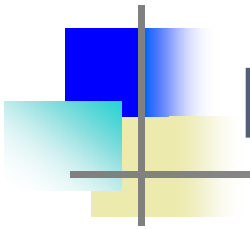
Prednáška 4:
Metódy - deklarácia, parametre metód, primitívne typy, objektové typy, modifikátory prístupu

Ján Lang

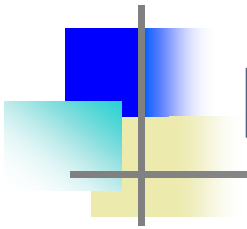
kanc. 4.34, lang@fiit.stuba.sk, http://www2.fiit.stuba.sk/~lang/zoop/

Ústav informatiky, informačných systémov a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave
10. október 2023



Rámcové zadanie

Produkty: tovary či služby



Produkty: tovary či služby

Výroba produktov je značne zložitá činnosť, ktorá vyžaduje veľa plánovania a riadenia. To je prakticky nepredstaviteľné bez zodpovedajúcej softvérovej podpory. Plánovanie a riadenie výrobných procesov je špecifický druh plánovania podnikových zdrojov. Zahŕňa smerovanie (materiálov), rozvrhovanie, dispečing a následné zhodnotenie. Mnohí výrobcovia sa usilujú byť úsporní (lean manufacturing), čo znamená nevyrábať to, čo nie je potrebné a vyrábať práve včas. Priame mapovanie entít reálneho sveta do objektov v modeli podporovanom počítačom aplikujte vo svojom programe - v jazyku Java.

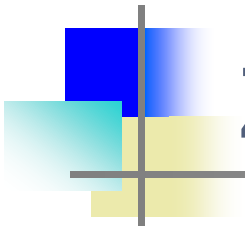
Vypracujte konkretizáciu rámcovej témy do podoby zámeru projektu súvisaceho s produktami a to tovarmi, alebo službami.

Prednáška 4:
Metódy - deklarácia, parametre metód, primitívne typy, objektové typy, modifikátory prístupu

Ján Lang

kanc. 4.34, lang@fiit.stuba.sk, http://www2.fiit.stuba.sk/~lang/zoop/

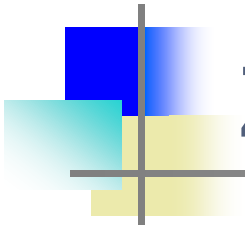
Ústav informatiky, informačných systémov a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave
10. október 2023



Základné OOP pojmy

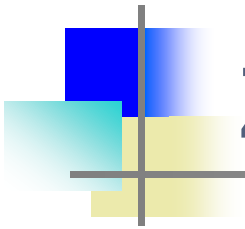
- Trieda (class) – definícia abstraktného typu dát
- Objekt (object) – inštancia triedy – implementuje stav entity, poskytuje navonok funkcionalitu prostredníctvom metód a ich implementácií, istá forma rozhrania
- Zapuzdrenie, obalenie (encapsulation)
- Preťažovanie (overloading)
- Prekonávanie (overriding)
- Dedičnosť (inheritance)
- Polymorfizmus (polymorphism)

Zapuzdrenie (encapsulácia)



- O to, ako trieda funguje, by sa mala starať len sama trieda
- Iné triedy (resp. programátori, ktorí iné triedy vytvárajú) o tom nemusia vedieť
- Fikcia „čiernej skrinky“, o ktorej je známe, čo robí ale nie, ako to robí. Skrývanie informácií
- Riešenie pomocou:
  - Modikátorov prístupu
  - Balíkov
  - Metód (ako rozhrania triedy/inštancie triedy voči vonkajšiemu svetu)
  - ...oddelenie vnútrnej reprezentácie...

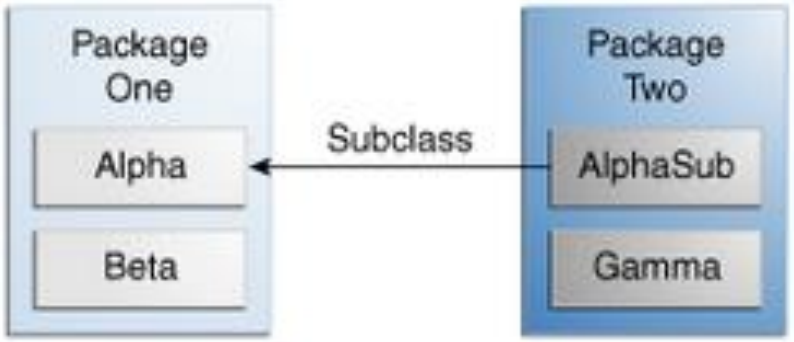
Zapuzdrenie (encapsulácia)



- O to, ako trieda funguje, by sa mala starať len sama trieda
- Iné triedy (resp. programátori, ktorí iné triedy vytvárajú) o tom nemusia vedieť
- Fikcia „čiernej skrinky“, o ktorej je známe, čo robí ale nie, ako to robí. Skrývanie informácií
- Riešenie pomocou:
  - **Modikátorov prístupu**
  - Balíkov
  - Metód (ako rozhrania triedy/inštancie triedy voči vonkajšiemu svetu)

Zapuzdrenie (encapsulácia)

- Metódy, ktoré pristupujú k atribútom triedy by mali mať názov vytvorený z názvu premennej, ku ktorej pristupujú a prefixu get alebo set, podľa toho, či vracajú jej hodnotu alebo ju menia. Tieto metódy sú slangovo nazývané gettery a settery
- Prístup k skrytým informáciám len cez poskytované rozhranie, ktoré predstavujú implementované metódy

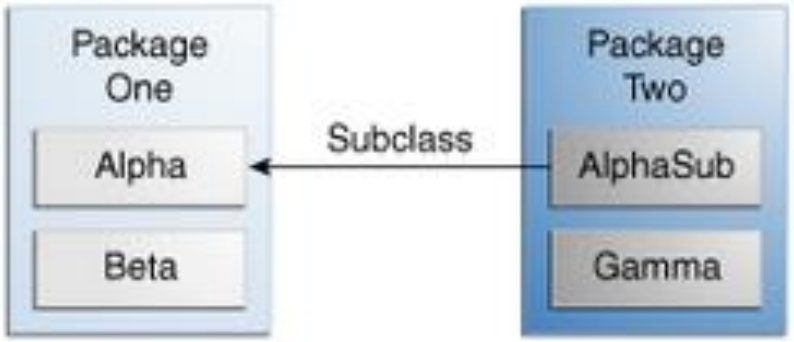


Modelovanie tried v UML

- Online app http://www.umlet.com/umletino/umletino.html
- Standalone app https://www.umlet.com/
- Plugin UMLet

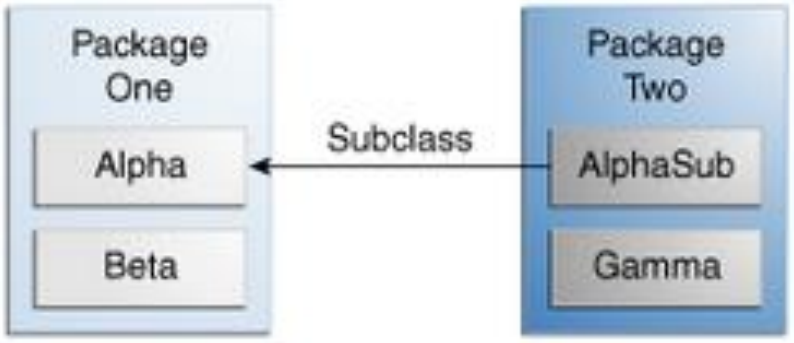
Zapuzdrenie (encapsulácia)

- Metódy, ktoré pristupujú k atribútom triedy by mali mať názov vytvorený z názvu premennej, ku ktorej pristupujú a prefixu get alebo set, podľa toho, či vracajú jej hodnotu alebo ju menia. Tieto metódy sú slangovo nazývané gettery a settery
- Prístup k skrytým informáciám len cez poskytované rozhranie, ktoré predstavujú implementované metódy



Modifikátory prístupu

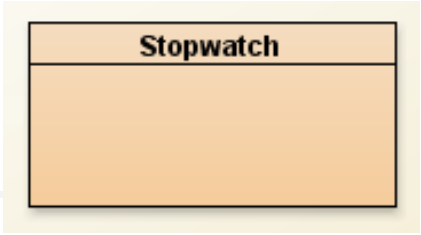
- bez uvedenia modifikátora, public, protected a private



Modifier	Alpha	Beta	Alphasub	Gamma
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

- Atribúty triedy Alpha sú v rámci triedy Alpha viditeľné s uvedením ľubovoľného modifikátora prístupu či bez neho
- Atribúty triedy Alpha sú v rámci triedy Beta viditeľné s uvedením ľubovoľného modifikátora prístupu či bez neho s výnimkou privátnych atribútov
- Atribúty triedy Alpha sú v rámci triedy Alphasub viditeľné s uvedením modifikátora `public` a `protected`

Zapuzdrenie (encapsulácia)




```
public class Stopwatch {
    private final long start;

    public Stopwatch() {
          start = System.currentTimeMillis();
    }

    // vracia čas (v sekundách) odkedy bola vytvorená
    inštancia tejto triedy
    public double elapsedTime() {
          long now = System.currentTimeMillis();
          return (now - start) / 1000.0;
    }
}

...projekt sw
```

Rekurzia

```
public static int factorial(int N) {
        if (N == 1) return 1;
        return N * factorial(N-1);
        }
```

```
public class Fibonacci {
    public static long fib(int n) {
        if (n <= 1) return n;
        else return fib(n-1) + fib(n-2);
    }
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        while(N > 0) {
            System.out.println(fib(N));
            N--;
        }
        //for (int i = 1; i <= N; i++)
            //System.out.println(i + ": " + fib(i));
    }
}
```

...konverzia zo String

Zapuzdrenie (encapsulácia)

```
public class Complex {
    private final double re;    // reálna čať
    private final double im;    // imaginárna časť

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }
}
```

Zapuzdrenie (encapsulácia)

Getery, setery

// vracajú reálnu resp. imaginárnu časť

```
public double getRe() { return re; }
public double getIm() { return im; }
```

// nastavujú reálnu resp. imaginárnu časť

```
public void setRe() {  }
public void setIm() {  }
```

Zapuzdrenie (encapsulácia)

```
public String toString() {
```
// vráti reťazecovú reprezentáciu dotknutého objektu triedy Complex

```
        if (im == 0) return re + "";
        if (re == 0) return im + "i";
        if (im <  0) return re + " - " + (-im) + "i";
        return re + " + " + im + "i";
    }

public double abs()   { return Math.hypot(re, im); }
```
//veľkosť

```
    public double phase() { return Math.atan2(im, re); }
```
// medzi -pi a pi, uhol

Zapuzdrenie (encapsulácia)

```
public Complex plus(Complex b) {     // vráti nový
  objekt triedy Complex, ktorého hodnota je (this +
  b)
        Complex a = this;                  // referencia
  na aktuálny objekt
        double real = a.re + b.re;
        double imag = a.im + b.im;
        return new Complex(real, imag);
    }
```

Zapuzdrenie (encapsulácia)

```
public Complex minus(Complex b) {  // vráti nový
  objekt triedy Complex, ktorého hodnota je (this -
  b)
        Complex a = this;
        double real = a.re - b.re;
        double imag = a.im - b.im;
        return new Complex(real, imag);
    }
```

Zapuzdrenie (encapsulácia)

```
public Complex times(Complex b) {
        Complex a = this;
        double real = a.re * b.re - a.im * b.im;
        double imag = a.re * b.im + a.im * b.re;
        return new Complex(real, imag);
    }
```

Zapuzdrenie (encapsulácia)

```
public static Complex plus(Complex a, Complex b) {
        double real = a.re + b.re;
        double imag = a.im + b.im;
        System.out.println("bola volaná metoda
 static plus");
        Complex sum = new Complex(real, imag);
        return sum;
   }
```

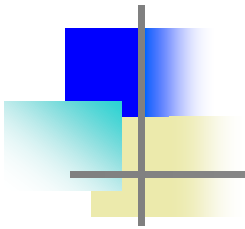
Zapuzdrenie (encapsulácia)

```
public static void main(String[] args) {
        Complex a = new Complex(4.0, 7.0);
        Complex b = new Complex(-2.0, 5.0);

        System.out.println("a                = " + a);
        System.out.println("b                = " + b);
        System.out.println("Re(a)            = " + a.re());
        System.out.println("Im(a)            = " + a.im());
        System.out.println("b + a            = " +
  b.plus(a));
        System.out.println("a - b            = " +
  a.minus(b));
        System.out.println("a * b            = " +
  a.times(b));
        System.out.println("b * a            = " +
  b.times(a));
        System.out.println("|a|              = " + a.abs());
    }
```

Zapuzdrenie (encapsulácia)

Výsledok:

```
a                = 4.0 + 7.0i
b                = -2.0 + 5.0i
Re(a)            = 4.0
Im(a)            = 7.0
b + a            = 2.0 + 12.0i
a - b            = 6.0 + 2.0i
a * b            = -43.0 + 6.0i
b * a            = -43.0 + 6.0i
|a|              = 8.06225774829855
```

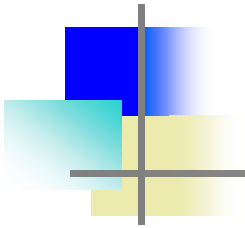


Preťažovanie metód

- Podobné ako v prípade preťažených konštruktorov...
- Preťažená metóda/konštruktor sa musí dať identifikovať (letmým pohľadom do programu) iným počtom resp. typom argumentov
- ... metódy triedy String Java 8 API

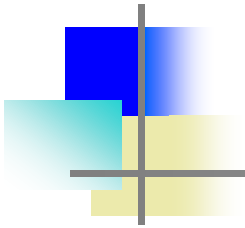
Preťažovanie metód

```
public int zakric() {
System.out.println("");
return 0;
}

public int zakric(String s) {
System.out.println(s);
return 0;
}

public int zakric(String s, int i) {
int j;

for (j = 1; j < i; j++)
System.out.println(s);
return 0;
}

public int zakric(String s, double i) {
int j;

for (j = 1; j < i; j++)
System.out.println(s);
return 0;
}
```



Preťažovanie metód

- Preťaženie zmenou počtu argumentov
- Preťaženie zmenou typu argumentov
- Preťaženie zmenou usporiadania argumentov
- Preťaženie kombináciou zmeny počtu a typu argumentov
- NESTAČÍ len zmena typu návratovej hodnoty, ani zmena z prázdneho dátového typu void na ľubovoľný



Preťažovanie metód

- Preťaženie zmenou počtu argumentov
- Preťaženie zmenou typu argumentov
- Preťaženie zmenou usporiadania argumentov
- Preťaženie kombináciou zmeny počtu a typu argumentov
- **NESTAČÍ** len zmena typu návratovej hodnoty, ani zmena z prázdneho dátového typu void na ľubovoľný

Pretypovanie, konverzie

- Konvencia: (nový typ)pôvodný typ
- rozširujúce konverzie:

  `byte->short->int->long->float->double`

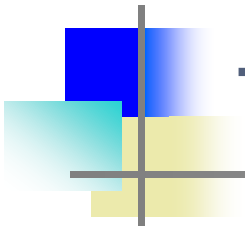
- zužujúce konverzie:

  `double->float->long->int->short->byte`

- `char c = 'B';`
- `int j = (int) c;` // rozširujúca konverzia
- `char ch = (char)j;` // zužujúca konverzia

Pretečenie

```
public class Hlavna {
public static void main(String[] args) {
    short s = 250;
    for(int i=0;i<10;i++) {
        s++;
        System.out.println("short s=" + s + " | byte
          s=" + (byte)s);
    }
    byte b = 122;
    for(int i=0;i<10;i++) {
        System.out.println("byte b=" + b++);
    }
    b = -122;
    for(int i=0;i<10;i++) {
        System.out.println("byte b=" + b--);
    }
    }
}
```



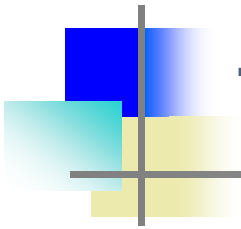
Typ String

- `java.lang` (umiestnenie v default balíku)

`public final class String` // ide o konštantu

`extends Object` //rozširuje triedu Object

- Trieda Object je koreňom v hierarchii tried. Každá trieda má triedu Object ako nadtyp. Všetky objekty implementujú metódy tejto triedy.



Typ String

```
String str = "abc";
```

je ekvivalentné k:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

Náhodný String

SecureAndRandomlyString.java | Hlavna.java

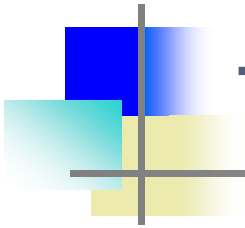
```java
package sk.stuba.fiit.randomString;

import java.security.SecureRandom;
import java.util.Random;

public class Hlavna {

    public static void main(String[] args) {
        SecureAndRandomlyString rs = new SecureAndRandomlyString();

        System.out.println(rs.rnd(new Random(), SecureAndRandomlyString.DATA, 10));
        System.out.println(rs.rnd(new SecureRandom(), SecureAndRandomlyString.DATA, 10));
        System.out.println("\u5B66");
    }
}
```

Typ String

```
public static void main(String[] args) {
    double x = Double.parseDouble(args[0]);
    double y = Double.parseDouble(args[1]);
    Charge c1 = new Charge(.51, .63, 21.3);
    Charge c2 = new Charge(.13, .94, 81.9);
    System.out.println(c1);
    System.out.println(c2);
    double v1 = c1.potentialAt(x, y);
    double v2 = c2.potentialAt(x, y);
    System.out.println(v1 + v2);
}
```

Nutné sú argumenty

```
public class Charge {
    private double rx, ry; // position
    private double q; // charge

    public Charge(double x0, double y0, double q0) {
        rx = x0;
        ry = y0;
        q = q0;
    }

    public double potentialAt(double x, double y) {
        double k = 8.99e09;
        double dx = x - rx;
        double dy = y - ry;
        return k * q / Math.sqrt(dx * dx + dy * dy);
    }

    public String toString() {
        return q + " at " + "(" + rx + ", " + ry + ")";
    }
```



Typ String

- Konštrukcia na základe argumentov príkazového riadku.
- Príklad znázorňuje vytvorenie inštancie triedy...
- Príklad znázorňuje vytvorenie viacerých inštancií triedy...
- Java API String

Typ String

Operation	Description	Invoking string s	Return value
`s.length()`	return length of s	`Hello`	`5`
`s.charAt(1)`	return character of s with index 1	`Hello`	`e`
`s.substring(1, 4)`	return substring from 1 (inclusive) to 4 (exclusive)	`Hello`	`ell`
`s.substring(1)`	return substring starting at index 1	`Hello`	`ello`
`s.toUpperCase()`	return upper case version of s	`Hello`	`HELLO`
`s.toLowerCase()`	return lower case version of s	`Hello`	`hello`
`s.startsWith("http:")`	does s start with `http:`?	`http://www.cnn.com`	`true`
`s.endsWith(".com")`	does s end with `.com`?	`http://www.cnn.com`	`true`
`s.indexOf(".java")`	return index of first occurrence of `.java` in s (-1 if no occurrence)	`Hello.java.html`	`5`
`s.indexOf(".java", 6)`	return index of first occurrence of `.java` in s, starting at index 6	`Hello.java.html`	`-1`
`s.lastIndexOf(".")`	return index of last occurrence of . in s	`Hello.java.html`	`10`
`s.trim()`	return s with leading and trailing whitespace removed	`" Hello there "`	`"Hello there"`
`s.replace(",", ".")`	return s with all occurrences of , replace by .	`13,125,555`	`13.125.555`
`s.compareTo("abc")`	compare s to `abc` lexicographically	`"abc"`	`0`

z/do String konverzie

- `char[] String.toCharArray()` //Converts this string to a new character array
- `String String.toLowerCase()` //Converts all of the characters in this String to lower case using the rules of the default locale
- `String String.toUpperCase()` //Converts all of the characters in this String to upper case using the rules of the default locale

Preťažená metóda `valueOf()`

- `valueOf(char c)`
- `valueOf(double d)`
- `valueOf(float f)`
- `valueOf(int i)`
- `valueOf(Object obj)`

z/do String konverzie

String -> int

`static Integer Integer.valueOf(String s)` //Returns an Integer object holding the value of the specified String

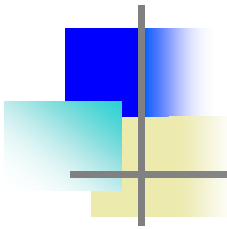
`static intparseInt(String s)` //Parses the string argument as a signed decimal integer.

- Java 8 API
- https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html
-
```
Integer i;
i = Integer.valueOf("0123");
System.out.println(++i); //124
i = Integer.parseInt("0123"); //124
```

z/do String konverzie

int -> String

- `String.valueOf(123);`
- `Integer.toString(123,10);`
- `Integer.toBinaryString(31)` //11111
- `Integer.toOctalString(15)` //17
- `Integer.toHexString(255)` // ff



z/do String konverzie

String -> double

- `Double.valueOf("3.1415")`
- `Double.parseDouble("3.1415")`

double -> String

- `String.valueOf(Math.PI)`
- `Double.toString(Math.PI)`

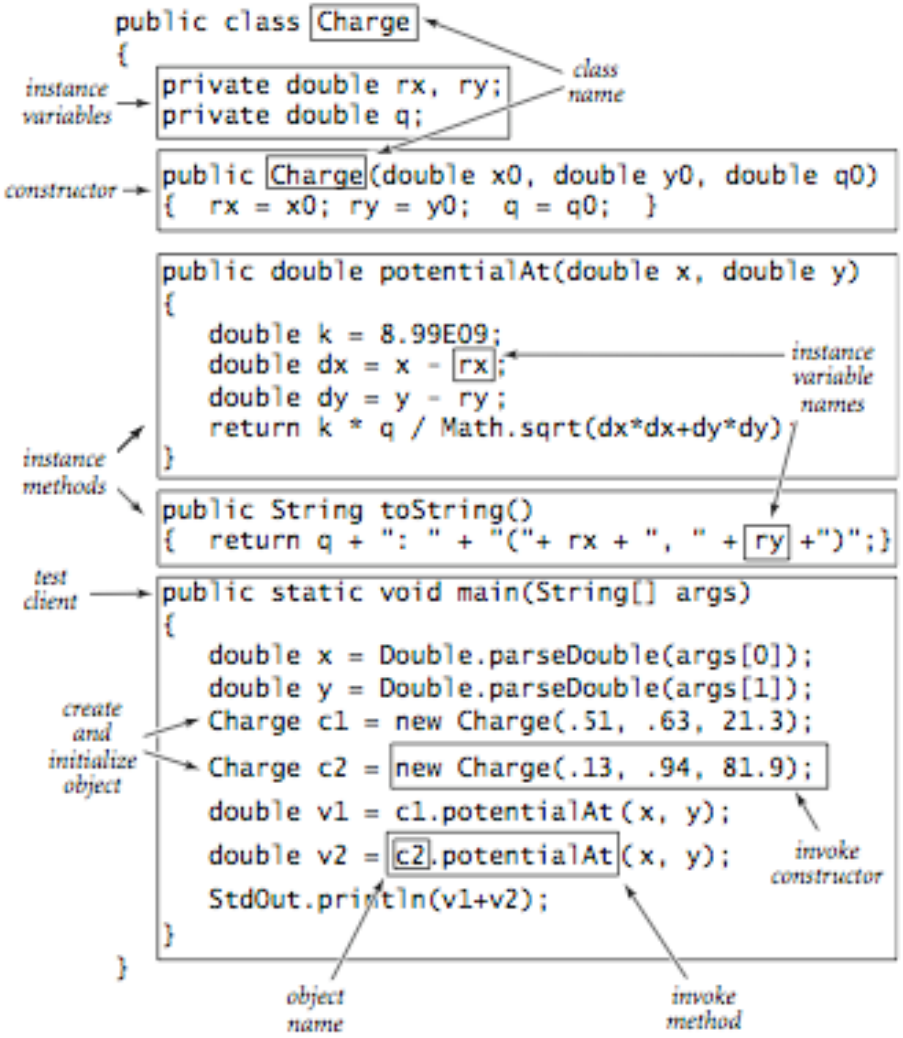
String -> Boolean

- `Boolean.valueOf("true")`
- `Boolean.parseBoolean("false")`

Boolean -> String

- `String.valueOf(true)`
- `Boolean.toString(false)`

Dátové typy - sumarizácia

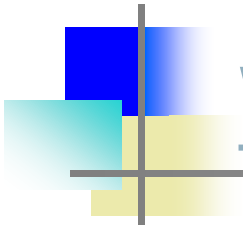


Anatomy of a class

Dátové typy - sumarizácia

- Deštruktory v Jave sú implicitné
- Povytvárané objekty v Jave netreba explicitne zrušiť
- Túto úlohu zabezpečuje garbage collector (smetiar)

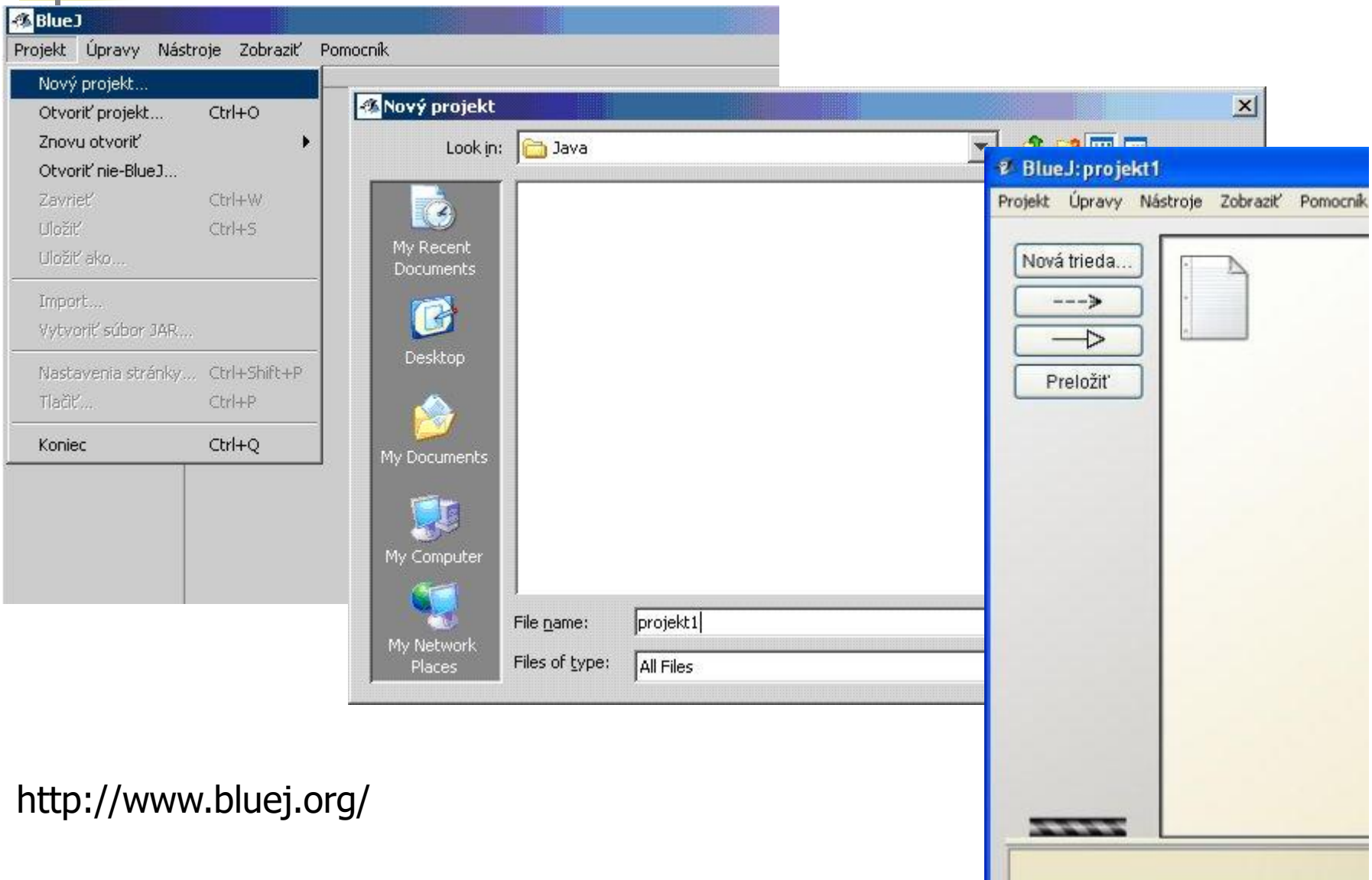
```
public void finalize() { // deštruktor triedy sa volá finalize
   System.out.println("Destruktor");
}
```



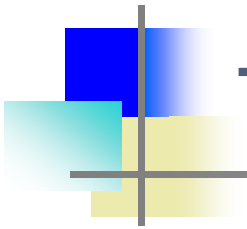
When is the finalize() method called?

- Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
- finalize() is probably not the best method to use in general unless there's something specific you need it for.

Alternatívne IDE - BlueJ



http://www.bluej.org/



TODO

- Aj vy môžete pomôcť vylepšiť tento predmet študentom pre nasledujúci akademický rok. Vaše odporúčanie, komentár či otázka.