

# 10

## Pokročilé vlastnosti objektov

# Pojmy zavedené v 9. prednáške<sub>(1)</sub>

- 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<sub>(2)</sub>

- metóda toString
  - automatická konverzia na reťazec
- generická trieda ako prvok poľa v jazyku Java

# Pojmy zavedené v 9. prednáške<sub>(3)</sub>

- trieda `HashSet<TPrvok>`
  - úpravy (pridávanie, odoberanie)
  - zisťovanie, či je objekt prvkom množiny
  - zisťovanie, či je množina prázdna
  - zisťovanie, či je množina A podmnožinou B
  - zjednotenie množín A a B
  - prienik množín A a B
  - rozdiel množín A a B

# Cieľ prednášky

- zapuzdrenie
- trieda ako objekt
- preťažovanie správ a metód
- konštantné atribúty
- nemeniteľné objekty
  
- príklad: míny

# Zapuzdrenie<sub>(1)</sub>

- 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<sub>(2)</sub>

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

# Zapuzdrenie<sub>(3)</sub>

- 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

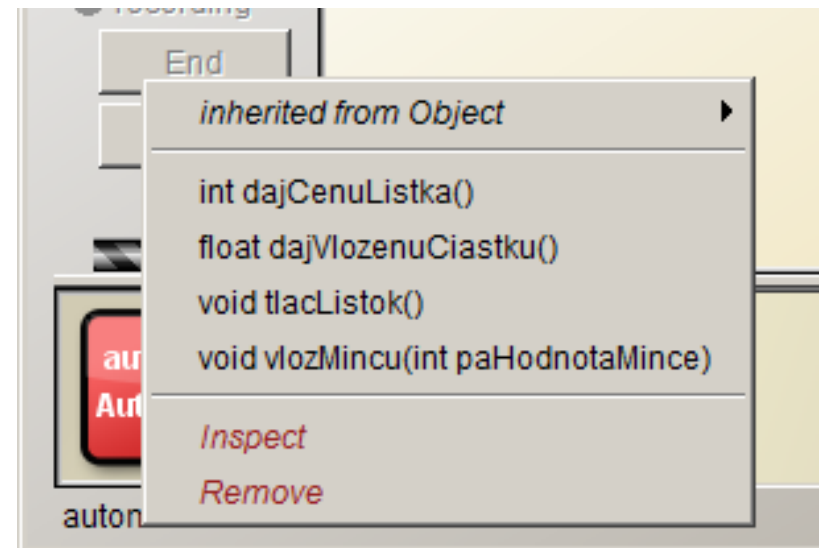
- prístup cez public porušuje zapúzdrenie
  - objekt nemá kontrolu nad svojim stavom
    - každý môže objektu zmeniť stav ľubovoľne
  - objekt zverejňuje svoj vnútorný pohľad
    - objekt sa teda nerozhoduje sám, čo zverejní, je mu to vnútené
  - ostatné objekty sú závislé na implementácii
    - implementačná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

```
public class AutomatMHD extends java.lang.Object
```

Trieda modeluje primitívny automat na predaj cestovných lístkov MHD. Model predpokladá, že kupujúci vloží presnú čiastku podľa ceny lístka. Cena lístka je určená parametrom konštruktora.

## Constructor Summary

[AutomatMHD](#)(int paCenaListka)

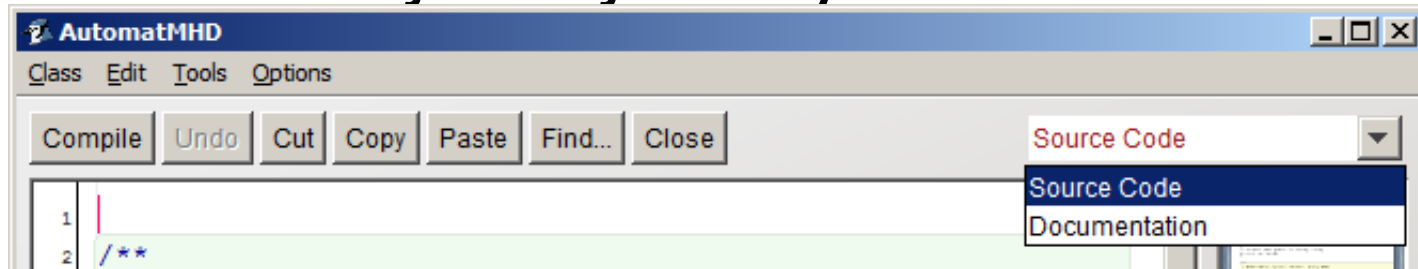
Konštruktor vytvorí automat, ktorý bude tlačiť cestovné lístky pevnej ceny.

## Method Summary

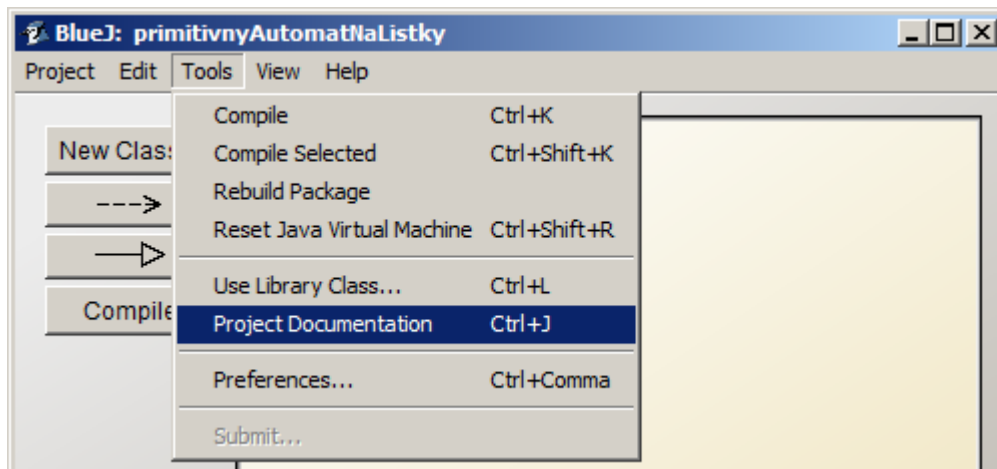
int	<a href="#">dajCenuListka</a> () Vráti hodnotu ceny lístka
float	<a href="#">dajVlozenuCiastku</a> () Vráti doteraz vloženú čiastku
void	<a href="#">tlacListok</a> () Vytlačí cestovný lístok, pripočíta vloženú čiastku k tržbe a vynuluje vloženú čiastku
void	<a href="#">vlozMincu</a> (int paHodnotaMince) Prijme mincu danej hodnoty od kupujúceho

# BlueJ – generovanie dokumentácie

- dokumentácia jednej triedy – editor



- dokumentácia projektu – prostredie  
– menu Tools>Project Documentation

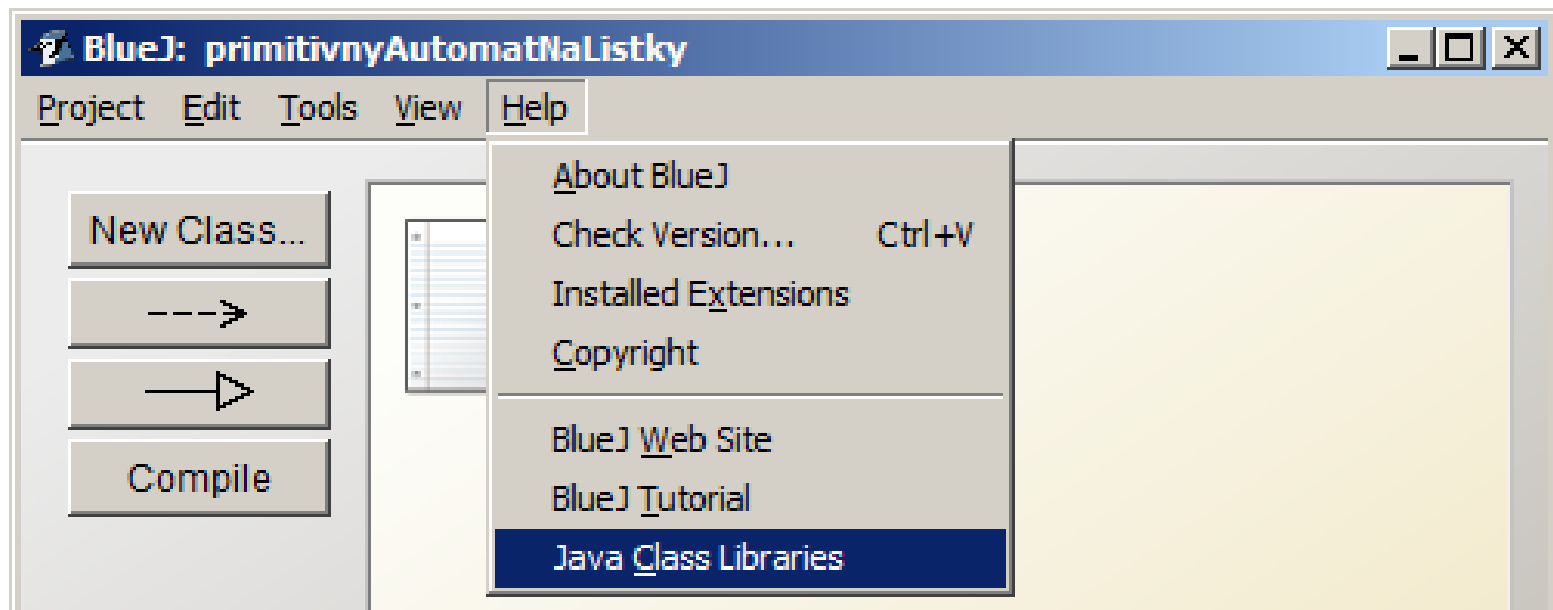


Dokumentácia sa generuje do podadresára doc v adresári projektu



# BlueJ – štandardná knižnica

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



- generátor dokumentácie objektu
- špeciálne komentáre – dokumentačné komentáre
- krátke popisy nasledované 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 AutomatMHDExtends java.lang.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

```
/**  
 * Konštruktor vytvorí automat, ktorý bude  
 * tlačit cestovné listky pevnej ceny.  
 * Cena je určená parametrom paCenaListka.  
 * Pozor - cena listka musí byť kladné celé  
 * číslo a táto podmienka sa nekontroluje.  
 *  
 * @param paCenaListka hodnota ceny listka  
 */
```

```
public AutomatMHD(int paCenaListka)
```

# Javadoc – konštruktor – dokumentácia

## Constructor Summary

[AutomatMHD](#)(int paCenaListka)

Konstruktor vytvori automat, ktorý bude tlačiť cestovné listky pevnej ceny.

## Constructor Detail

### AutomatMHD

```
public AutomatMHD(int paCenaListka)
```

Konstruktor vytvori automat, ktorý bude tlačiť cestovné listky pevnej ceny. Cena je určená parametrom paCenaListka. Pozor - cena listka musí byť kladné celé číslo a táto podmienka sa nekontroluje.

#### Parameters:

paCenaListka - hodnota ceny listka

# Javadoc – metóda – zdrojový kód

```
/**  
 * Vrací hodnotu ceny listka  
 *  
 * @return hodnota ceny listka  
 */
```

```
public int dajCenuListka()
```

# Javadoc – metóda – dokumentácia

## Method Summary

int	<a href="#">dajCenuListka()</a> Vrati hodnotu ceny listka
float	<a href="#">dajVlozenuCiastku()</a> Vrati doteraz vlozenu ciastku
void	<a href="#">tlacListok()</a>

### dajCenuListka

```
public int dajCenuListka()
```

Vrati hodnotu ceny listka

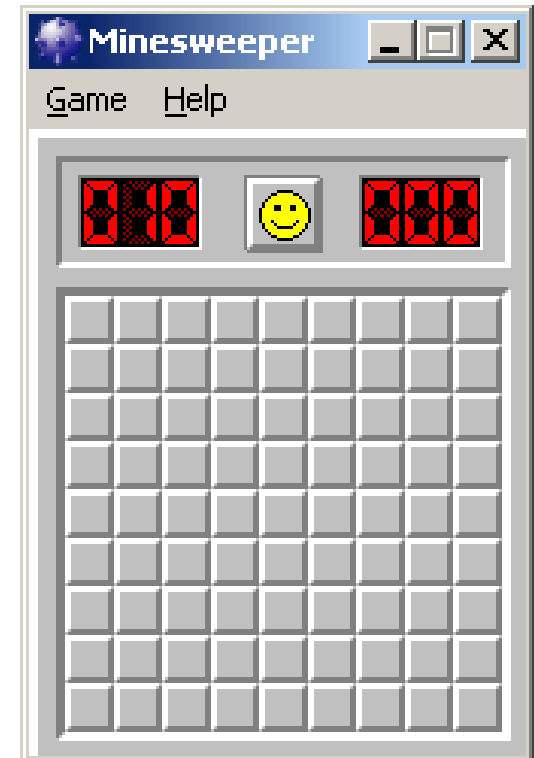
#### Returns:

hodnota ceny listka

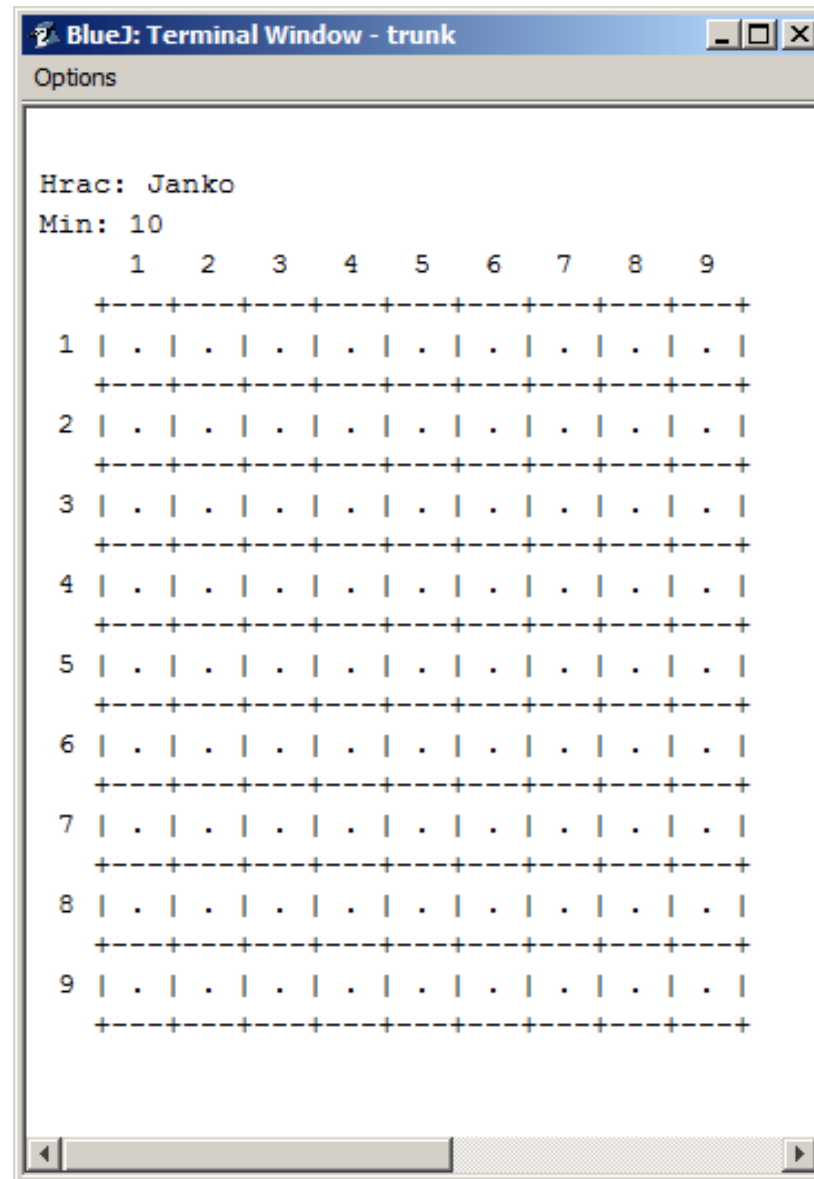


# Projekt miny – zadanie

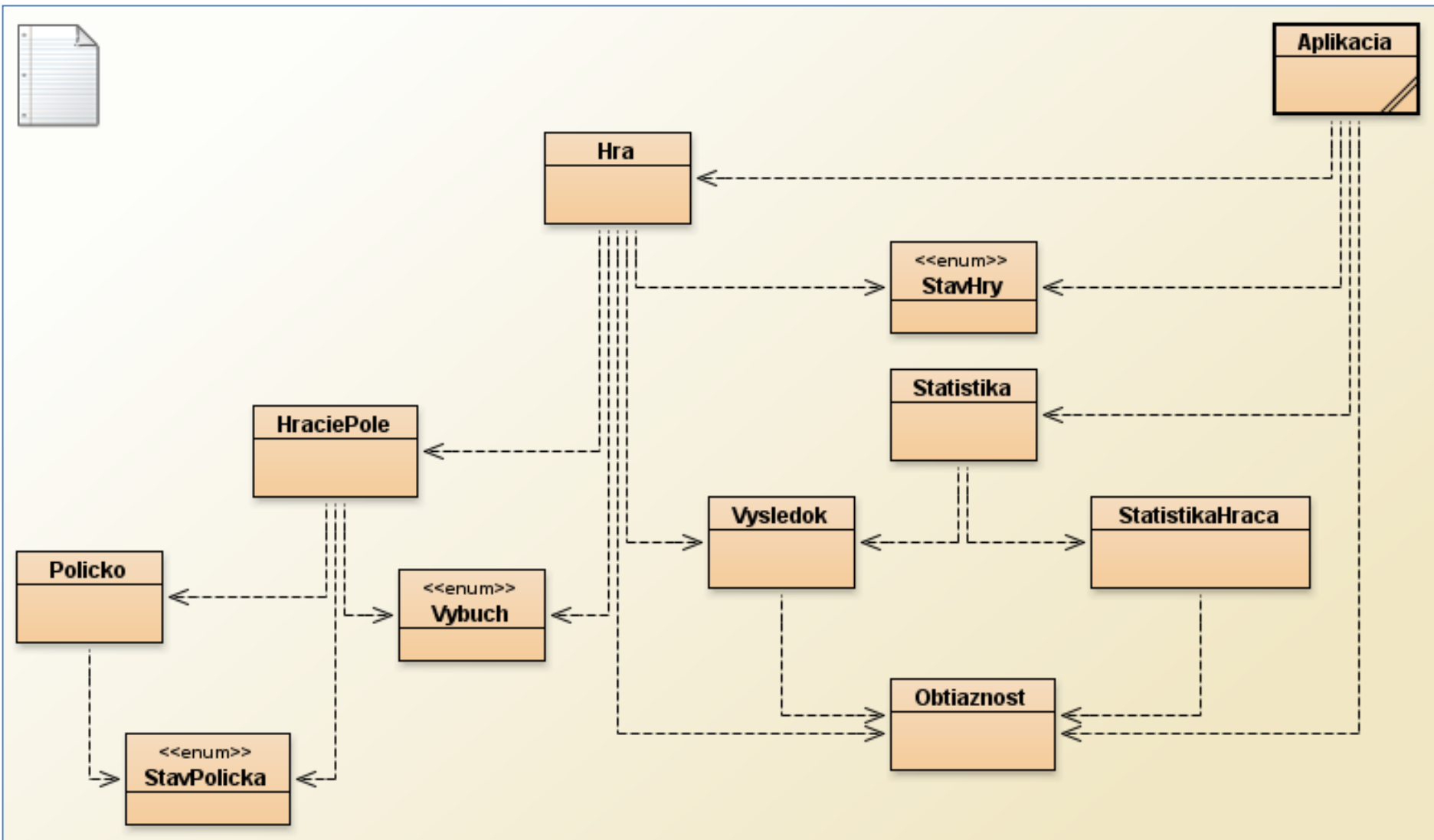
- známa logická hra míny
- cieľom hry Míny je určiť umiestnenia všetkých mín tak, aby ste pritom žiadnu z nich neodkryli
- ak odkryjete mínu, prehrávate



# Projekt miny – grafická reprezentácia



# BlueJ – diagram tried



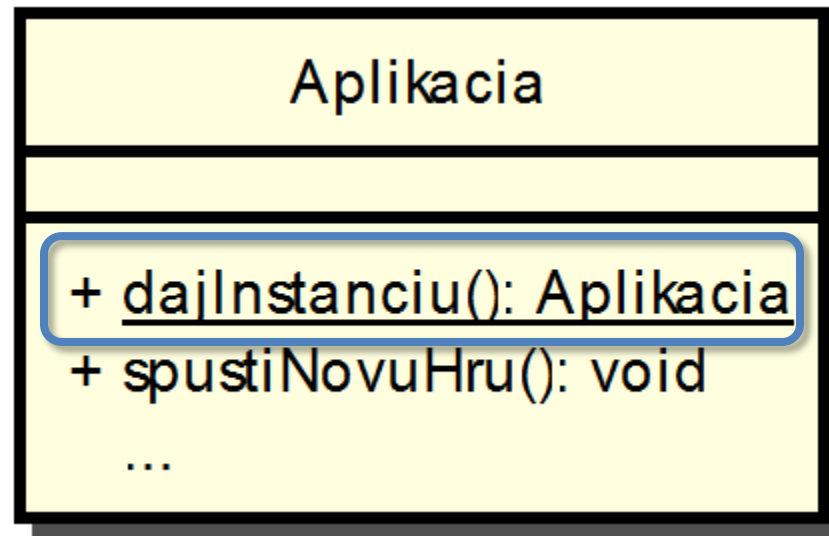
# Trieda Aplikacia – zadanie

- prístup ku hre míny
- len jedna inštancia
- zmena mena hráča
- zmena obtiažnosti
- spustenie novej hry
- odkrytie políčka a označenie míny
- poskytovanie informácií o stave hry (výhra/prehra)
- zobrazenie štatistiky
- ...

# Trieda Aplikacia

- jediná inštancia
  - správa new – ľubovoľný počet inštancií
  - nová správa triede – dajInstanciu
  - správa new – nesmie byť vo verejnom rozhraní
- 
- Návrhový vzor Singleton – Jedináčik
  - Projekt „tvary“ – trieda Platno

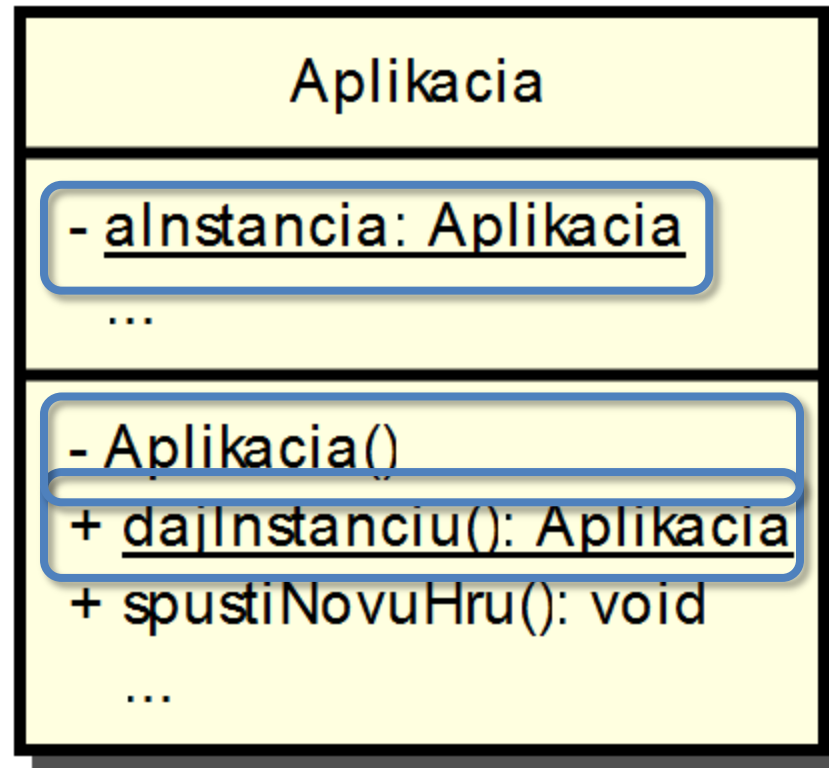
# Trieda Aplikacia – rozhranie



# Trieda Aplikacia – vnútorný pohľad<sub>(1)</sub>

- nová metóda triedy
  - reakcia na správu triede Aplikacia.dajInstanciu()
- nový atribút triedy
  - aInstancia – jediná inštancia triedy
- konštruktor označený ako private
  - označenie, že trieda nemá verejnú správu new

# Trieda Aplikacia – vnútorný pohľad<sub>(2)</sub>





# Aplikacia – trieda

```
public class Aplikacia
{
    private static Aplikacia aInstancia;

    private Aplikacia()
    {
        ...
    }
}
```

# Aplikacia – metóda daJInstanciu

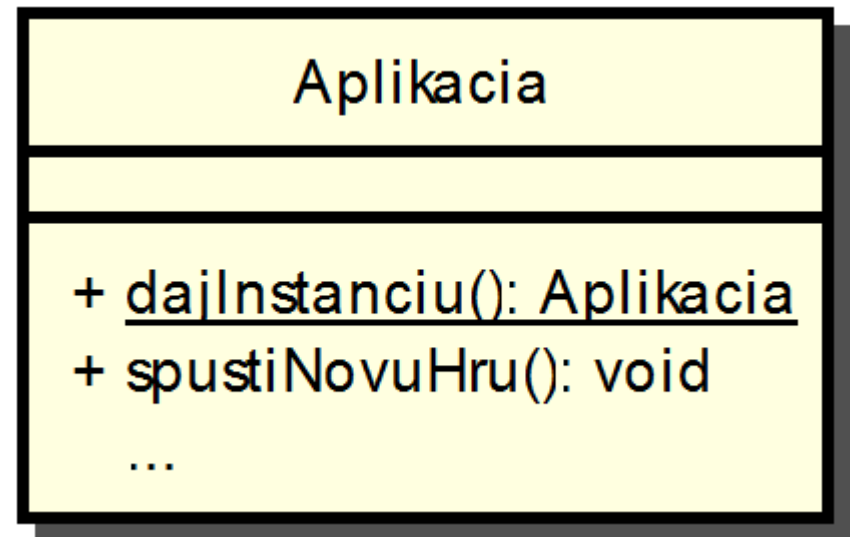
```
public static Aplikacia daJInstanciu()  
{  
    if (aInstancia == null) {  
        aInstancia = new Aplikacia();  
    }  
  
    return aInstancia;  
}
```

# Trieda ako objekt

- vonkajší pohľad
  - rozhranie triedy – správy triede
    - doteraz len správa new
- vnútorný pohľad
  - atribúty triedy
  - metódy triedy

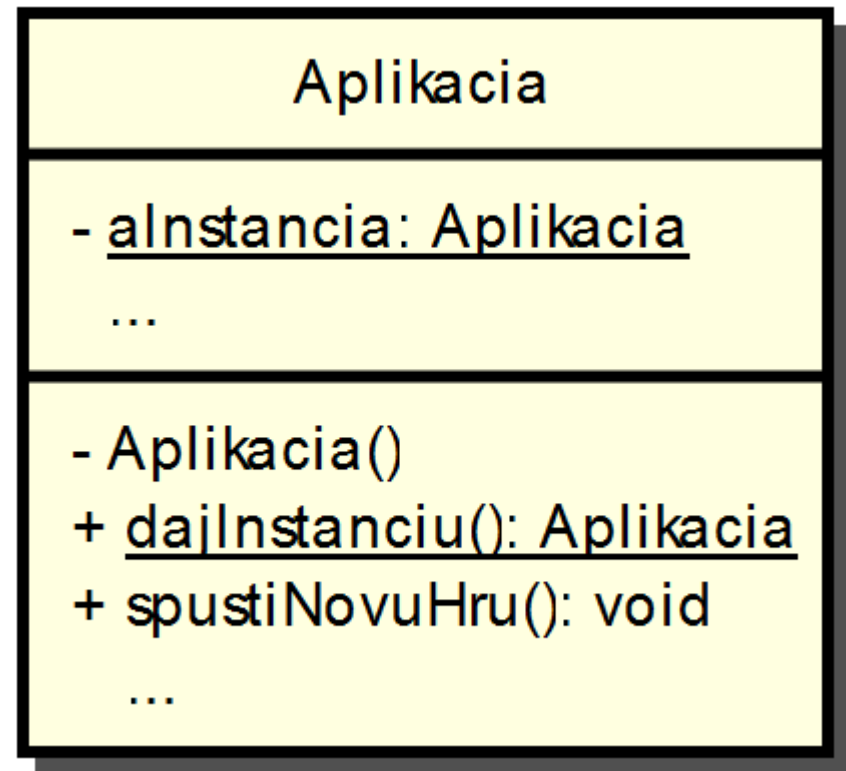
# Trieda ako objekt v UML

- správy triede a správy inštancii – spoločné rozhranie
- správy triede – podčiarknuté



# Trieda ako objekt v UML

- vnútorný pohľad – atribúty a metódy triedy aj inštancie sú spolu
- atribúty a metódy triedy – podčiarknuté



# Trieda ako objekt v jazyku Java

- rovnako ako v UML – atribúty a metódy triedy aj inštancie sú spolu v definícii triedy
- rozlíšenie – kľúčové slovo static

```
private static Aplikacia aInstancia;
```

```
public static Aplikacia dajInstanciu()  
{  
    ...  
}
```

# Poradie definícií v triede – konvencia

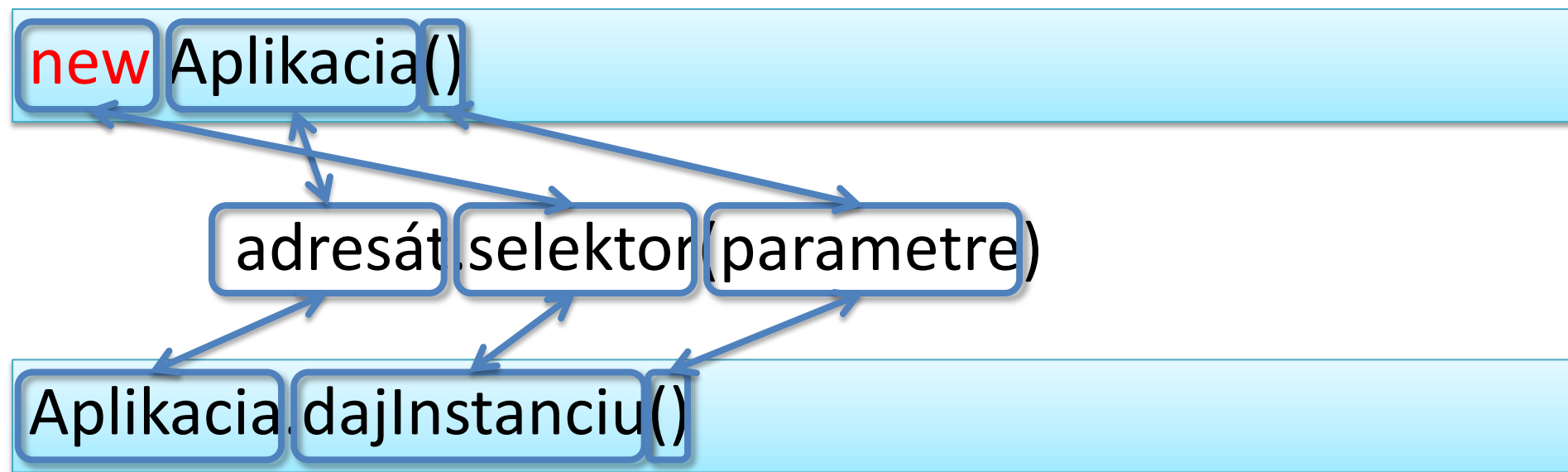
- konvencia zavedená firmou Sun

1. atribúty triedy
2. atribúty inštancie
3. konštruktory
4. metódy triedy
5. metódy inštancie

- a) verejné zložky – public
- b) neverejné zložky – private

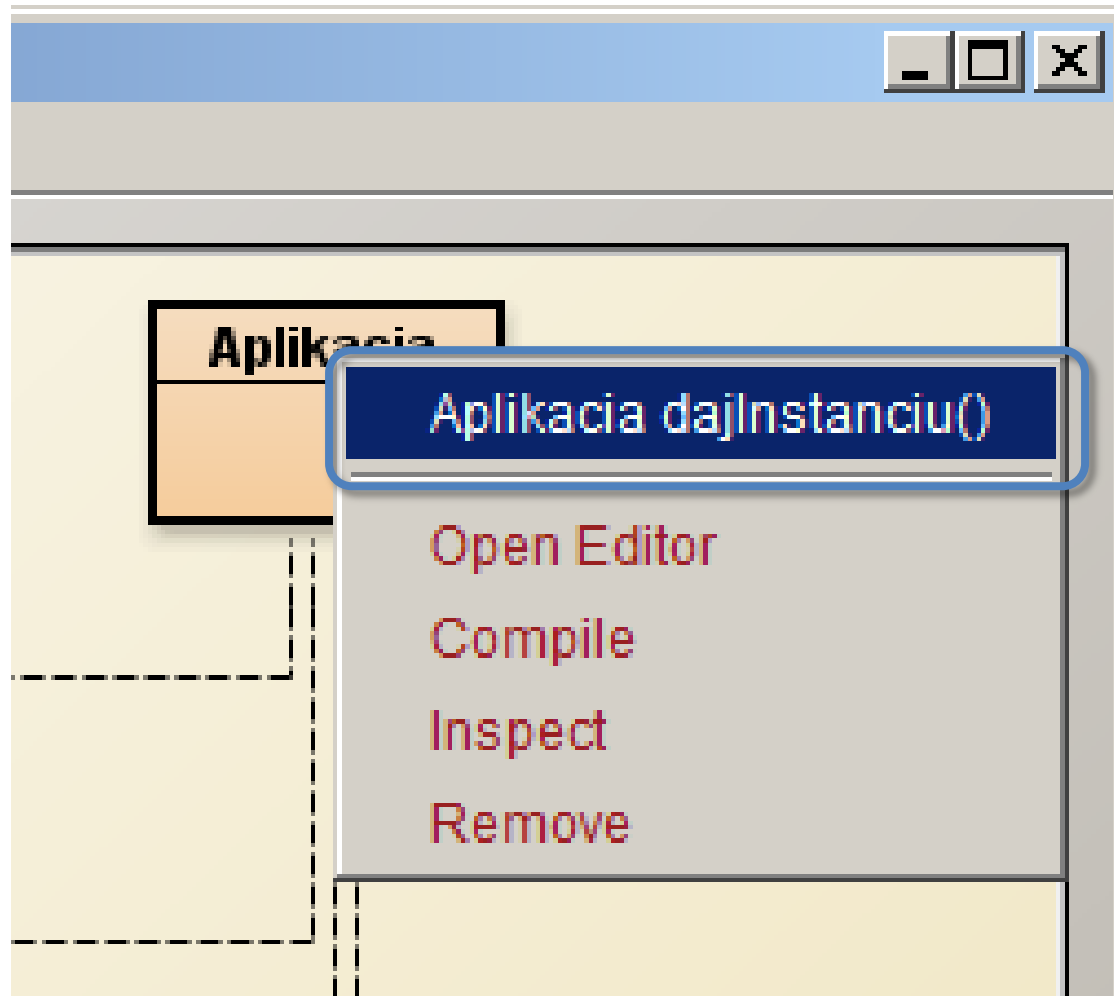
# Posielanie správy triede v jazyku Java

- ako adresát správy sa uvádza trieda
- špeciálna správa new – špeciálny zápis
- ostatné správy – štandardný zápis





# Posielanie správy triede v nástroji BlueJ

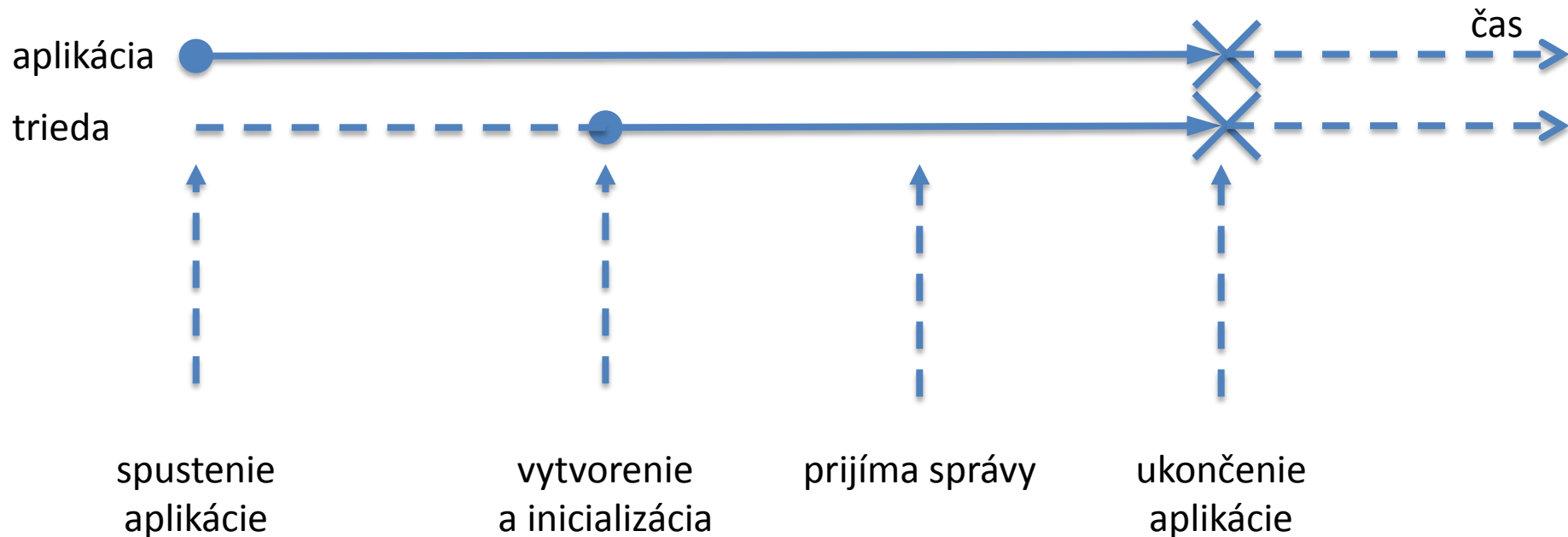


# Inštanície a trieda – vzájomný prístup

- trieda môže svojim inštanciam posilať správy zo súkromného rozhrania
- inštancia môže svojej triede posilať správy zo súkromného rozhrania
- trieda môže priamo pristupovať ku atribútom svojich inštancií
- inštancia môže priamo pristupovať ku atribútom svojej triedy

# Životný cyklus triedy v jazyku Java

- trieda je priamo definovaný objekt
- vzniká pri prvom použití (prvá správa triede)
- zaniká spolu s ukončením aplikácie



# Súkromný konštruktor

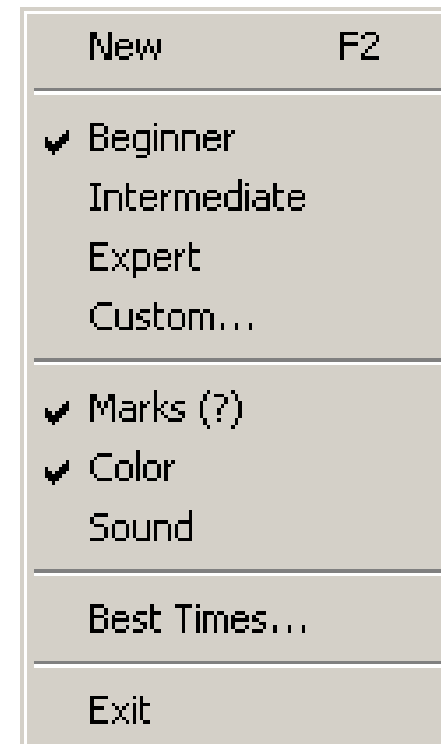
- private konštruktor – správa new v súkromnom rozhraní
- ak má trieda len súkromný konštruktor, musí mať metódu triedy, ktorá sa stará o vytvorenie

# Jedináčik

```
public class Jedinacik
{
    private static Jedinacik aInstancia;
    private Jedinacik() { }
    public static Jedinacik dajInstanciu()
    {
        if (aInstancia == null)
            aInstancia = new Jedinacik();
        return aInstancia;
    }
}
```

# 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

## Obtiaznost

```
+ new(paNazov: String, paPocetMin: int, paVyska: int, paSirka: int): Obtiaznost  
+ dajNazov(): String  
+ dajPocetMin(): int  
+ dajVysku(): int  
+ dajSirku(): int  
+ toString(): String
```



# 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
  - obalovacie 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 aLahka
        = new Obtiaznost("Lahka", 10, 9, 9);
    private static final Obtiaznost aStredna
        = new Obtiaznost("Stredna", 40, 16, 16);
    private static final Obtiaznost aTazka
        = 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

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

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

```
private static final String aNazov = "Nikto";
```

```
private static final Obtiaznost aLahka  
    = 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

## Aplikacia

- «final» aLahka: Obtiaznost = new Obtiaznost("Lahka", 10, 9, 9)
- «final» aStredna: Obtiaznost = new Obtiaznost("Stredna", 40, 16, 16)
- «final» aTazka: Obtiaznost = new Obtiaznost("Tazka", 99, 16, 30)

...

...



# Metóda Aplikacia.nastavObtiaznost

```
public void nastavObtiaznost(String paNazov)
{
    if (aLahka.dajNazov().equals(paNazov)) {
        aObtiaznost = aLahka;
    } else if (aStredna.dajNazov().equals(paNazov)) {
        aObtiaznost = aStredna;
    } else if (aTazka.dajNazov().equals(paNazov)) {
        aObtiaznost = aTazka;
    }
}
```

# 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 paPocetMin,  
                             int paVyska,  
                             int paSirka)  
{  
    aObtiaznost = new Obtiaznost(  
        "Vlastna", paPocetMin, paVyska, paSirka);  
}
```

# Preťažovanie správ a metód<sub>(1)</sub>

- dve správy s rovnakým selektorom:
  - nastavObtiaznost(String paNazov)
  - nastavObtiaznost(int paPocetMin, int paVyska, int paSirka)
- odlišnosť – počet a typ parametrov

# Preťažovanie správ a metód<sub>(2)</sub>

- 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<sub>(3)</sub>

- identifikátor metódy si vyjadríme ako:
  - nazovMetody#typParametra1#typParametra2#...
  - typParametra = typ formálneho parametra
- napr:
  - **public void** nastavObtiaznost(String paNazov)  
=> nastavObtiaznost#String
  - **public void** nastavObtiaznost(**int** m, **int** r, **int** s)  
=> nastavObtiaznost#int#int#int

# Preťažovanie správ a metód<sub>(4)</sub>

- 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