println(“Entered number: " + number);

    }

}
```

Poznámky

Třída Scanner

metoda next() třídy String

metoda nextLine() třídy String

metoda nextDouble() třídy Double

metoda nextInt() třídy Integer

```
Enter string:
Adam Standa Jirka
Result: Adam
Result buffer: Standa
Result:  Jirka
Enter string:
Kamil
Enter double:
126,456
double number: 126.456
Enter integer:
55
Entered number: 55
```

Poznámky

Třída Scanner

výsledky

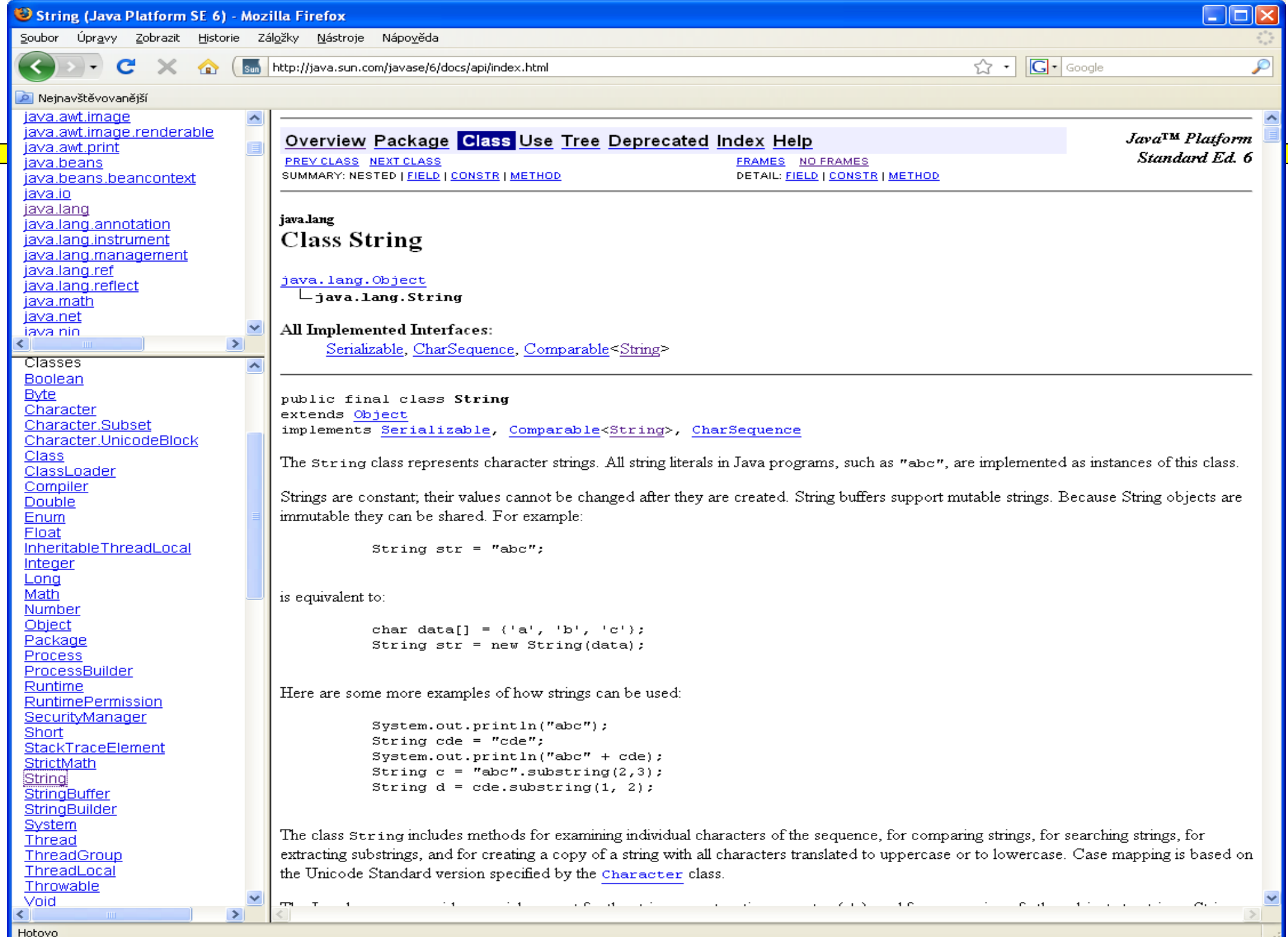
Java Class Browser

- Dokumentace v Javě
- Prohlížení javovských tříd a jejich metod a atributů
- Prohlížení on-line přes web:

  https://download.oracle.com/javase/8/docs/api/

  https://docs.oracle.com/javase/8/docs/api/

- Prohlížení lokálně vytvořeného prohlížeče


String (Java Platform SE 6) - Mozilla Firefox
Soubor Úpravy Zobrazit Historie Záložky Nástroje Nápověda
http://java.sun.com/javase/6/docs/api/index.html
Nejnavštěvovanější
java.awt.image
java.awt.image.renderable
java.awt.print
java.beans
java.beans.beancontext
java.io
java.lang
java.lang.annotation
java.lang.instrument
java.lang.management
java.lang.ref
java.lang.reflect
java.math
java.net
Classes
Boolean
Byte
Character
Character.Subset
Character.UnicodeBlock
Class
ClassLoader
Compiler
Double
Enum
Float
InheritableThreadLocal
Integer
Long
Math
Number
Object
Package
Process
ProcessBuilder
Runtime
RuntimePermission
SecurityManager
Short
StackTraceElement
StrictMath
String
StringBuffer
StringBuilder
System
Thread
ThreadGroup
ThreadLocal
Throwable
Void
Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 6
PREV CLASS NEXT CLASS
FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
java.lang
Class String
java.lang.Object
java.lang.String
All Implemented Interfaces:
Serializable, CharSequence, Comparable<String>
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.
Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:
String str = "abc";
is equivalent to:
char data[] = {'a', 'b', 'c'};
String str = new String(data);
Here are some more examples of how strings can be used:
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the Character class.
Hotovo

```
import java.util.Scanner;
public class PointRun {

   public static void main(String[] args) {
          Point a = new Point(12,4);
           a.print();
           Point b = new Point(a);
           b.print();
           b.move(-100,200);
           Point c = new Point();
           c.setPoint(a);
           c.print();
           Bod x;
           x = b.getPoint();
           x.print();
           Scanner sc = new Scanner(System.in);
           System.out.println("Coordinates of a new point: ");
           int sx = sc.nextInt();
           int sy= sc.nextInt();
           Point f = new Point(sx, sy);
         // Point f = new Point(sc.nextInt(), sc.nextInt());
           f.print();

           }

}
```

Poznámky

Třída BodTest a aplikace třídy Scanner pro vstup souřadnic bodu

Souřadnice na vstupu oddělíme mezerou, nebo Entrem

Využití třídy JOptionPane

- Třída *JOptionPane* patří do tříd GUI – Grafické uživatelské rozhraní.
- Vytváří dialogové okno, do kterého zadáváme požadované hodnoty.
- Nejvýhodněji zadat text a ten pak převádět na daný typ – možnost využití výjimek.

```
import javax.swing.JOptionPane;
public class Zkouska {
  public static void main(String[] args) {

    JOptionPane.showMessageDialog(null, “Welcome \nin \nJava");
    String name = JOptionPane.showInputDialog(“What is your name?");
    // enter the whole line
    String message =
            String.format(“My name is %s and I am glad \nto be here", name);

    JOptionPane.showMessageDialog(null, message);
    //integer
    name = JOptionPane.showInputDialog(“Enter an integer:");
    int k = Integer.parseInt(name);
    System.out.println(“Integer number: “ + k);
    String result = Integer.toString(k);
    message = String.format(“Entered number: %d is here", k);
    JOptionPane.showMessageDialog(null, message);

    //double
    name = JOptionPane.showInputDialog(“Enter a double:");
    Double db =Double.parseDouble(name);
    message = String.format("Double number: %7.2f is here", db);
    JOptionPane.showMessageDialog(null, message);
    result = Double.toString(db);

    //Boolean, Character, Long, Float, Double
   }
}
```

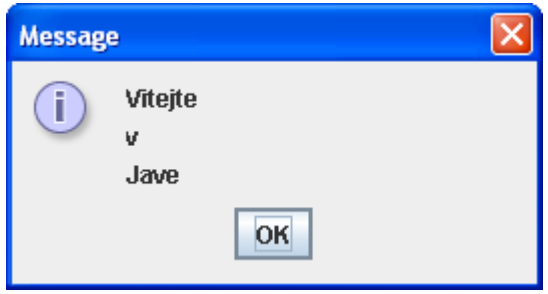
Poznámky

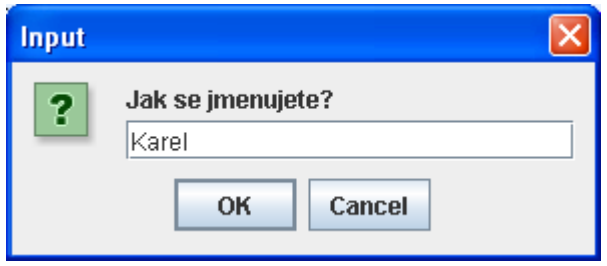
Třída JOptionPane

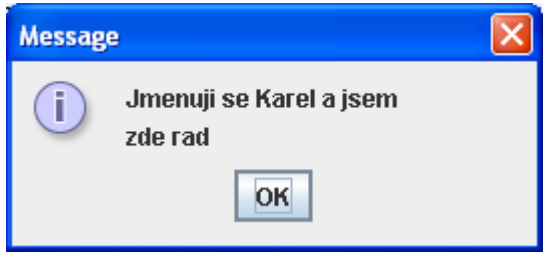
Poznámky

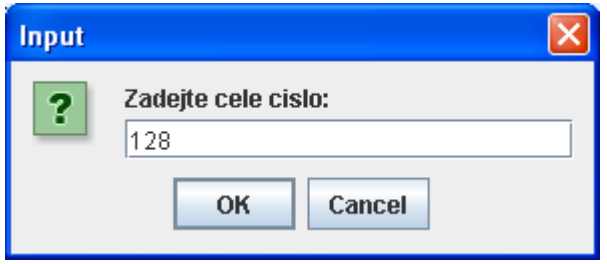
Třída JOptionPane

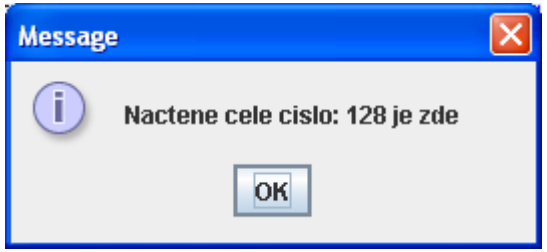
Dialogová okna zobrazovaná předchozím programem

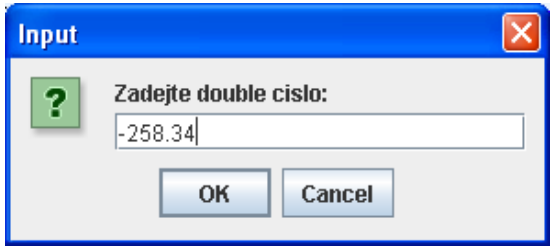


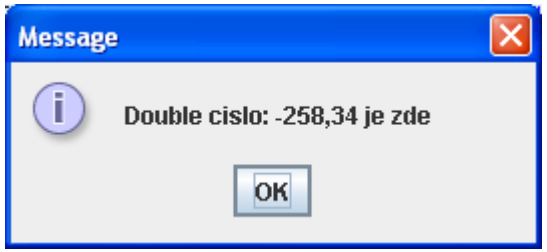




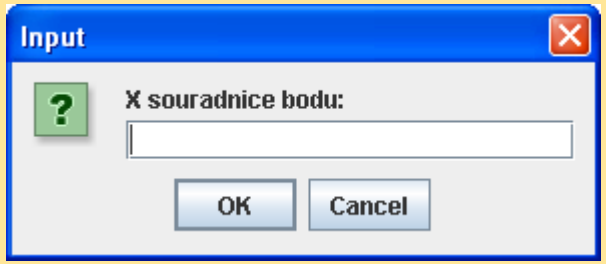








```
package input;
import javax.swing.JOptionPane;
public class BodTest {

  public static void main(String[] args) {
    String x1 = JOptionPane.showInputDialog("X coordinate: ");
    String x2 = JOptionPane.showInputDialog("Y coordinate: ");
    Point a = new Point(Integer.parseInt(x1), Integer.parseInt(x2));
    a.print();

   }
}
```



Poznámky

Třída JOptionPane

Využití dialogových oken v aplikaci

Objektově orientovaná perspektiva

- Objektově orientované programování není na rozdíl od *funkcionálního* nebo *logického* programování přímo založeno na matematické teorii, ale spíše na filozofických pojmech.
- Jádrem objektově orientovaného paradigma je programování jako *antropomorfický* pohled na objekty výpočtu.
  - antropomorfický – způsob vysvětlování na způsobu chování člověka, jeho vlastností a schopností (funkcí).

Objektově orientovaná perspektiva

- Tyto objekty mají podobně jako lidé svoji identitu, která přetrvává v čase a svoje vnitřní stavy, které se mohou měnit a schopnosti (operace).

Základní objektově orientované pojmy

- objekt (instance)
- zpráva
- metoda

Objekt (instance)

- Základem objektově orientovaného programování (OOP) je objekt.
- Je to nedělitelná sebeidentifikovatelná entita, obsahující datové atributy, jejich identifikaci a metody, které realizují příslušné operace na těchto datech.
  - objekt zapouzdření
- Objekt – stavební jednotka která modeluje nějakou část reálného nebo imaginárního světa.

Objekt

- Objekt může představovat *živou bytost* (člověk, zvíře …), *zařízení* (tiskárna, detektor pohybu…) nebo *abstraktní pojem* (stav počasí, manželství, členství, diagnóza u lékaře).
- Další příklady objektů:
  - Čísla, řetězce znaků,
  - Datové struktury: zásobník, fronta seznam, slovník,
  - Grafické obrazce, adresáře souborů, soubory,
  - Kompilátory, výpočetní procesy
  - Grafické pohledy na informace, finanční historie atd.

Objekt

- Objekty jednoho systému jsou mezi sebou propojené různými vazbami a vzájemně na sebe působí.

Zpráva

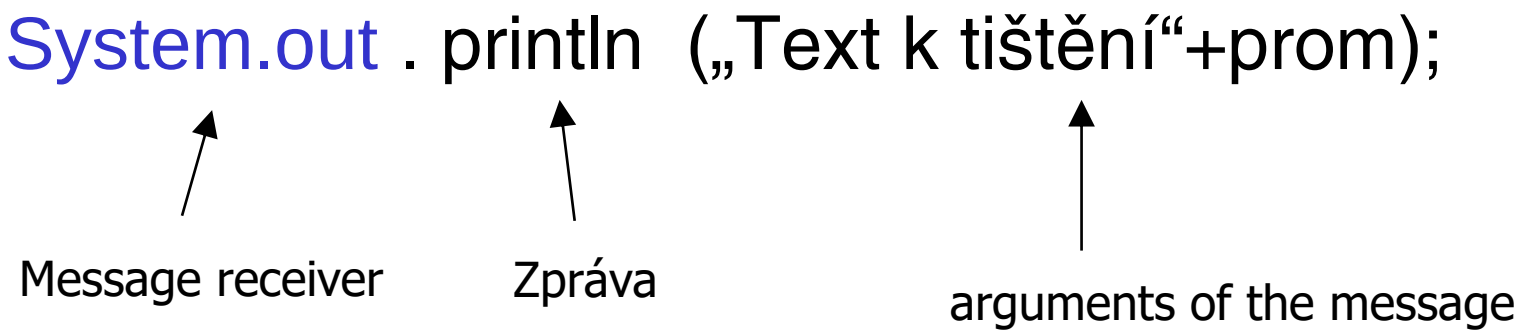
- Objekty komunikují s jinými objekty pomocí posílání zpráv.
- Množina zpráv na kterou objekt reaguje se nazývá protokol (protokol zpráv).
- Zpráva je žádost, aby objekt provedl jednu ze svých operací (metod).
- Zpráva specifikuje o jakou operaci (metodu) se jedná, ale nespecifikuje, jak by se operace měla provést.

Zpráva

- Objekt jemuž je poslána zpráva určuje, jak provést požadovanou operaci.
- Na provedení operace je nahlíženo jako na vnitřní schopnost objektu, která může být inicializovaná jedině zasláním zprávy.

Zpráva

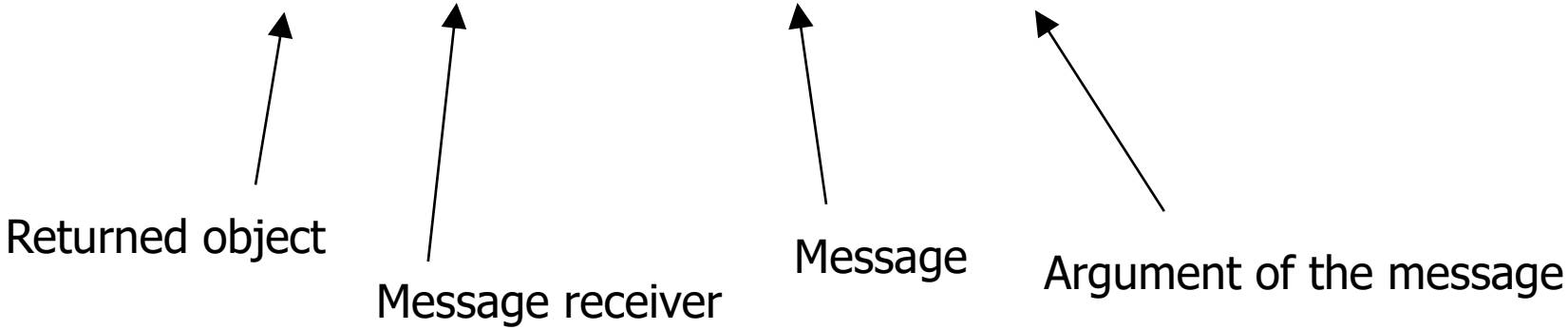
- Objektům se posílají zprávy, které se typicky skládají z adresáta neboli příjemce zprávy, selektoru zprávy (název zprávy) a eventuálních parametrů zprávy:



Zpráva

- Pokud zaslaná zpráva způsobí vyslání zpět objektu (návratové hodnoty) pak je forma zápisu:

```
Object o = ucet.mesicniVydaje(mesic);
```



Mechanismus posílání zpráv

- Objekty reagují na zprávy (požadavky), které jsou jim v čase adresované. Reakce objektů na zprávu může být:
  - *Jednoduchá odpověď* objektu (vrácení hodnoty, nebo objektu),
  - *změna vnitřního stavu* objektu,
  - *odeslání zprávy* jinému objektu,
  - *vytvoření nového* objektu,
  - *kombinace* uvedených možností.

Metoda

- Kromě datových atributů v sobě objekty uchovávají operace, které lze pomocí zaslaných zpráv spouštět.
- Takové operace se v terminologii OOP nazývají metody.
- Zpráva = *požadavek na provedení* operace, metoda = *konkrétní provedení* požadavku.
- Rozlišení mezi **zprávou** a **metodou** umožňuje realizovat tzv. polymorfismus – vícetvarost.

Model klient / server pro posílání zpráv

- OOP je založené na výpočetním modelu klient / server. (uživatel / poskytovatel)
- Např. třída String je poskytovatelem řady standardních služeb pro zpracování řetězců.
- Třída String – server, který poskytuje řetězcově orientované služby aplikacím – klientům.
- Aplikace, která využívá třídu String (její objekty) je klient, který vyžaduje služby serveru vyvoláním odpovídajících metod.

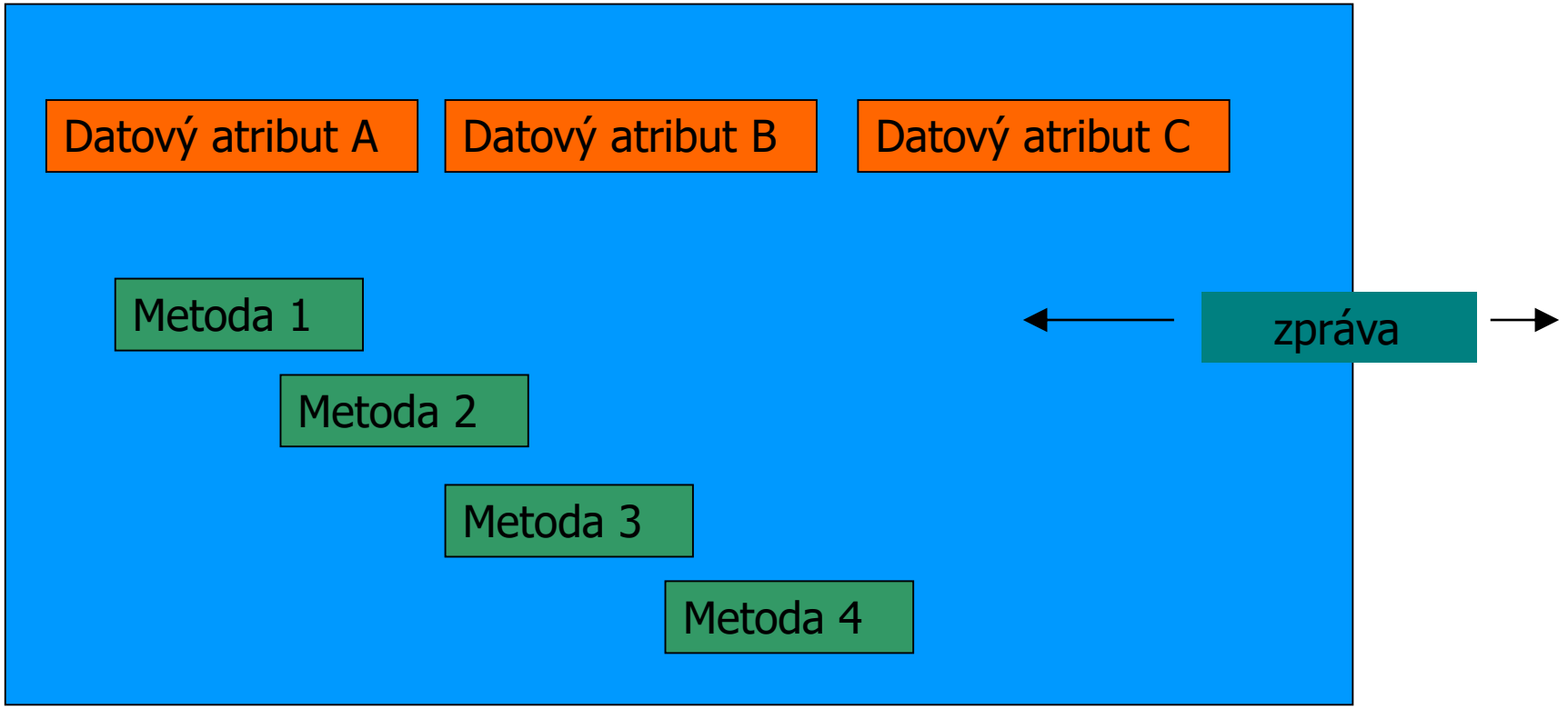
Notace zápisu

```
String s1 = “Libovolny textovy retezec”;

int n = s1.length();

s1 = s1.toLowerCase();
```

Zapouzdřenost – vlastnost objektu

- K datovým atributům objektu se dostáváme výhradně prostřednictvím metod.
  - přístupové metody get…()
  - modifikační metody set…(parametry)
  - další deklarované metody
- Odstraní se tím častá chyba, která bývá skryta ve sdíleném přístupu ke společným datům.
- S datovými atributy můžeme jen to, co nám dovolí deklarované metody.

Struktura objektu



Zapouzdřenost - encapsulation

- Výhody zapouzdření:
  - S objektem je možné pracovat pouze prostřednictvím protokolu **metod** deklarovaných ve třídě.
  - Programy mohou být testovány po menších částech.
  - Interní datová struktura může být změněna bez nutnosti změn okolí objektu (změna názvu datových atributů)
  - Mohou být vytvářeny knihovny objektů, tedy abstrakcí datových typů, které je možné použít v jiných aplikacích.
  - Zapouzdření napomáhá k oddělení rozhraní (interface) – viditelná část objektu od implementace (skrytá část deklarace objektu).

Zapouzdřenost

- Z definice objektu vyplývá, že je pro něj typické spojení dat a operací v jeden nedělitelný celek s ochranou dat, tedy zapouzdřením.
- Data jsou spojena s operacemi tak těsně, že se k nim bez těchto operací nedostaneme.
- Sebeidentifikace – objekt sám o sobě ví kdo je, takže paměť obsahující objekty obsahuje i informace o struktuře svého obsahu.

Ukrývání informací (zapouzdřenost) externí - interní

- Při externím ukrývání informací máme na mysli to, že objekty by neměly zpřístupňovat přímo svá lokální data a ani kódy jednotlivých operací. Objekt je tedy zvenku neprůhledná entita.
- Při interním ukrývání informací máme na mysli to, že při dědění nemusí mít následníci objektu přístup k lokálním datům předchůdců a také nemusí mít přístup ke kódu jednotlivých operací svých předchůdců.

Ukrývání informací externí - interní

- Objekty chápeme jako dynamické entity, které v průběhu výpočtu vznikají, vytvářejí nové objekty a zase zanikají.

Třídně instanční model

- Práce pouze se samotnými objekty by bylo velmi namáhavé a zdlouhavé jazyk Self.
- Třída popisuje implementaci množiny objektů, které všechny reprezentují stejný druh systémové komponenty.
- Třídy se zavádějí z důvodů:
  - datové abstrakce,
  - efektivnímu sdílení kódu,
  - zavedení taxonomie do popisu programové aplikace.

Třídně instanční model

- V systémech se třídami jsou *objekty* chápány jako *instance* tříd.
- Třída popisuje formu soukromých pamětí objektů a popisuje, jak se provádějí operace.
- Vztah *třída-objekt* můžeme charakterizovat jako vztah *popisu* a *realizace*.
- Programování pak sestává z vytváření tříd a vazeb mezi nimi a specifikuje sekvenci zpráv, které se vyměňují mezi objekty.

Třídně instanční model

- Objekty chápeme jako dynamické entity, které v průběhu výpočtu vznikají, vytvářejí nové objekty a zase zanikají.
- Třídy je možné vytvářet v hierarchii.
- Pro uklízení nepotřebných objektů se stará speciální program garbage collector.
- V některých OOP jazycích C++ není garbage collector.

Objektová technologie

- V objektovém modelu výpočtu pracujeme pouze se dvěma možnými operacemi s objekty.
- První z nich je pojmenování nějakého objektu – jedná se o přiřazení proměnné objektu, pomocí které je objekt dosažitelný.
- Druhou operací je tzv. posílání zprávy. Zpráva představuje žádost o provedení operace (metody) daného objektu.

Objektová technologie

- Součástí zprávy mohou být parametry zprávy, které představují *dopředný datový tok* (ve směru šíření zprávy) směrem k příjemci dané zprávy.
- Poslaná zpráva má za následek *provedení kódu* jedné z metod objektu, který zprávu přijal, tak tento zmíněný kód také většinou dává nějaký výsledek v podobě nějakých dat – objektů, které představují *zpětný datový tok* ve směru od objektu (příjemci zprávy) k objektu (vysílači zprávy).

Objektová technologie

- Vzhledem k možnostem kódů metod se výsledky po poslaných zprávách neomezují pouze na hodnoty jednotlivých atributů objektu, ale mohou to být celé *objekty*.

Objektová technologie

- Běžící objektově orientovaný program je tvořen soustavou mezi sebou *navzájem komunikujících objektů*, který je řízen především sledem vnějších událostí z rozhraní programu.
- Hlavní program může být tvořen např. pouze deklarací objektu a *zasláním zprávy* danému objektu.

Důvody přechodu na OOP

- Přechod na objektově orientované paradigma byl vyvolán nedostatkem znovupoužitelnosti a adaptability kódu programů.
  - Roste složitost programového vybavení,
  - krátká doba na realizaci projektu,
  - vysoké náklady na pracovní sílu a údržbu programového vybavení,
  - požadavky na systémy „šité na míru“,
  - inkrementální (přírůstkové) programování.

Objekt (instance) versus odkaz

- V Javě program nikdy neobdrží vytvořený objekt, ale pouze odkaz (referenci) na vytvořený objekt.
- Objekt (instance) je zřízena někde ve zvláštní paměti v haldě (heap).
- O haldu se stará správce paměti (garbage collector). Jeho funkce:
  - přidělování paměti nově vznikajícím objektům
  - rušení objektů, které nikdo nepotřebuje, (na které nejsou žádné odkazy)

Datové typy

- Na jeden objekt může být více odkazů. Zrušení objektu = zrušení všech odkazů na něj.
- Typ údaje popisuje, „co je daný údaj zač“
- V typově orientovaných jazycích mají veškerá data se kterými program pracuje svůj typ.
  - u každého údaje předem znám typ
- Výhody:
  - rychlejší práce
  - kompletnější kontrola (robustnost)
- Java rozlišuje:
  - primitivní datové typy
  - objektové datové typy

Primitivní datové typy

- Např. čísla – zabudována hluboko v jazyku, chování pevně dané; na vytvoření není třeba konstruktor tedy posílání žádných zpráv.
- int
  - definuje typ celých čísel – rozsah cca - + 2 miliardy
- boolean
  - definuje typ logických hodnot true (pravda), false (nepravda)
- double
  - označuje typ reálných čísel s přesností 15 platných číslic v rozsahu 10^{308}

Primitivní datové typy

Typ	Velikost v bitech	Zobrazená hodnota
boolean		true false - implementace závislá na JVM (Java Virtual Machina)
char	16	‘\u0000’ až ‘\uFFFF’ (0 – 65 535)
byte	8	-127 + 128
short	16	- 32 767 +32 768
int	32	-2 147 483 648 +2 147 483 647
long	64	- 9 223372 036 854 775 808 +9 223 372 036 854 775 807
float	32	záporné: -3,40 . . E+38 -1,40 . . e-45 kladné: 1,40 . . e-45 3,40 . . E+38
double	64	záporné: -1,79 . . E + 308 -4,94 . . e - 324 kladné: 4,94 . . e – 324 1,79 . . E + 308

Objektové datové typy

- Objektové datové typy = třídy
- Třídy knihoven a uživatelem definované třídy; standardní knihovna obsahuje cca 1500 tříd

- String
  - definuje typ znakových řetězců, posloupnost znaků chápána jako objekt

Vrácení hodnot primitivních typů

- int getX() objekt vrací celé číslo
- double getBalance() objekt vrací reálné číslo

- V prostředí BlueJ v inspekčním okně, se primitivní typ přímo zobrazí; je možné jeho hodnotu zkopírovat do stejného primitivního typu

Vrácení hodnot objektových typů

- K převzetí odkazu musíme mít připravený odkaz (referenci) odpovídajícího typu, (která bude vytvořena v zásobníku odkazů) a do které bude požadovaný odkaz na objekt uložen.
- Výjimkou je objektový typ (třída) String, který se někdy chová i jako primitivní typ – automaticky se v okně BlueJ zobrazí a zároveň předává odkaz.

Příkaz printf

- dostupný od verze 5
- umožňuje formátování výstupu a tím usnadňuje tisk
- parametry pro tisk:
  - formát tisku uzavřený v uvozovkách
  - vlastní konstanty nebo proměnné pro vlastní tisk

  - s symbol pro formát řetězců **%s** **%14s**
  - d symbol pro formát celých čísel **%d** **%7d**
  - f symbol pro formát čísel v pohyblivé řádové čárce
    %7.2f **%.2f**

```
System.out.printf("Name: %s year of birth: %4d weight: %7.2f",
      getName(), getYearOfBirth(), getWeight());

   Name: Jaromir year of birth: 1985 weight:   76,28


System.out.printf("%5s %s %11s %4d %5s %7.2f",
   "Name",getName(),"year of birth:", getYearOfBirth(),"weight:",
     getWeight());


   Name: Jaromir year of birth: 1985 weight:   76,28


System.out.printf("%5s*%s!%11s&%4d*%5s#%7.2f",
   "Name",getName(),"year of birth:", getYearOfBirth(),"weight:",
     getWeight());


  Name:*Jaromir!year of birth:&1985*weight:#  76,28
```

Poznámky

Příklad stejného vyjádření

```
public String toString()
   {  return String.format("%5s %s %15s %4d %5s %.3f\n",
         “Name",getName(),“year of birth:",getYearOfBirth(),
         “weight:",getweight());  }

    public void print()
    {  System.out.println(this.toString());  }
```

Poznámky

Metoda toString(), která vrací formátovaný řetězec

Využívá třídní metody format

String.format( )

```
public String toString()
        { String t = String.format("%5s %9s\n","Index",“Value");
            for (int i=0; i<=top; ++i)
              t = t+ String.format("%5d %9s\n",i, array[i]);
          return t;
  }

  public void print()
  { System.out.println(this.toString());  }
```

Poznámky

Příklad na cyklus

Příklad třídy Osoba

- Třída Osoba – datové atributy
  - jméno
  - rok narození
  - váha
- přístupové a modifikační metody
- metoda toString() s formátovaným výstupem

```
public class Person
{
   private String name;
   private int yearOfBirth;
   private double weight;

   public Person(String name) {
      this(name, 1900, 0);
     }

   public Person(String name, int year, double weight) {
      this.name = name;
      yearOfBirth = year;
      this.weight = weight;
     }

   // copy constructor
   public Person(Person ps) {
      name = ps.getName();
      yearOfBirth = ps.getYearOfBirth();
      weight = ps.getWeight();
     }

    public String getName() {
       return name;
     }
```

Poznámky

Minimálně musí uživatel zadat v konstruktoru *jméno*

```
  public void setName(String jm) { name = jm;
     }

  public int getYearOgBirth() { return yearOfBirth;
     }

  public void setYearOfBirth(int year) {
     yearOfBirth = year;
    }

  public double getWeight() { return weight;
    }

  public void setWeight(double v) {
     weight = v;
    }

 public String toString() {
   return String.format("Name: %s year of birth: %4d weight: %.2f",
              getName(), getYearOfBirth(), getWeight());
          }

  public void print() {
     System.out.println(this.toString());
   }
}
```

Poznámky

```
public class PersonRun
{
  public static void main(String args[]) {
    Person person1 = new Person("Jan",1958,78.36);
    Person person2 = new Person("Eliska",1978,66.24);
    Person person3;
    person3 = new Person(person2);
    person1.print();
    person2.print();
    person3.print();
   if(person3 == person2) System.out.println(“Variables refer to the same object");
        else System.out.println(“Variables do not refer to the same object");

    Person person4;
    person4 = person1;
    if(person4 == person1) System.out.println(“Variables refer to the same object");
        else System.out.println(" Variables do not refer to the same object ");

  }
}
```

Poznámky

```
Jmeno: Jan rok narozeni: 1958 vaha: 78,36
Jmeno: Eliska rok narozeni: 1978 vaha: 66,24
Jmeno: Eliska rok narozeni: 1978 vaha: 66,24
Neodkazuji na stejny objekt
Odkazuji na stejny objekt
```

Poznámky

```java
import java.util.ArrayList;
/**
 * A class to maintain an arbitrarily long list of notes.
  * @author David J. Barnes and Michael Kolling.
 * @version 2008.03.30
 */
public class Notebook {
    // Storage for an arbitrary number of notes.
    private ArrayList<String> notes;

    // constructor
    public Notebook() {
        notes = new ArrayList<String>();
    }
    // store a note
    public void storeNote(String note) {
        notes.add(note);
    }

    /**
     * @return The number of notes currently in the notebook.
     */
    public int numberOfNotes(){
        return notes.size();
    }
```

Poznámky

```java
/**
     * Remove a note from the notebook if it exists.
     * @param noteNumber The number of the note to be removed.
     */
    public void removeNote(int noteNumber) {
        if(noteNumber < 0) {
            // This is not a valid note number, so do nothing.
        }
        else if(noteNumber < numberOfNotes()) {
            // This is a valid note number.
            notes.remove(noteNumber);
        }
        else {
            // This is not a valid note number, so do nothing.
        }
    }
    // List all notes in the notebook.
    public void listNotes()  {
        for(String note : notes) {
            System.out.println(note);
        }
    }
```

Poznámky

```
/* alternative for cycle
   public void listNotesA() {
      for(int i = 0; i < numberOfNotes(); i++)
         System.out.println(notes.get(i));
   }
   */
   public int search(String searchString) {
      int index = 0;
      boolean found = false;
      while(index < numberOfNotes() && !found) {
         String note = notes.get(index);
         if(note.contains(searchString)) {
               found = true;
         } else {
             index++;
      }
   }
     if(!found) index = -1;
   return index;
   }
   public void removeNote(String note) {
      int index = search(note);
      if(index > -1) notes.remove(index);
      else System.out.println("String not found in the arrayList");
   }
```

Poznámky

```
    public void removeNoteObject(String note) {
        notes.remove(note);
    }

    public void replace(int index, String note) {
       if(index>= 0 && index < numberOfNotes())
           notes.set(index, note);
       else System.out.println("Index out of range");
    }
}
```

Poznámky

```
public class NoteBookRun {

   public static void main(String[] args) {
      Notebook notebook = new Notebook();
      notebook.storeNote("blue");
      notebook.storeNote("yellow");
      notebook.storeNote("black");
      notebook.storeNote("green");
      notebook.storeNote("white");
      notebook.storeNote("gray");
      notebook.storeNote("pink");
      notebook.storeNote("navy blue");
      notebook.storeNote("orange");
      notebook.storeNote("purple");
      notebook.listNotes();

      notebook.removeNote("black");
      notebook.listNotes();

      notebook.removeNoteObject("white");
      notebook.listNotes();

      notebook.replace(2, "dark gray");
      notebook.listNotes();

   }

}
```

Poznámky