Informatika 1

10

Viac na tému objekt

KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Pojmy zavedené v 9. prednáške(1)

- práca so súbormi
  - trieda java.io.File
  - throws java.io.IOException – povinné
- čítanie zo súboru
  - trieda java.util.Scanner
- zápis do súboru
  - trieda java.io.PrintWriter

Pojmy zavedené v 9. prednáške(2)

- trieda ako objekt
  - atribúty triedy
  - metódy triedy
  - kľúčové slovo static
- návrhový vzor Singleton
  - súkromný konštruktor

Pojmy zavedené v 9. prednáške(3)

- trieda ako množina inštancií
  - extenzia triedy
- enum
  - jednoduchý
  - s atribútmi, konštruktorom a metódami
  - enum v UML

Cieľ prednášky

- zapuzdrenie
- preťažovanie správ a metód
- konštantné atribúty
- nemeniteľné objekty

- príklad: míny

Zapuzdrenie(1)

- vnútorný vs. vonkajší pohľad
- vnútorný pohľad – prístupný len objektu samému a jeho tvorcom
  - implementácia objektu – atribúty a metódy
- vonkajší pohľad – prístupný všetkým čo objekt využívajú
  - rozhranie objektu

Zapuzdrenie(2)

- objekt je celok
  - vonkajší + vnútorný pohľad
  - atribúty – dátová časť
  - metódy – chovanie objektu
  - správy – rozhranie objektu

Zapuzdrenie(3)

- ostatní môžu objekt len žiadať o vykonanie operácie prostredníctvom posielania správ
- objekt sa sám rozhodne či a akým spôsobom správu spracuje
- objekt zverejňuje len tie informácie o svojom stave, ktoré sám uzná za vhodné

- ukrývanie informácií – ostatné objekty nemajú prístup ku dátovej zložke objektu

Modifikátory prístupu – prístupové práva

- private – vnútorný pohľad na objekt
- protected
- package
- public – vonkajší pohľad na objekt

Prístupové práva metód

- public – správa vo verejnom rozhraní objektu má priradenú danú metódu
- private – správa vo vnútornom rozhraní objektu má priradenú metódu

Prístupové práva atribútov

- private – atribút je prístupný len objektu samotnému
- public – atribút je prístupný všetkým objektom

- bodková notácia
  - nazovObjektu.nazovAtributu
  - možnosť využitia this

Prístupové práva atribútov

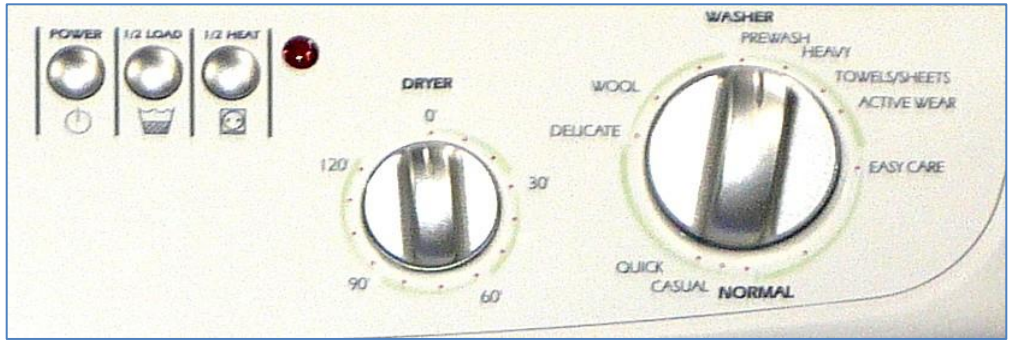
- prístup cez public porušuje zapúzdrenie
  - objekt nemá kontrolu nad svojim stavom
    - každý môže objektu zmeniť stav ľubovoľne
  - objekt zverejňuje svoj vnútorný pohľad
    - objekt sa teda nerozhoduje sám, čo zverejní, je mu to vnútené
  - ostatné objekty sú závislé na implementácii
    - implementačná závislosť

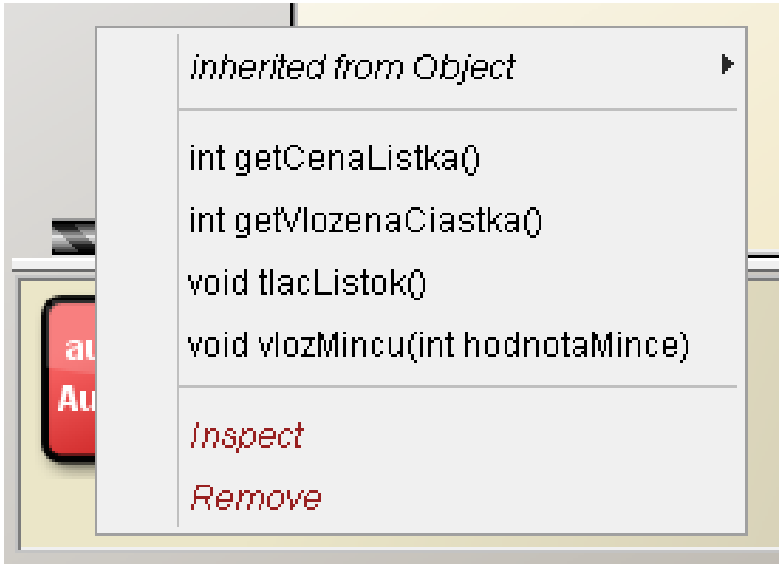
Implementačná závislosť

- iné objekty majú znalosť o implementácii daného objektu
- keď sa implementácia objektu zmení, treba zmeniť aj iné objekty
- snažíme sa minimalizovať
  - využívame len znalosť rozhrania

Forma rozhrania

- rozhranie – množina správ, ktoré je objekt schopný prijať
  - v reálnom svete – ovládací panel na rôznych prístrojoch, manuály...
  - v prostredí BlueJ – správy v menu





Dokumentácia – forma rozhrania

- dokumentácia objektu – verejné rozhranie

Class AutomatMHD

java.lang.Object
  └ **AutomatMHD**

```
public class AutomatMHD
extends Object
```

Trieda modeluje primitivny automat na predaj cestovnych listkov MHD. Model predpoklada, ze kupujuci vlozi presnu ciastku podla ceny listka. Cena listka je urcena parametrom konstruktora.

Version:
2009.10.30

Author:
David J. Barnes and Michael Kolling

Constructor Summary

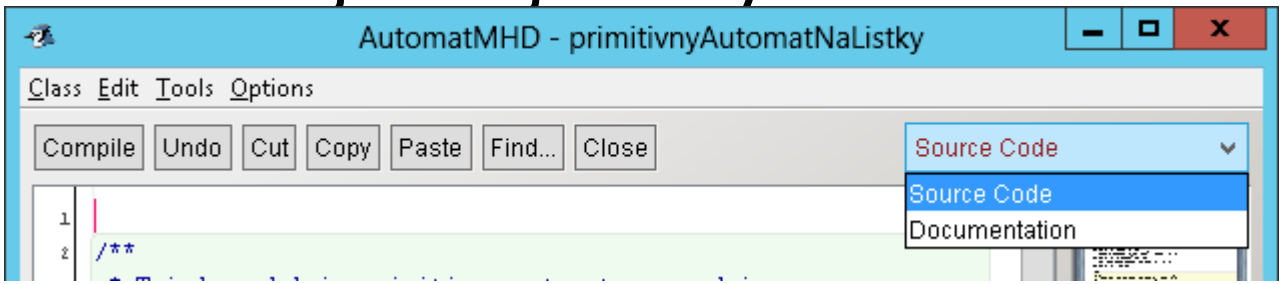
AutomatMHD(int cenaListka)	Konstruktor vytvori automat, ktory bude tlacit cestovne listky pevnej ceny.

Method Summary

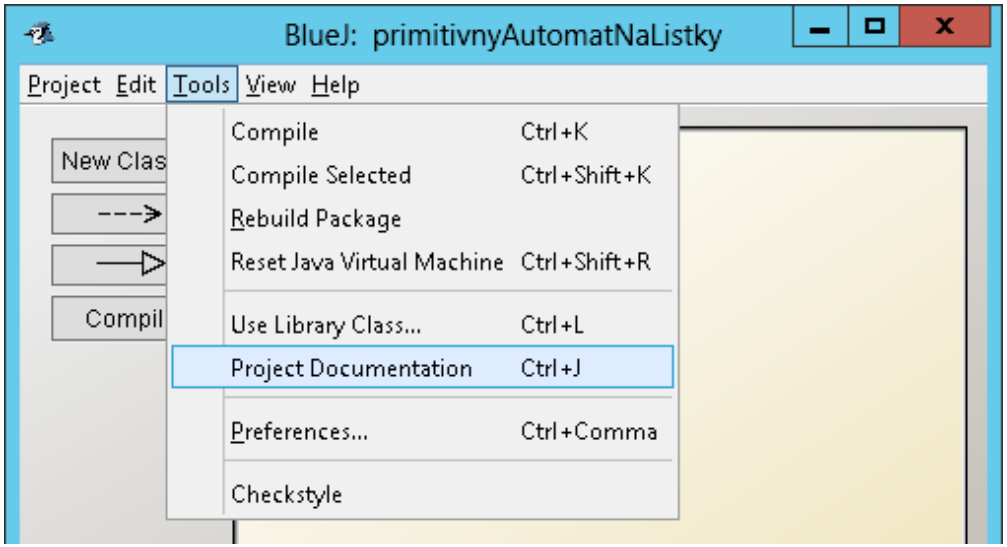
int	**getCenaListka**() Vrati hodnotu ceny listka
int	**getVlozenaCiastka**() Vrati doteraz vlozenu ciastku
void	**tlacListok**() Vytlaci cestovny listok, pripocita vlozenu ciastku k trzbe a vynuluje vlozenu ciastku
void	**vlozMincu**(int hodnotaMince) Prijme mincu danej hodnoty od kupujuceho

BlueJ – generovanie dokumentácie

- dokumentácia jednej triedy – editor



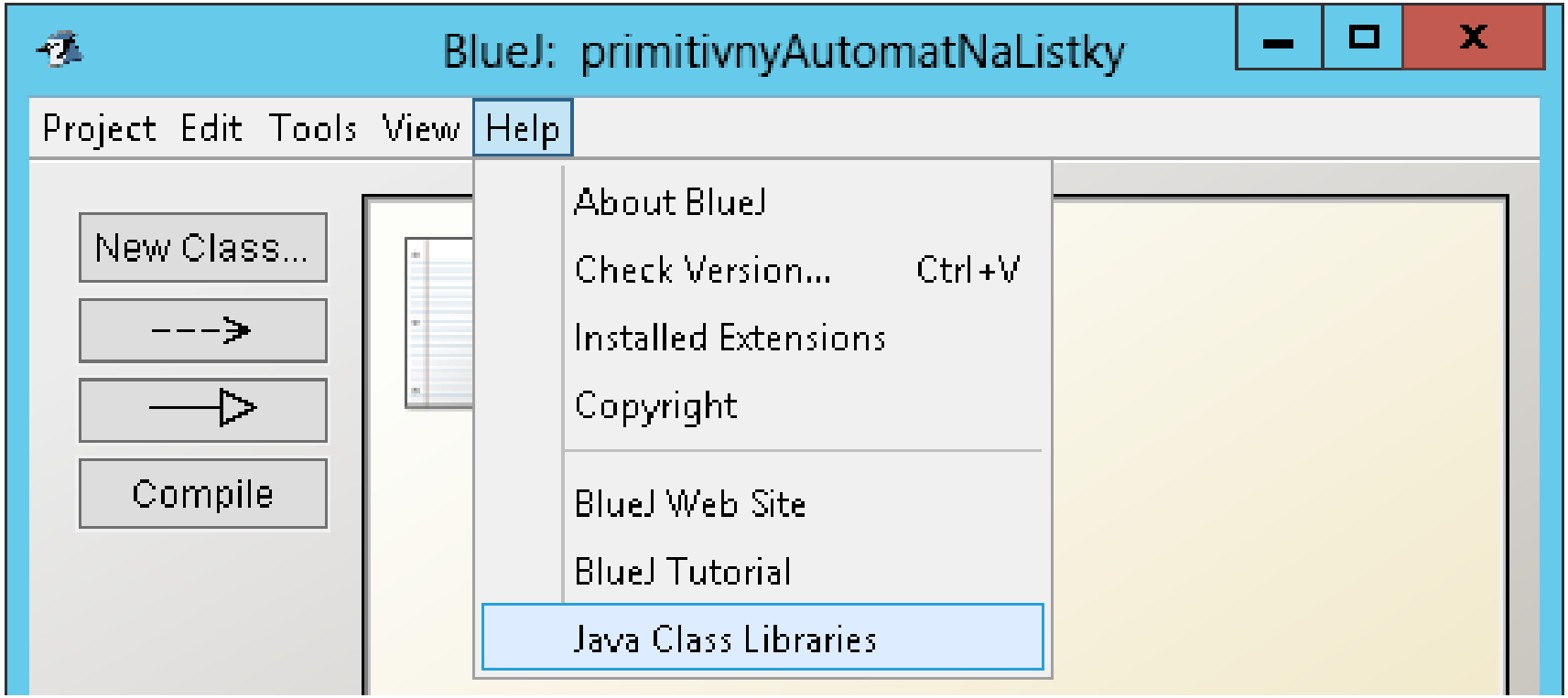
- dokumentácia projektu – prostredie
  - menu Tools>Project Documentation



Dokumentácia sa generuje do podadresára doc v adresári projektu

BlueJ – štandardná knižnica

- dokumentácia štandardnej knižnice – prostredie
  - Help>Java Class Libraries



Javadoc

- generátor dokumentácie objektu
- špeciálne komentáre – dokumentačné komentáre

- krátky popis nasledovaný tagmi

Javadoc – trieda – zdrojový kód

```
/**
 * Trieda modeluje primitivny automat na predaj
 * cestovnych listkov MHD.
 * Model predpoklada, ze kupujuci vlozi presnu
 * ciastku podla ceny listka.
 * Cena listka je urcena parametrom konstruktora.
 *
 * @version 2009.10.30
 * @author David J. Barnes and Michael Kolling
 */
public class AutomatMHD {
```

Javadoc – trieda – dokumentácia

```
public class AutomatMHD
extends Object
```

Trieda modeluje primitivny automat na predaj cestovnych listkov MHD. Model predpoklada, ze kupujuci vlozi presnu ciastku podla ceny listka. Cena listka je urcena parametrom konstruktora.

Version:

2009.10.30

Author:

David J. Barnes and Michael Kolling

Javadoc – konštruktor – zdrojový kód

```
/**
 * Konstruktor vytvori automat, ktory bude
 * tlacit cestovne listky pevnej ceny.
 * Cena je urcena parametrom cenaListka.
 * Pozor - cena listka musi byt kladne cele
 * cislo a tato podmienka sa nekontroluje.
 *
 * @param cenaListka hodnota ceny listka v centoch
 */
public AutomatMHD(int cenaListka) {
```

Javadoc – konštruktor – dokumentácia

Constructor Summary

AutomatMHD(int cenaListka)
Konstruktor vytvori automat, ktory bude tlacit cestovne listky pevnej ceny.

Constructor Detail

AutomatMHD

```
public AutomatMHD(int cenaListka)
```

Konstruktor vytvori automat, ktory bude tlacit cestovne listky pevnej ceny. Cena je urcena parametrom cenaListka. Pozor - cena listka musi byt kladne cele cislo a tato podmienka sa nekontroluje.

Parameters:

cenaListka - hodnota ceny listka v centoch

Javadoc – metóda – zdrojový kód

```
/**
 * Vrati hodnotu ceny listka
 *
 * @return hodnota ceny listka v centoch
 */
public int getCenaListka()
```

Javadoc – metóda – dokumentácia

Method Summary	
int	**getCenaListka**() Vrati hodnotu ceny listka
int	**getVlozenaCiastka**() Vrati doteraz vlozenu ciastku
void	**tlacListok**() Vytlaci cestovny listok, pripocita vlozenu ciastku k trzbe a vynuluje vlozenu ciastku
void	**vlozMincu**(int hodnotaMince) Prijme mincu danej hodnoty od kupujuceho

Method Detail

getCenaListka

```
public int getCenaListka()
```

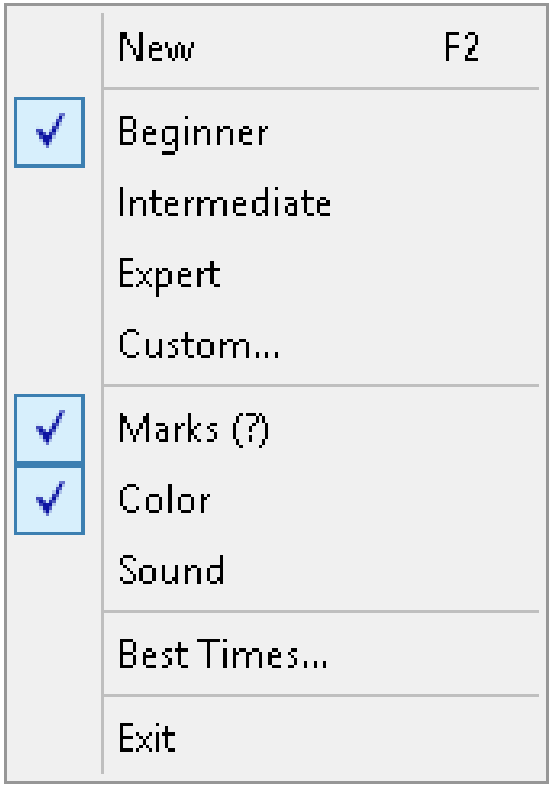
Vrati hodnotu ceny listka

Returns:

hodnota ceny listka v centoch

Obtiažnosť hry

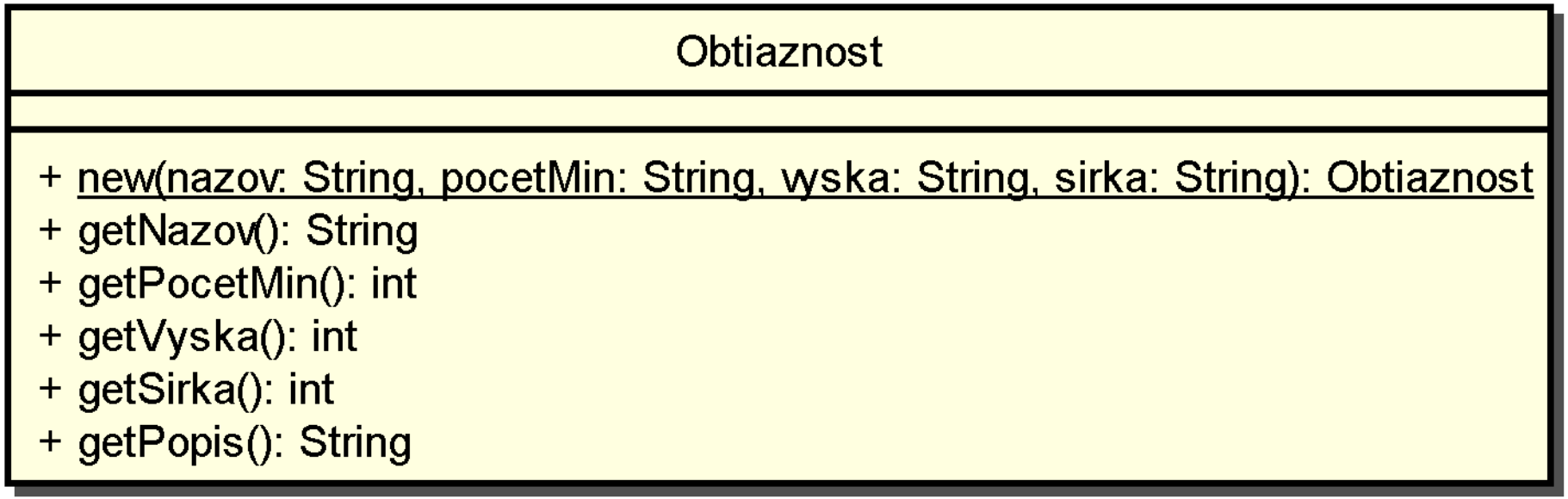
- obtiažnosť:
  - počet mín na hracom poli
  - rozmery hracieho poľa (riadky x stĺpce)
- obtiažnosti hry:
  - ľahká (10 mín, pole 9x9)
  - stredná (40 mín, pole 16x16)
  - ťažká (99 mín, pole 16x30)
  - vlastná – definuje hráč



Trieda Obtiaznost

- inštancie uchovávajú informácie o obtiažnosti:
  - názov
  - počet mín
  - počet riadkov
  - počet stĺpcov
- informácie o konkrétnej obtiažnosti sa nesmú meniť – ľahká obtiažnosť je vždy „10 mín, 9x9“
  - nesmie mať zmenové metódy
  - informácie sa nastavujú pri vzniku

Trieda Obtiaznost – rozhranie



Nemeniteľné objekty

- trvalý stav objektu definovaný na začiatku životného cyklu
- neexistuje spôsob, ako stav zmeniť

- takéto objekty nazývame nemeniteľné – immutable

Nemeniteľné objekty – príklady

- štandardná knižnica jazyka Java
  - String
  - obaľovacie triedy (Integer, Double...)
  - ...
- použité v projektoch
  - Datum
  - Cas
  - Obtiaznost
  - ...

Štandardné obtiažnosti

- aplikácia má pevne dané tri obtiažnosti:
  - Ľahká
  - Stredná
  - Ťažká
- ich inštancie vznikajú pri štarte aplikácie – nemenia sa

Definícia štandardných obtiažností

```
public class Aplikacia {
    private static final Obtiaznost LAHKA
                        = new Obtiaznost("Lahka", 10, 9, 9);
    private static final Obtiaznost STREDNA
                    = new Obtiaznost("Stredna", 40, 16, 16);
    private static final Obtiaznost TAZKA
                    = new Obtiaznost("Tazka", 99, 16, 30);
    ...
}
```

Konštantné atribúty

- každý objekt (trieda i inštancia) môže mať niektoré atribúty označené ako konštantné
- musia byť inicializované na začiatku životného cyklu objektu
- nedajú sa meniť v priebehu života objektu
- v jazyku Java označené kľúčovým slovom final

Konštantné atribúty triedy

- obvykle sú inicializované v definícii atribútu
- väčšinou konštanta prístupná počas celého behu aplikácie – pomenovaná konštanta
- konvencia – názov veľkými písmenami – oddeľovač slov je podčiarkovník

```
private static final double PI = 3.1415926539;
```

```
private static final int VELKOST_STRANY = 5;
```

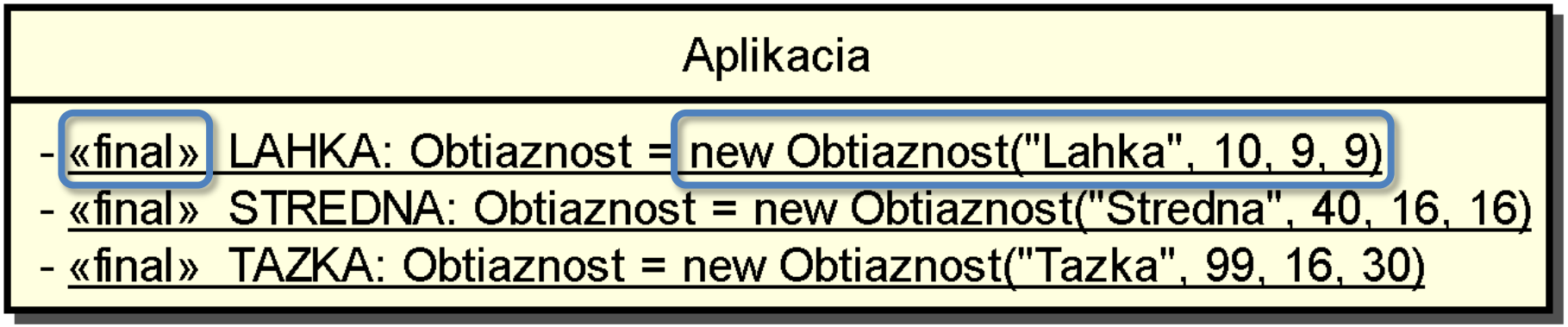
```
private static final Obtiaznost LAHKA_OBTIAZNOST
                    = new Obtiaznost("Lahka", 10, 9, 9);
```

Konštantné atribúty inštancie

- Reprezentujú konštantnú časť stavu

- z reálneho sveta:
  - uhlopriečka televízora
  - rozmery práčky
  - výrobné číslo motora
  - ...
- Java – doteraz sme sa stretli:
  - atribút length poľa
  - atribút out triedy System

Konštantné atribúty v UML



Metóda Aplikacia.nastavObtiaznost

```
public void nastavObtiaznost(String nazov) {
    if (LAHKA.getNazov().equals(nazov)) {
        this.obtiaznost = Aplikacia.LAHKA;
    } else if (STREDNA.getNazov().equals(nazov)) {
        this.obtiaznost = Aplikacia.STREDNA;
    } else if (TAZKA.getNazov().equals(nazov)) {
        this.obtiaznost = Aplikacia.TAZKA;
    }
}
```

Nastavenie vlastnej obtiažnosti

- používateľ definuje tri položky:
  - počet mín
  - počet riadkov
  - počet stĺpcov
- názov je vždy „vlastna“

- obtiažnosť je nemeniteľná – pri každom nastavení vlastnej obtiažnosti treba vytvárať inštanciu triedy Obtiaznost

Metóda Aplikacia.nastavObtiaznost

```
public void nastavObtiaznost(int pocetMin,
                             int vyska, int sirka) {
   this.obtiaznost = new Obtiaznost(
                        "Vlastna", pocetMin, vyska, sirka);
}
```

Preťažovanie správ a metód(1)

- dve správy s rovnakým selektorom:
  - nastavObtiaznost(String nazov)
  - nastavObtiaznost(int pocetMin, int vyska, int sirka)
- odlišnosť – počet a typ parametrov

Preťažovanie správ a metód(2)

- zjednodušenie – identifikátor správy si vyjadríme ako:
  - selektor#typParametra1#typParametra2#...
  - typParametra = typ skutočného parametra

- napr:
  - nastavObtiaznost("Lahka")
    => nastavObtiaznost#String
  - nastavObtiaznost(5, 10, 10)
    => nastavObtiaznost#int#int#int

Preťažovanie správ a metód(3)

- identifikátor metódy si vyjadrime ako:
  - nazovMetody#typParametra1#typParametra2#...
  - typParametra = typ formálneho parametra

- napr:
  - public void nastavObtiaznost(String nazov)
    => nastavObtiaznost#String
  - public void nastavObtiaznost(int m, int r, int s)
    => nastavObtiaznost#int#int#int

Preťažovanie správ a metód(4)

- príslušná metóda sa vyhľadáva na základe zhody identifikátora správy a identifikátora metódy
  => Protokol

Preťažovanie konštruktora

- rovnaký princíp funguje aj pre konštruktor a správu new

- napr:
  - new Obtiaznost("Lahka", 10, 9, 9)
    => new#String#int#int#int
  - public Obtiaznost(String n, int m, int s, int r)
    => new#String#int#int#int

- tento princíp umožňuje mať v triede definovaných viac konštruktorov

Vďaka za pozornosť