

# 9

Viac na tému trieda

# Pojmy zavedené v 8. prednáške<sub>(1)</sub>

- N-rozmerné polia
  - Dvojrozmerné polia – matica
  - definícia – `typ[][]` premenna
  - inicializácia – `new typ[pocetRiadkov][pocetStlpcov]`
  - práca s prvkami – `premenna[riadok][stlpec]`
- Pole polí
  - inicializácia – `new typ[pocetRiadkov][]`
  - inicializácia prvkov
    - `pole[riadok] = new typ[pocetStlpcov]`

# Pojmy zavedené v 8. prednáške<sub>(2)</sub>

- vnorené cykly
- this
- sekcie rozhrania
  - verejné – ľubovoľný objekt v okolí
  - neverejné – len objekt vo svojom súkromí

# Pojmy zavedené v 8. prednáške<sub>(3)</sub>

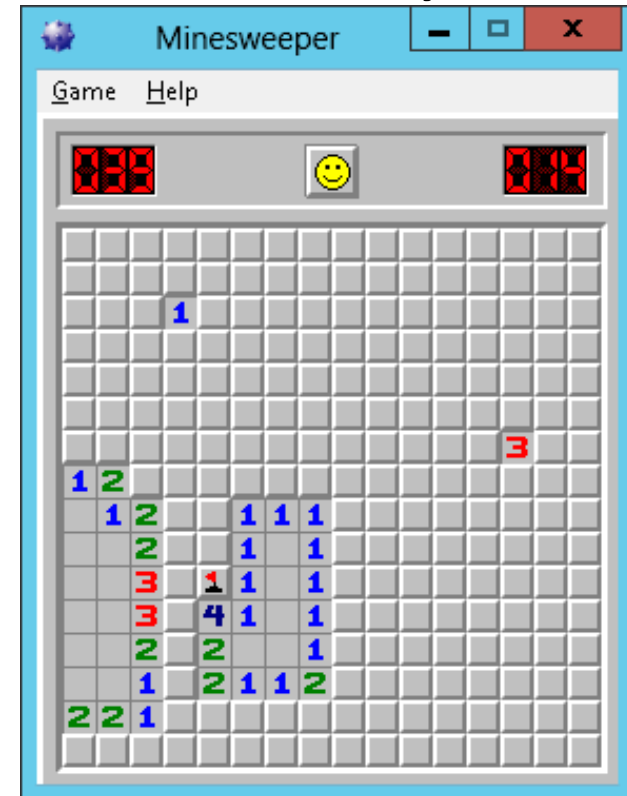
- rekurzia
  - princíp
  - v jazyku Java
  - v objektovom programovaní

# Cieľ prednášky

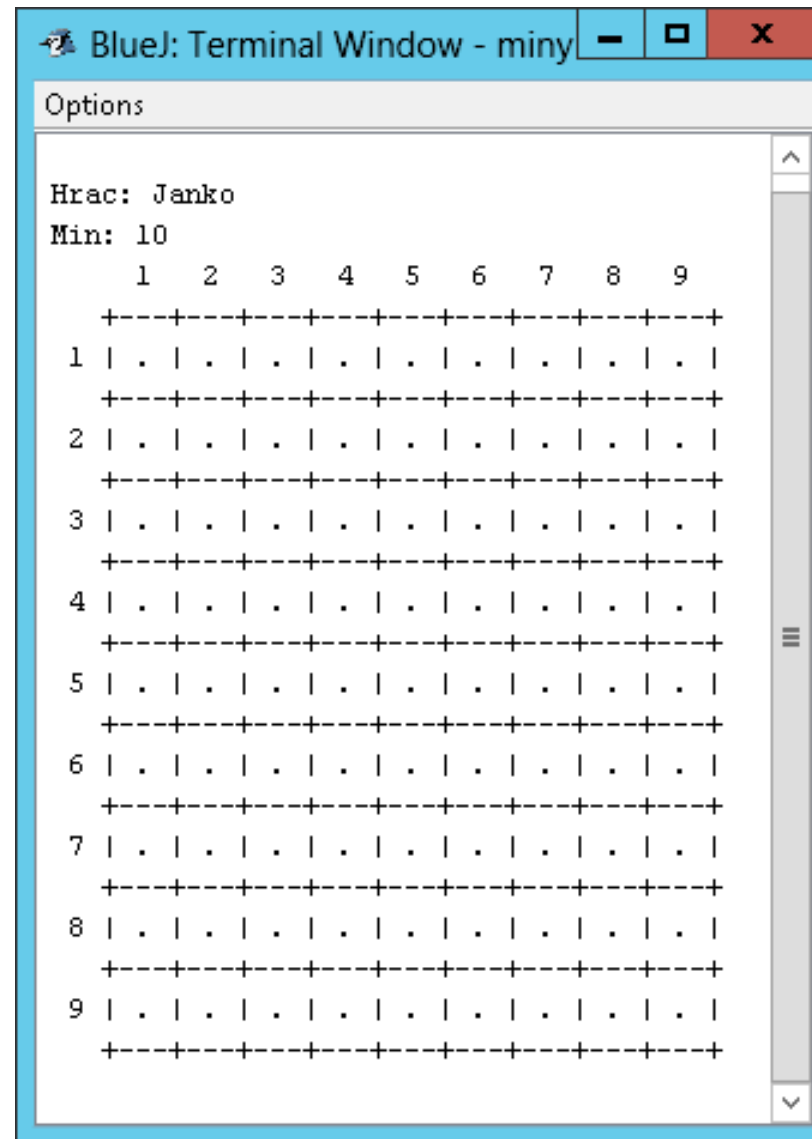
- trieda ako objekt
- trieda ako množina svojich inštancií
  - enum
- základná práca s textovými súbormi
- príklad: míny

# Projekt miny – zadanie

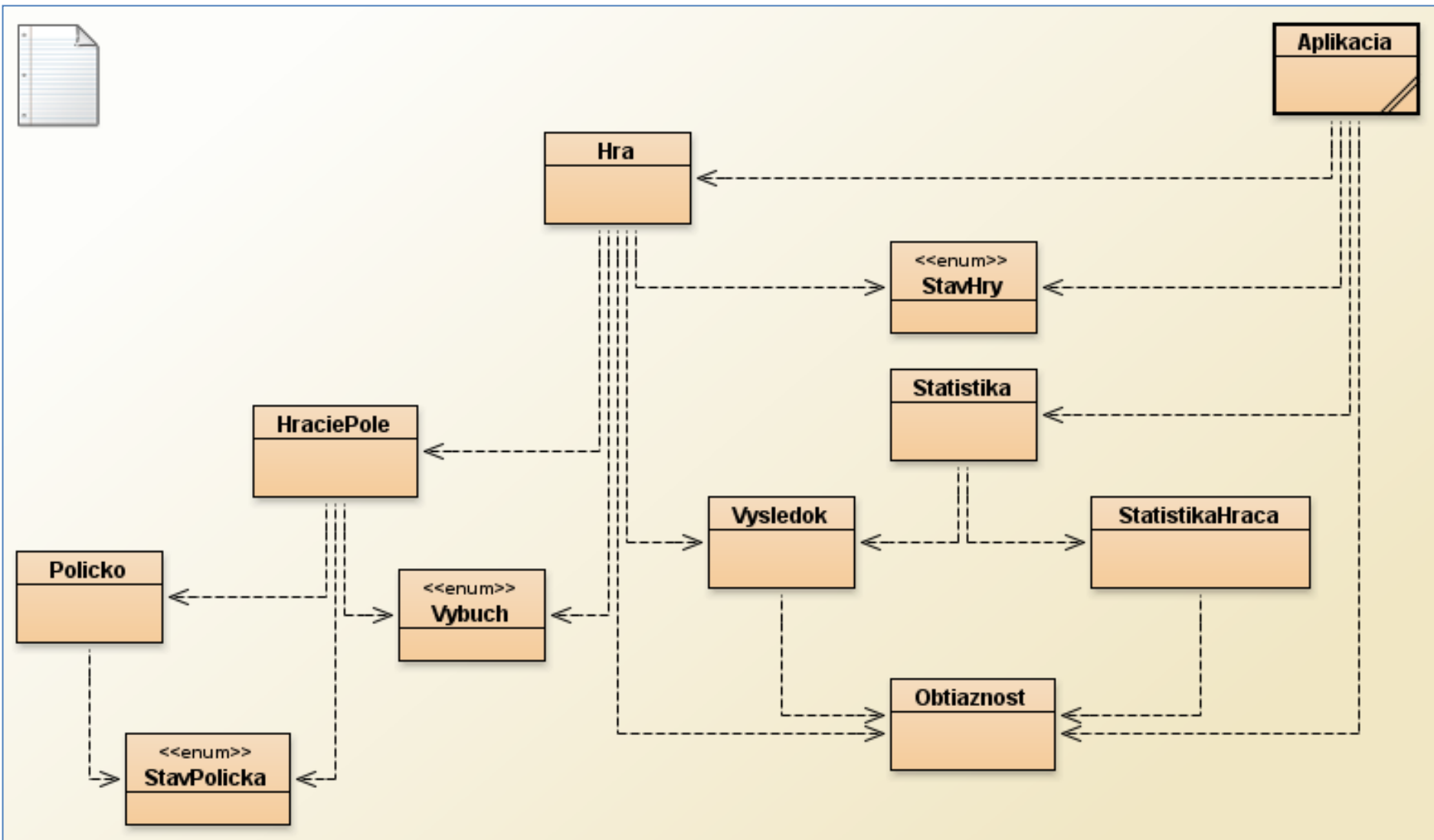
- známa logická hra míny
- cieľom hry Míny je určiť umiestnenia všetkých mín tak, aby ste pritom žiadnu z nich neodkryli
- ak odkryjete mínu, prehrávate



# Projekt miny – grafická reprezentácia



# BlueJ – diagram tried





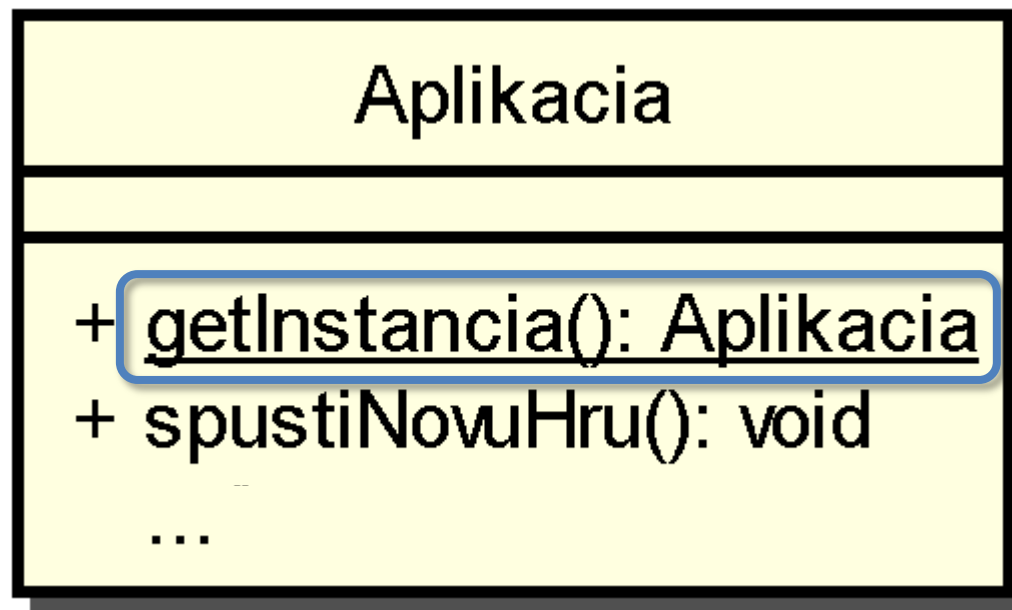
# Trieda Aplikacia – zadanie

- 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 – getInstance
  - správa new – nesmie byť vo verejnom rozhraní
- 
- Návrhový vzor Singleton – Jedináčik
  - Projekt „tvary“ – trieda Platno

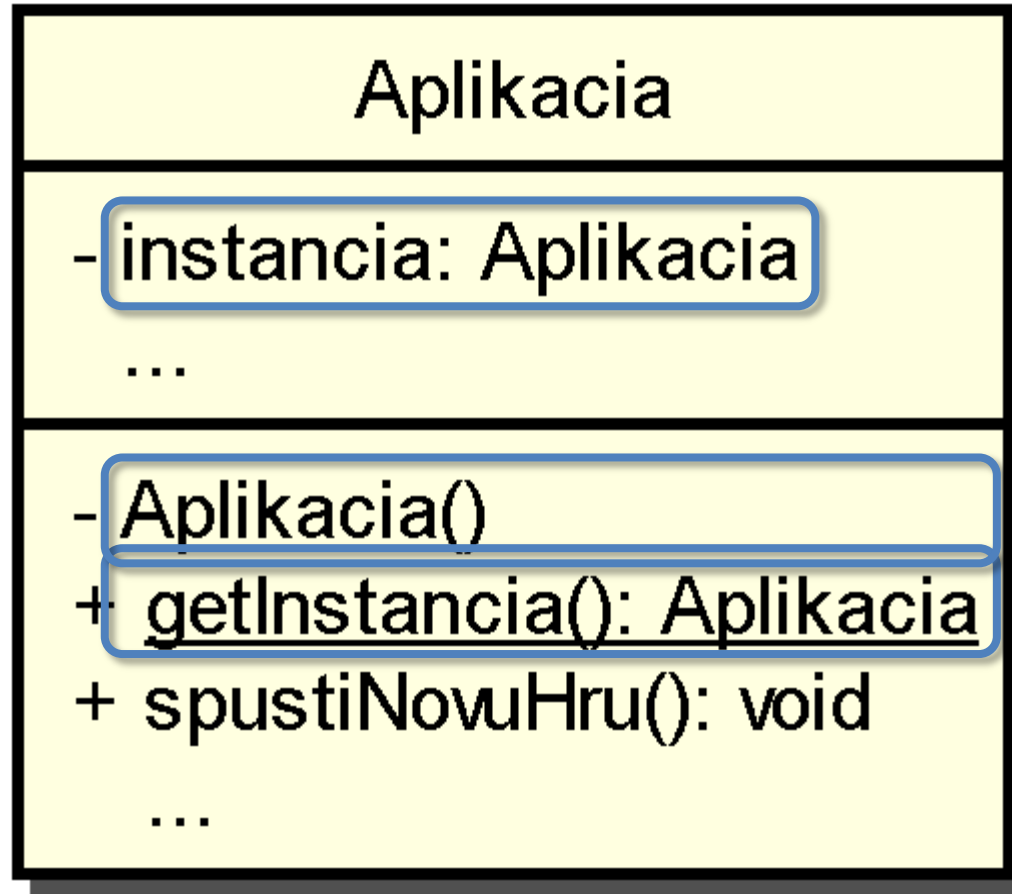
# Trieda Aplikacia – rozhranie



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

- nová metóda triedy
  - reakcia na správu triede Aplikacia.getInstance()
- nový atribút triedy
  - instancia – 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 instancia;  
  
    private Aplikacia() {  
        ...  
    }  
}
```

# Aplikacia – metóda getInstance()

```
public static Aplikacia getInstance() {  
    if (Aplikacia.instancia == null) {  
        Aplikacia.instancia = new Aplikacia();  
    }  
  
    return Aplikacia.instancia;  
}
```

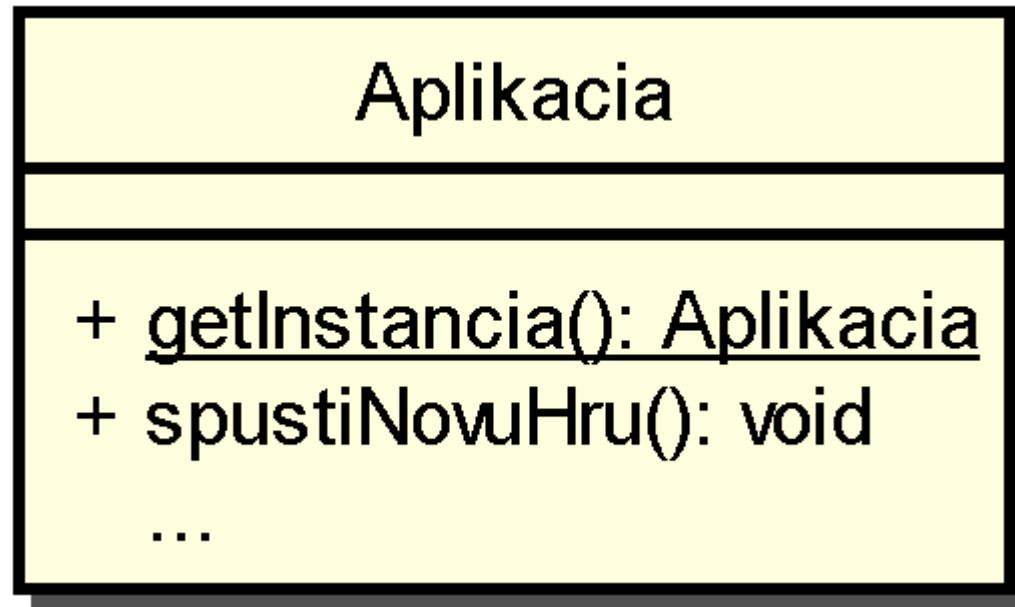
# Trieda ako objekt

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



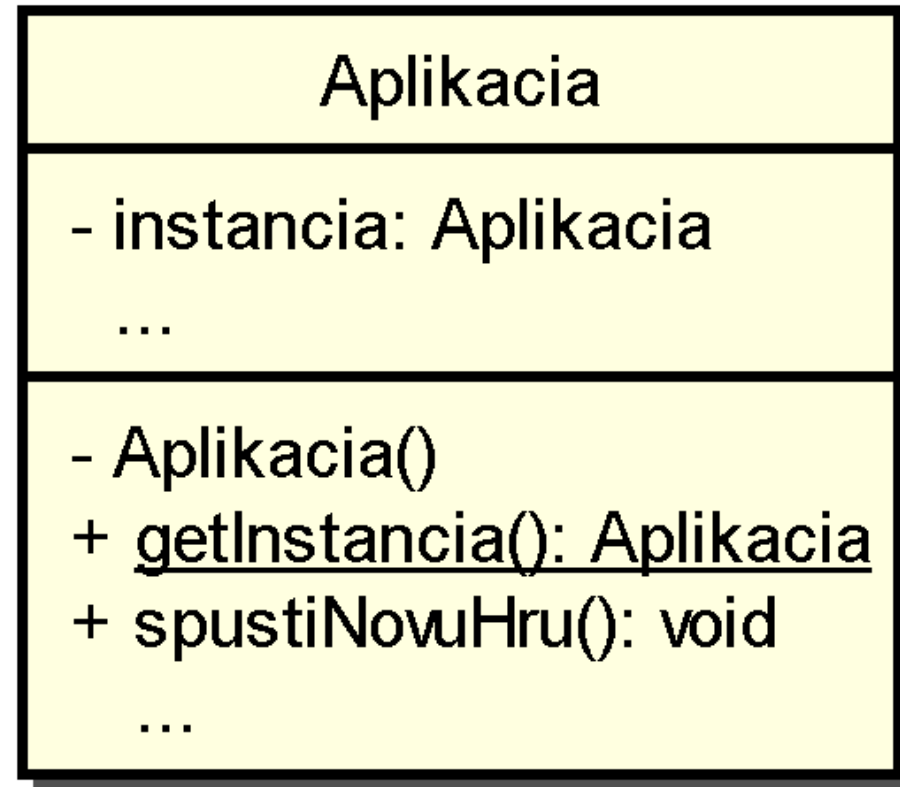
# Trieda ako objekt v UML<sub>(1)</sub>

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



# Trieda ako objekt v UML<sub>(2)</sub>

- 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 instancia;
```

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

# Poradie definícií v triede – konvencia

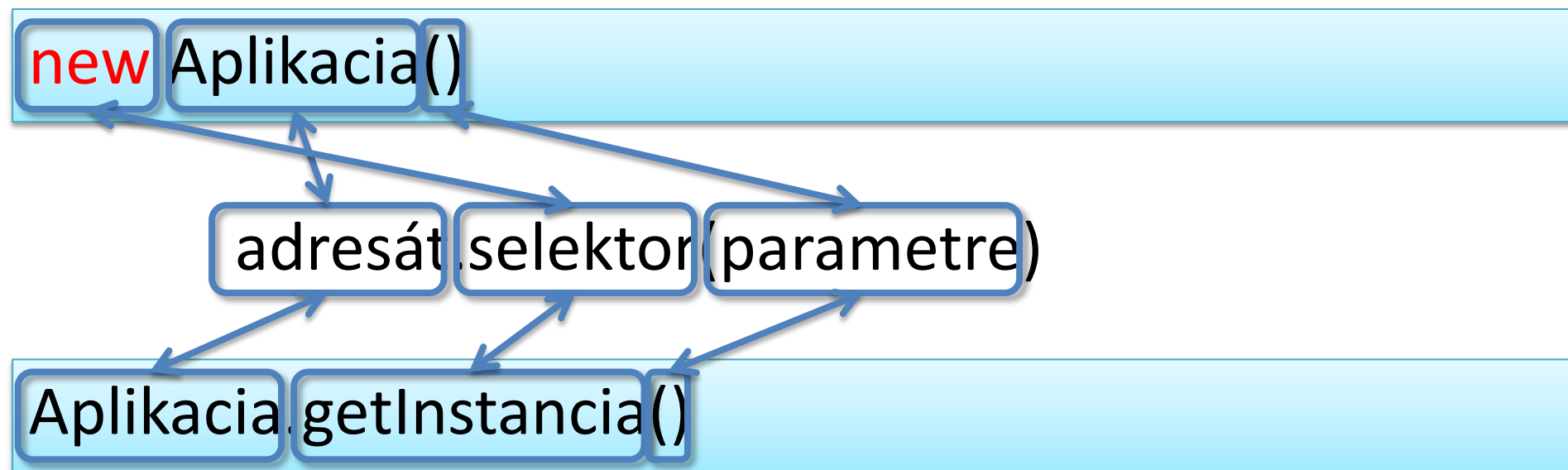
- konvencia zavedená firmou Sun

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

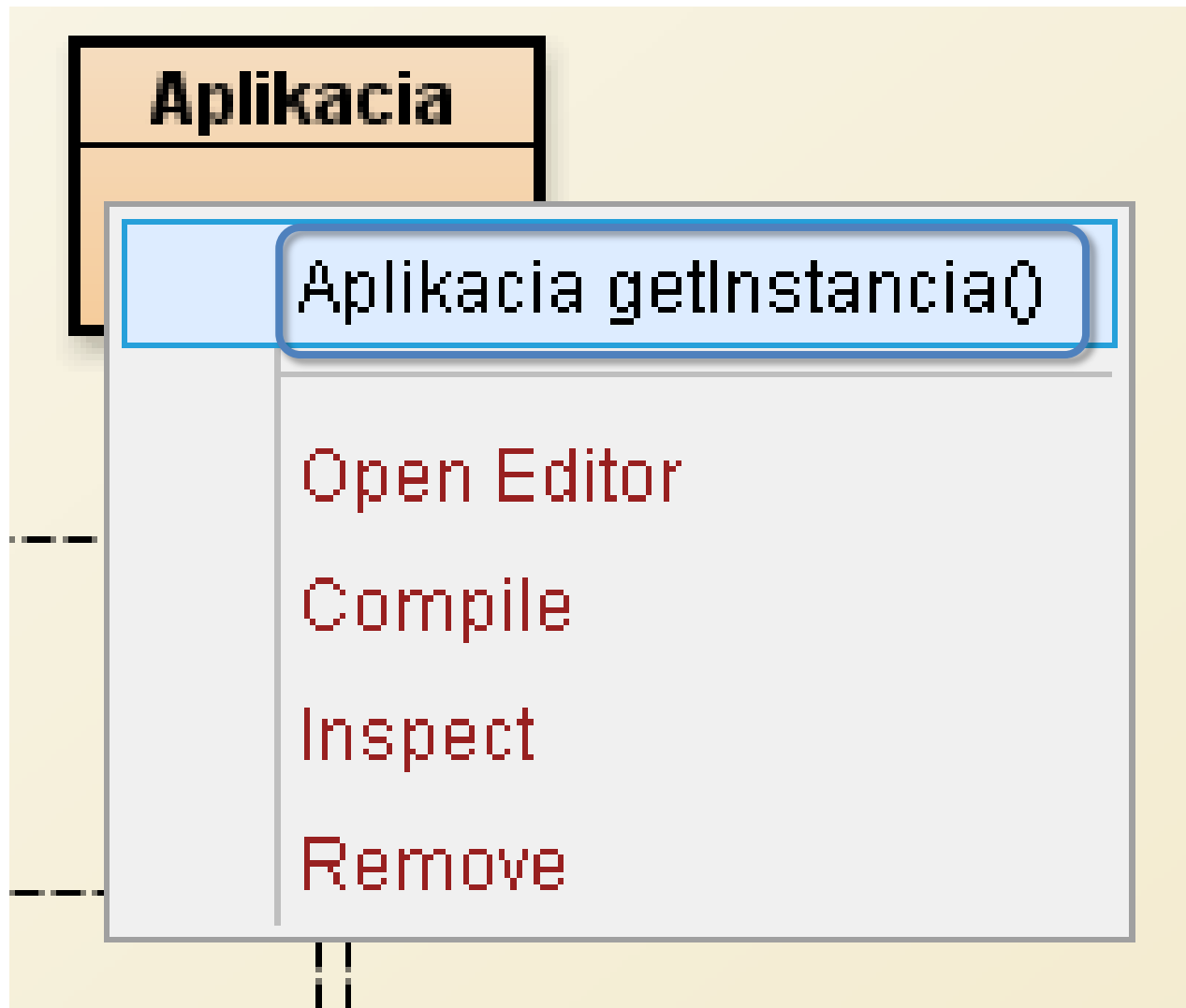
- a) verejné zložky – public
- b) neverejné zložky – private

# 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

# Súkromný konštruktor

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

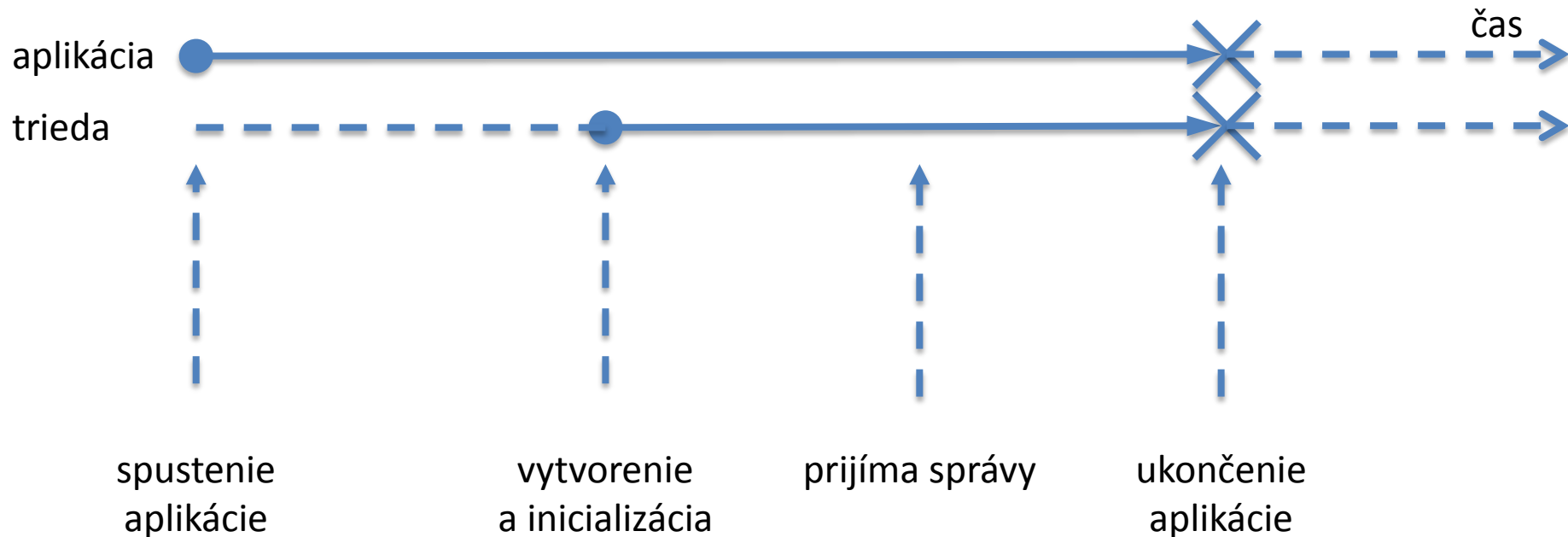


# Jedináčik

```
public class Jedinacik {  
    private static Jedinacik instancia;  
    private Jedinacik() {  
    }  
    public static Jedinacik getInstancia() {  
        if (Jedinacik.instancia == null)  
            Jedinacik.instancia = new Jedinacik();  
        return Jedinacik.instancia;  
    }  
}
```

# Životný cyklus triedy v jazyku Java

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



# Jedináčik – iná implementácia

```
public class Jedinacik {  
    private static Jedinacik instancia = new Jedinacik();  
  
    private Jedinacik() {  
    }  
  
    public static Jedinacik getInstancia() {  
        return Jedinacik.instancia;  
    }  
}
```

# Míny – ďalšia požiadavka

- poskytovanie informácií o stave hry
  - výhra
  - prehra
- metóda `getStavHry()`

# Vyjadrenie stavu hry<sub>(1)</sub>

- nový atribút „vyhral“
  - true – hráč vyhral
  - false – hráč prehral
- problémy:
  1. aký stav je v priebehu hry?
    - true – nesprávne, hráč ešte nevyhral
    - false – nesprávne, hráč ešte neprehral

# Vyjadrenie stavu hry v priebehu hry

- ďalší nový atribút „hraSkoncila“
  - false – hra ešte neskončila
  - true – hra už skončila, výsledok je vo „vyhral“
- problémy:
  1. ak hra ešte neskončila, dotazom na „vyhral“ dostaneme vždy nesprávnu odpoved'

# Vyjadrenie stavu hry<sub>(2)</sub>

- možné stavy:
  - nerozhodnuta – hra ešte neskončila
  - vyhra – hráč vyhral
  - prehra – hráč stupil na mínu
- záver: dve hodnoty nestačia, treba tri

# Vyjadrenie stavu hry<sub>(3)</sub>

- možné stavy – číslovanie stavov:
  - hodnota 0 – hra ešte neskončila
  - hodnota 1 – hráč vyhral
  - hodnota 2 – hráč stupil na mínu
- problémy:
  1. neprehľadné, pri pohľade na zdrojové kódy nie je jasný význam čísla
  2. pri preklade neoznámi prekladač nesprávnu hodnotu (-1, 3, ...)
  3. programátorská hrdosť nedovolí také primitívne riešenie



# Vyjadrenie stavu hry<sub>(4)</sub>

- možné stavy – označenie reťazcami:
  - hodnota "nerozhodnuta" – hra ešte neskončila
  - hodnota "vyhral" – hráč vyhral
  - hodnota "prehral" – hráč stupil na mínu
- problémy:
  1. pri preklade neoznami prekladač nesprávnu hodnotu (preklepy)
  2. pozor na porovnávanie – equals
  3. programátorská hrdosť nedovolí také primitívne riešenie

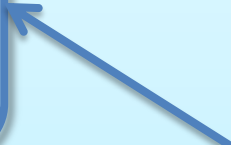
# Vyjadrenie stavu hry<sub>(5)</sub>

- riešenie – vymenovaný typ – enum
- extenzia triedy – množina všetkých inštancií triedy.
- enum – trieda s konštantnou extenziou.
- enum – trieda s pevne určenými inštanciami.
- enum – konštantná množina objektov
  - obsahuje svoje inštancie ako nemenné objekty

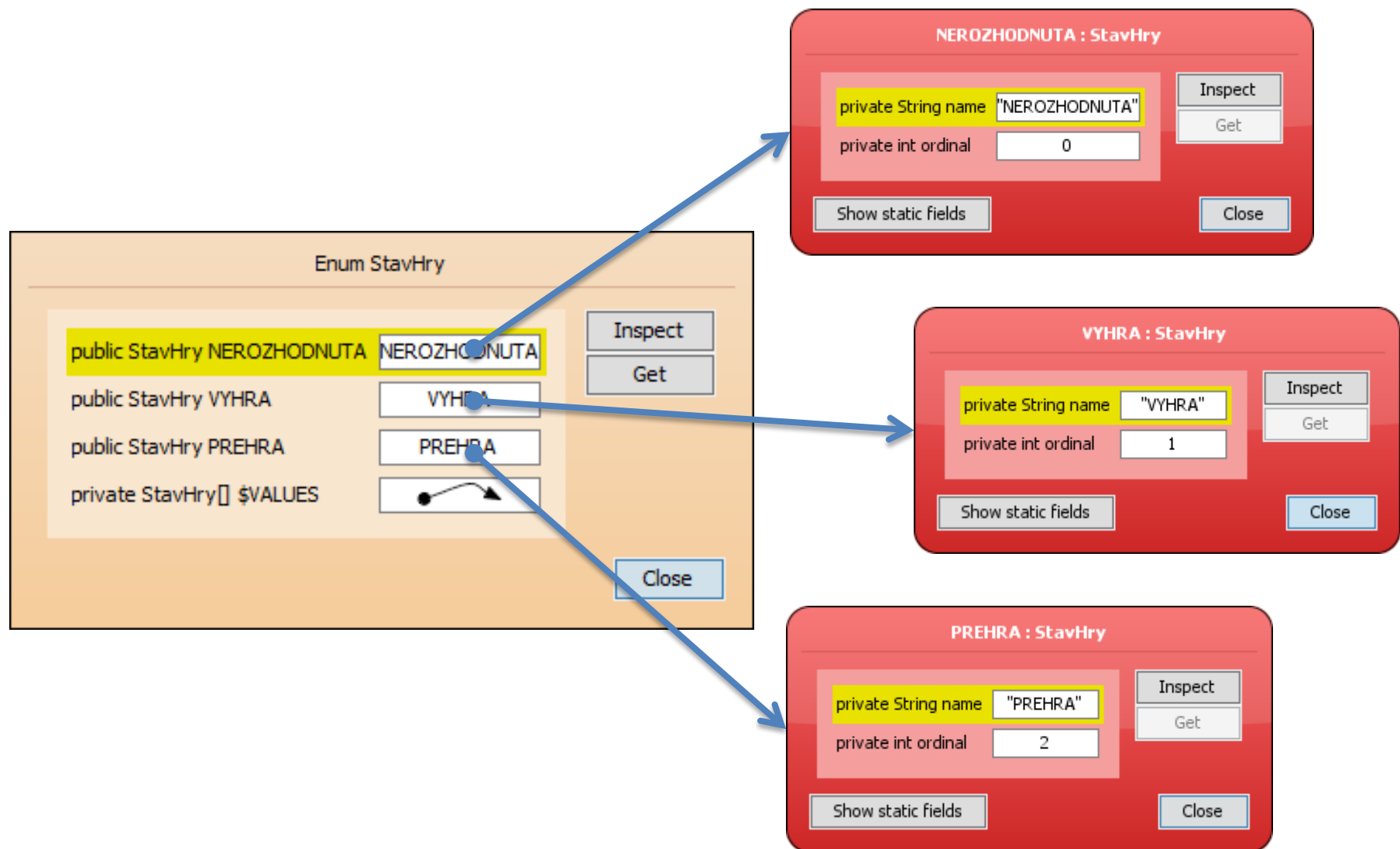
# Enum StavHry

```
public enum StavHry {  
    NEROZHODNUTA,  
    VYHRA,  
    PREHRA;  
}
```

zoznam inštancií  
– extenzia triedy StavHry



# Enum – StavHry



# Použitie enum StavHry – trieda Hra<sub>(1)</sub>

```
public class Hra {  
    private StavHry stav;  
  
    public Hra(...) {  
        this.stav = StavHry.NEROZHODNUTA;  
        ...  
    }  
}
```

# Použitie enum StavHry – trieda Hra<sub>(2)</sub>

```
if (this.stav == StavHry.VYHRA) {  
    ...  
}  
  
...  
System.out.println(this.stav);
```

# Enum – špeciálna trieda

- zjednodušená syntax
- inštancie môžu mať atribúty
- inštancie môžu mať metódy

# Míny – ďalšia požiadavka

- informácie o stave políčka
  - enum
  - `getReprezentacia()` vracia reťazec, ktorý sa má vypísať do terminálu



# Enum – StavPolicka<sub>(1)</sub>

```
private char reprezentacia;
```

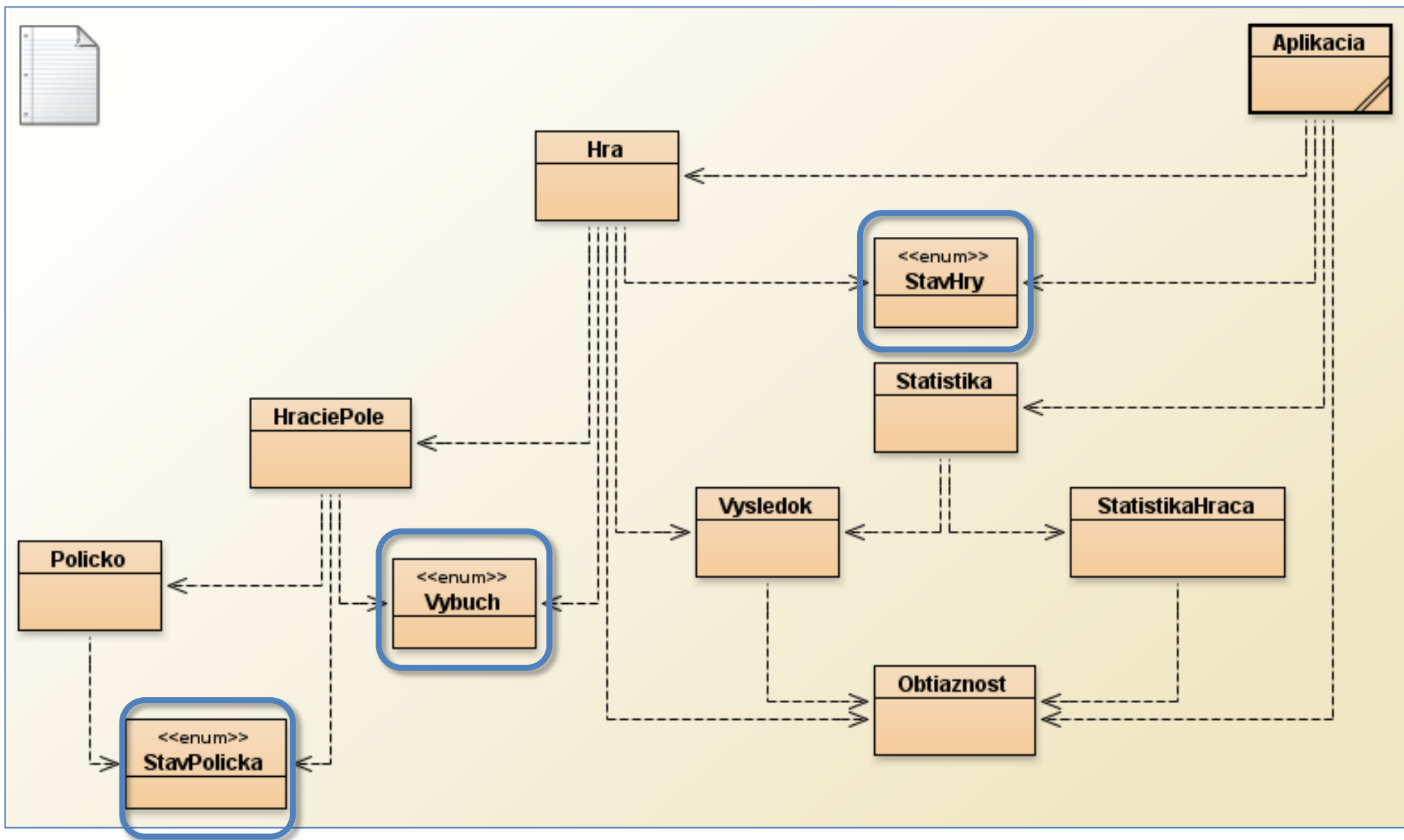
```
StavPolicka(char reprezentacia) {  
    this.reprezentacia = reprezentacia;  
}
```

```
public char getReprezentacia() {  
    return this.reprezentacia;  
}  
}
```

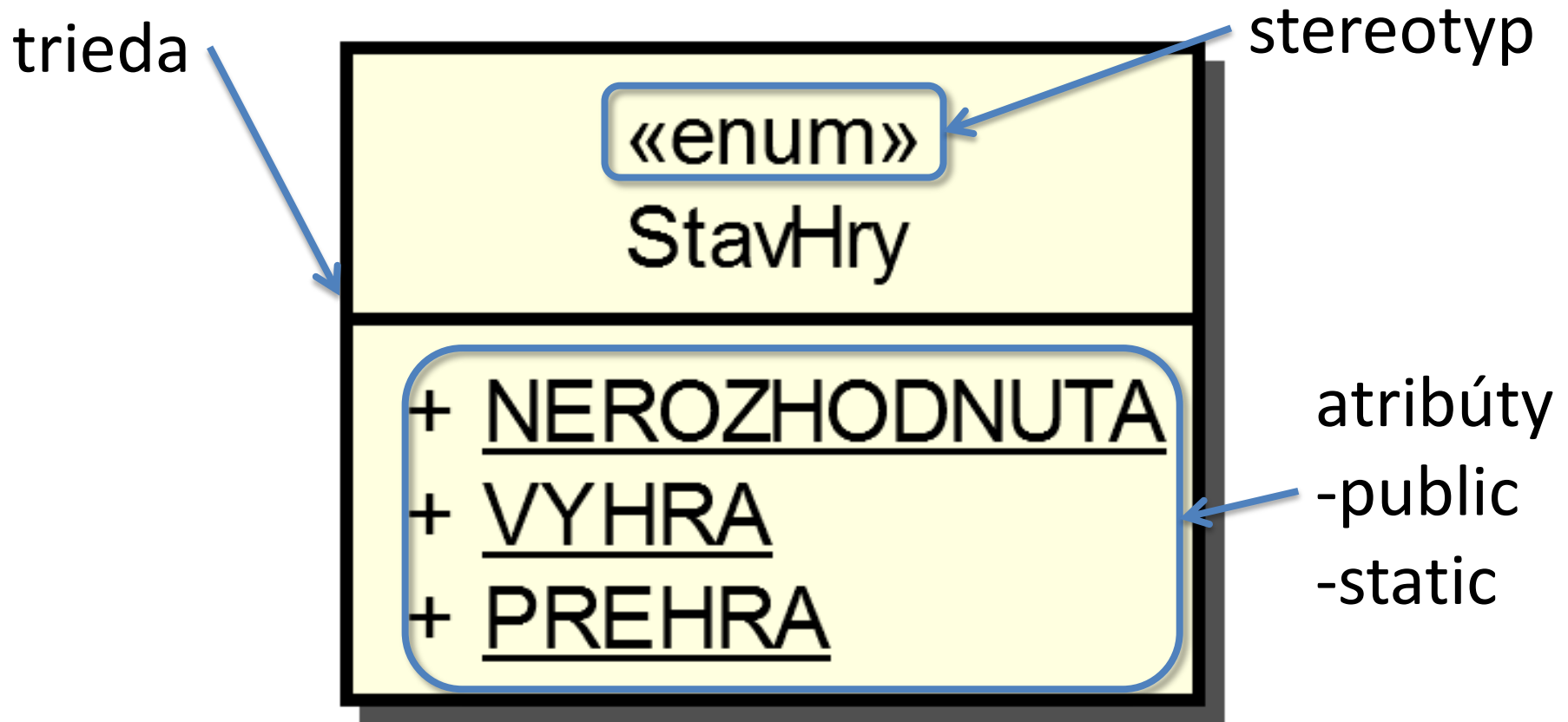
# Enum – StavPolicka<sub>(2)</sub>

```
public enum StavPolicka {  
    ZAKRYTE('.'),  
    PRAZDNE(''),  
    OZNACENE('F'),  
    ODKRYTE(' '),  
    UKAZANA_MINA('+'),  
    VYBUCHNUTE('*');  
    ...  
}
```

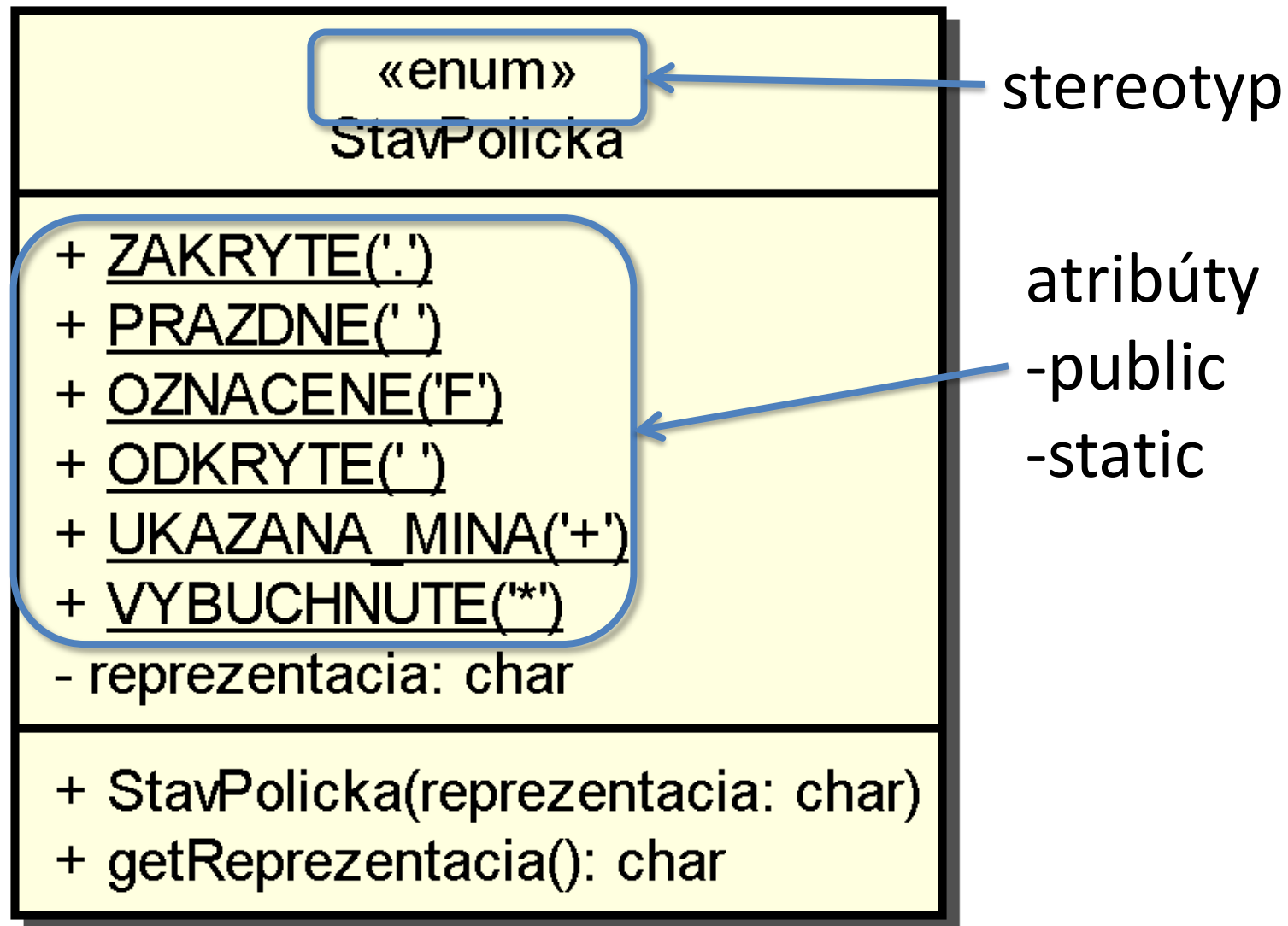
# BlueJ – diagram tried



# Enum v UML<sub>(1)</sub>



# Enum v UML<sub>(2)</sub>



# Vetvenie pomocou enum

- porovnanie inštancií enum na rovnosť referencií
- podmienené vykonanie:

```
if (this.stav == StavHry.VYHRA) {  
    ...  
}
```

# Vetvenie pomocou enum – else if

```
if (this.stav == StavHry.VYHRA) {  
    ...  
} else if (this.stav == StavHry.PREHRA) {  
    ...  
} else if (this.stav == StavHry.NEROZHODNUTA) {  
    ...  
}
```

# Vetvenie pomocou enum – switch

```
switch (this.stav) {  
    case VYHRA:  
        ...  
        break;  
    case PREHRA:  
        ...  
        break;  
    case NEROZHODNUTA:  
        ...  
        break;  
}
```



# Trieda ako množina

- trieda má extenziu – množinu inštancií
- ak zanedbáme ostatné vlastnosti – trieda reprezentuje množinu inštancií daného typu
- enum – konštantná množina

# Pohľady na triedu – zhrnutie<sub>(1)</sub>

## 1. trieda ako objekt

### – vonkajší pohľad

- rozhranie – správy triede

### – vnútorný pohľad

- atribúty triedy
- metódy triedy

## 2. trieda ako továreň

- hlavná úloha triedy – vytvárať inštancie
- špeciálna správa new – žiadosť o novú inštanciu

## 3. trieda ako šablóna

- potreba poznať štruktúru inštancie pri vytváraní
- definícia vnútorného pohľadu na inštancie
  - atribúty inštancie
  - metódy inštancie

## 4. trieda ako typ

- predstavuje typ inštancie
- definícia premenných (atribúty, parametre, lokálne premenné)
- definícia typu návratovej hodnoty

## 5. trieda ako množina

- extenzia triedy – množina všetkých inštancií danej triedy
- špecifický prípad – enum

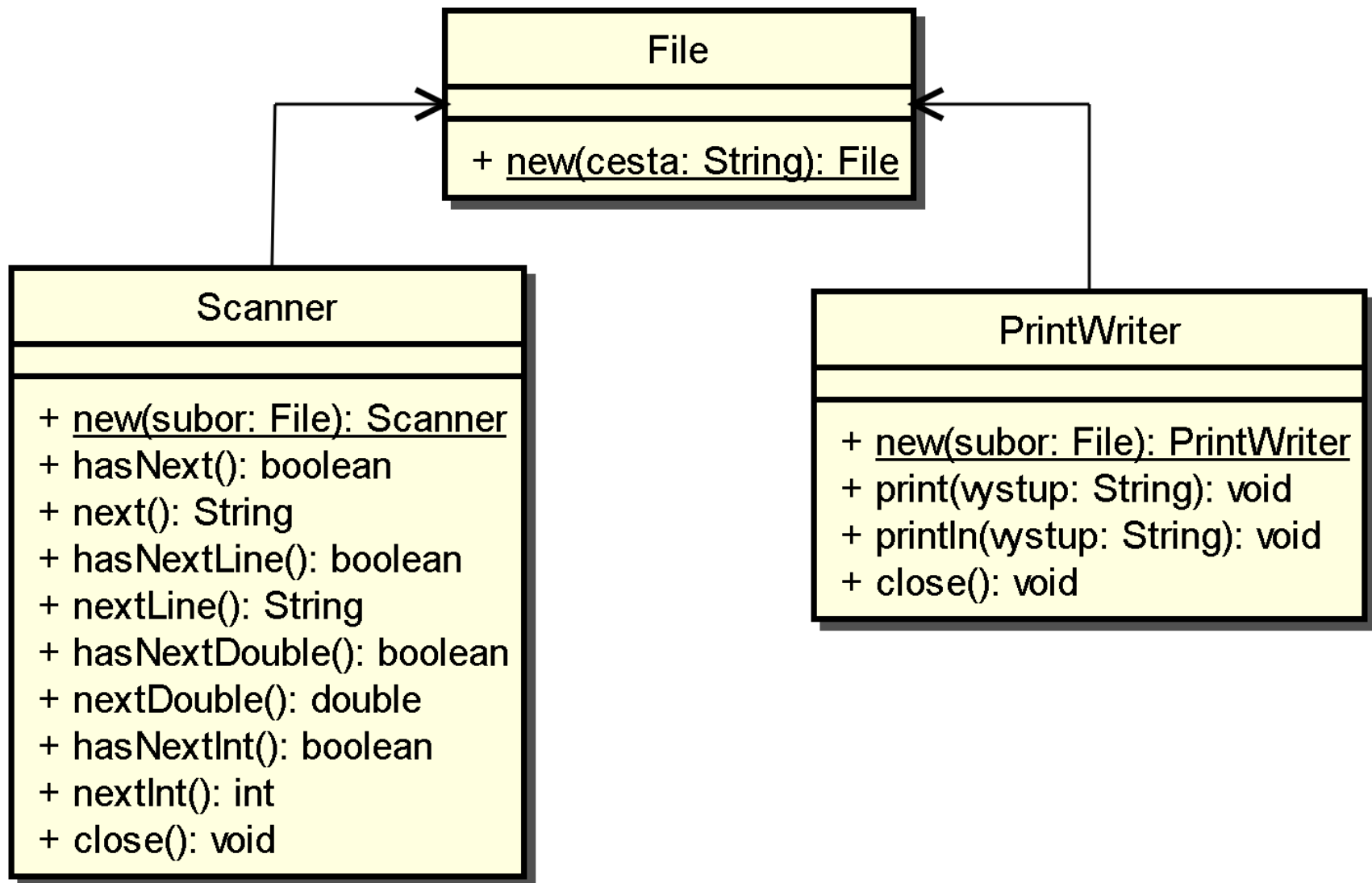
# Míny – nová požiadavka

- ukladanie štatistiky do súboru
  - počet výhier
  - počet prehíer
  - pre každého hráča, pre každú obtiažnosť
- načítanie štatistiky zo súboru

# Práca so súborom

- čítanie
  - otvorenie súboru na čítanie
  - postupné čítanie obsahu
  - zatvorenie súboru
- zápis
  - otvorenie súboru na zápis
  - postupný zápis nového obsahu
  - zatvorenie súboru

# Práca so súbormi v jazyku Java



# Práca so súborom – príkazy import

```
import java.util.Scanner;  
import java.io.File;  
import java.io.IOException;  
import java.io.PrintWriter;
```



# Míny – čítanie štatistiky

- súbor statistika.txt
- otvorenie
- čítanie kým existuje ďalší riadok
  - počet výhier – nextInt
  - počet prehíer – nextInt
  - názov obtiažnosti – next
  - meno hráča – nextLine
- zatvorenie

# Statistika – metóda nacitaj<sub>(1)</sub>

```
private void nacitaj() throws IOException {  
    File subor = new File("statistika.txt");  
    Scanner citac = new Scanner(subor);  
  
    while (citac.hasNextLine()) {  
        ...  
    }  
  
    citac.close();  
}
```

otvor

čítaj

zatvor

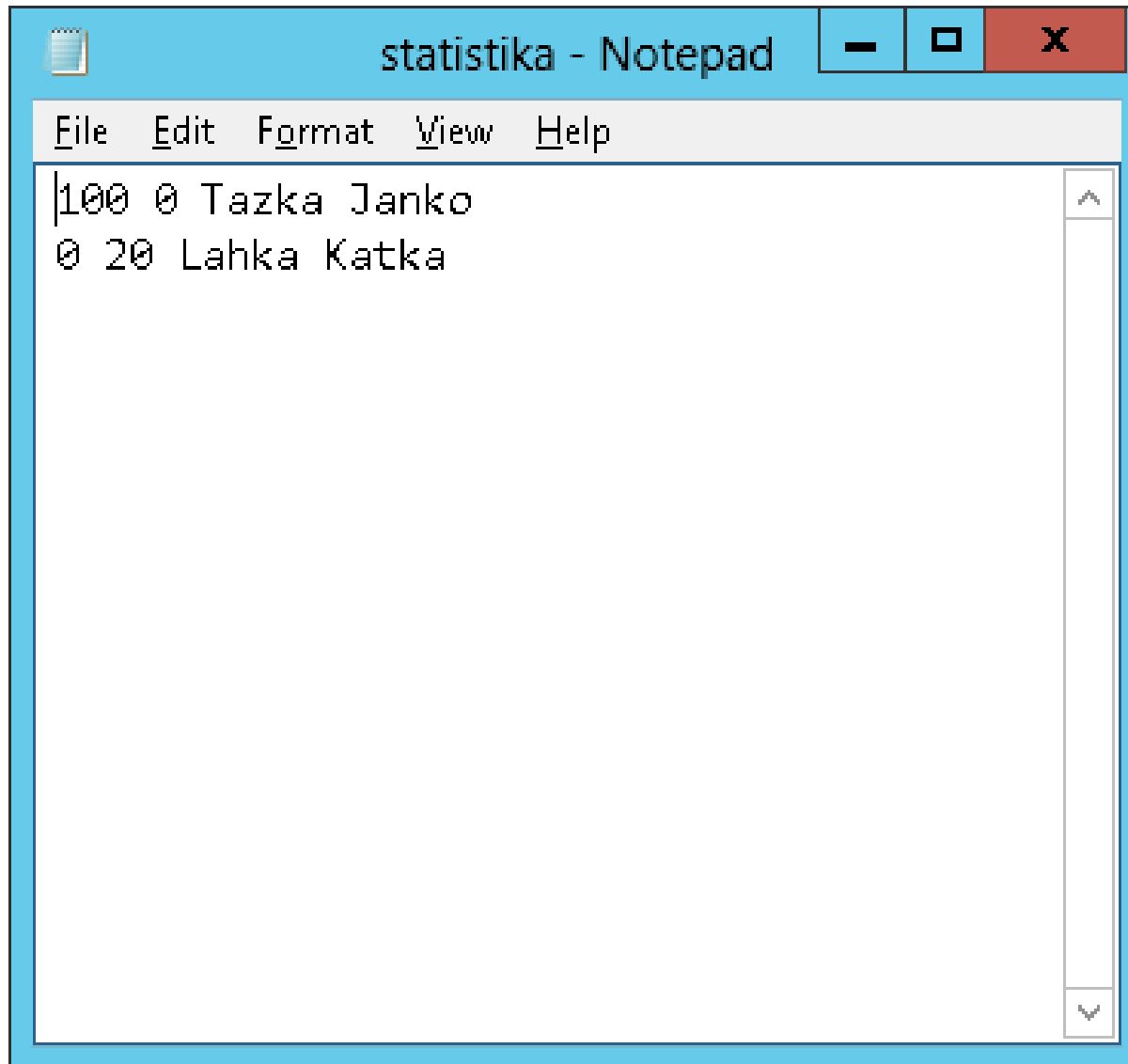
# Statistika – metóda nacitaj<sub>(2)</sub>

```
int pocetVyhier = citac.nextInt();  
int pocetPrehier = citac.nextInt();  
String obtiaznost = citac.next();  
String meno = citac.nextLine();
```

# Statistika – metóda nacitaj<sub>(3)</sub>

```
StatistikaHraca statistika =  
    new StatistikaHraca(meno, obtiaznost);  
  
for (int i = 0; i < pocetVyhier; i++) {  
    statistika.pridajHru(true);  
}  
  
for (int i = 0; i < pocetPrehier; i++) {  
    statistika.pridajHru(false);  
}  
  
this.statistiky.add(statistika);
```

# Štatistika v súbore



# Míny – zápis štatistiky

- súbor statistika.txt
- otvorenie
- pre každý záznam
  - počet výhier – print
  - počet prehíer – print
  - názov obtiažnosti – print
  - meno hráča – println
  - oddeľovač – medzera
- zatvorenie

# Statistika – metóda zapis<sub>(1)</sub>

```
private void zapis() throws IOException {    otvor  
    File subor = new File("statistika.txt");  
    PrintWriter zapisovac = new PrintWriter(subor);  
                                     zapíš  
    for (StatistikaHraca statistika : this.statistiky) {  
        ...  
    }  
  
    zapisovac.close(); zatvor  
}
```

# Statistika – metóda zapis<sub>(2)</sub>

```
zapisovac.print(statistika.getPocetVyhier());  
zapisovac.print(" ");  
zapisovac.println(statistika.getPocetPrehier());  
zapisovac.print(" ");
```

```
zapisovac.print(statistika.getNazovObtiaznosti());  
zapisovac.print(" ");  
zapisovac.print(statistika.getMenoHraca());
```



# Vďaka za pozornosť