

7

Polia objektov

Pojmy zavedené v 6. prednáške₍₁₎

- skupiny objektov – kontajnery
 - ArrayList
- generické triedy
- cyklus
 - for-each
 - while
- anonymné objekty

Cieľ prednášky

- kontajnery – polia
- obal'ovacie triedy
- cyklus for
- diagramy UML pre cykly

- príklad: Štatistika pripojení

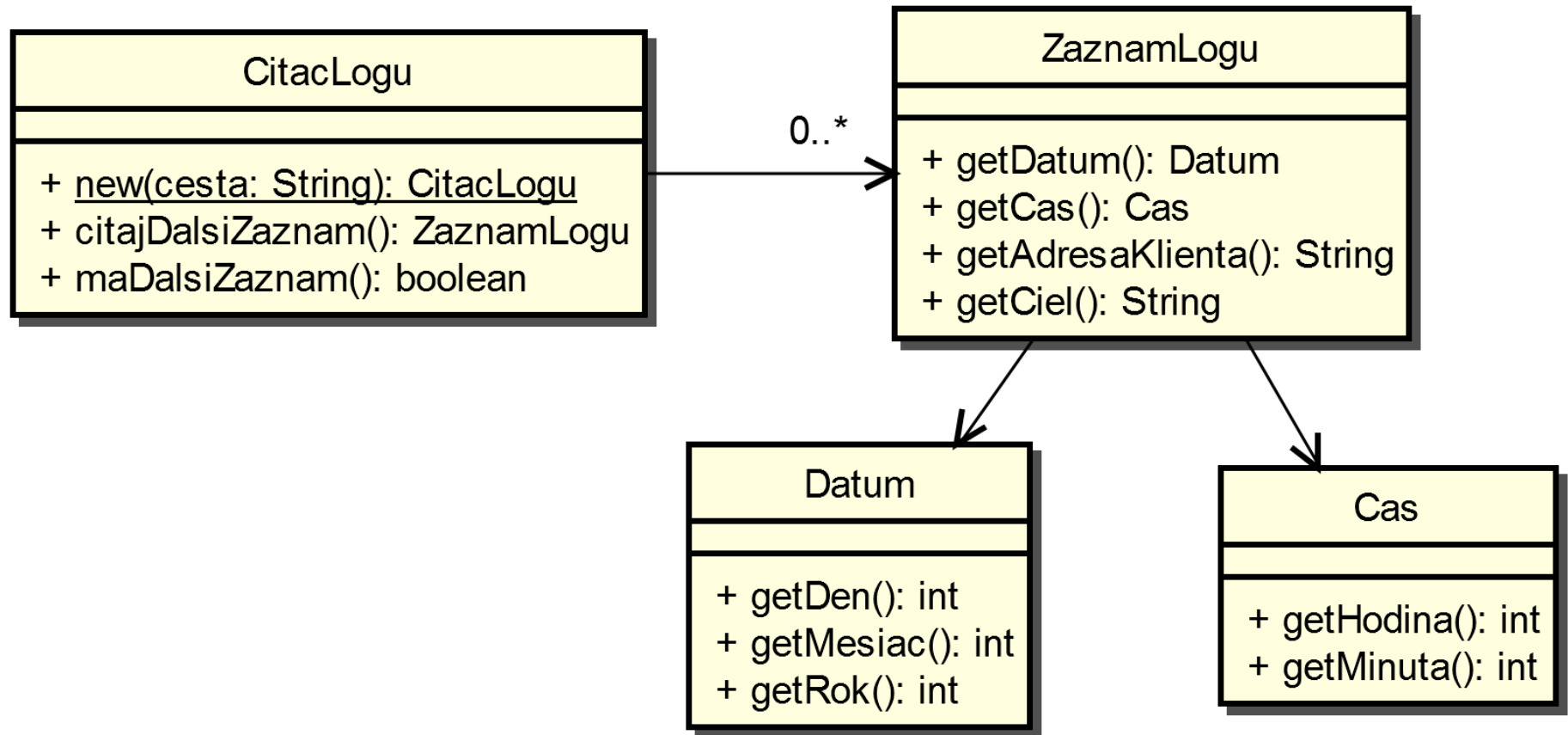
Štatistika pripojení na web server

- web server eviduje intenzitu pripájania
- pre každé pripojenie zaznamenáva
 - dátum a čas v tvare päťice celých čísel
 - rok mesiac deň hodina minúta
 - klient (adresa počítača)
 - adresa požiadavky
- máme k dispozícii súbor záznamov weblog.txt
- úloha – vytvoriť štatistiku pripojení podľa hodín (bez ohľadu na deň)

Log súbor

2010 09 01 07 45	166.254.109.216	/spravicky/16488
2010 09 01 08 40	173.137.190.129	/clanky/16763
2010 09 01 09 00	215.145.186.231	/clanky/7717
2010 09 01 09 50	190.232.201.202	/spravicky/43318
2010 09 01 10 04	109.97.94.95	/clanky/16189
2010 09 01 10 27	229.249.211.233	/clanky/13231
2010 09 01 10 59	165.246.158.207	/clanky/17016
2010 09 01 11 02	197.229.220.190	/clanky/22915
2010 09 01 11 04	217.220.209.238	/spravicky/13879
2010 09 01 11 06	85.128.39.62	/spravicky/13541
2010 09 01 11 35	186.246.148.205	/spravicky/23875
2010 09 01 11 40	244.162.121.146	/spravicky/49726
2010 09 01 11 44	109.97.94.95	/spravicky/38129
2010 09 01 11 45	96.201.235.232	/spravicky/42464
2010 09 01 11 49	180.184.204.217	/clanky/17001
2010 09 01 11 53	238.239.163.111	/clanky/6111

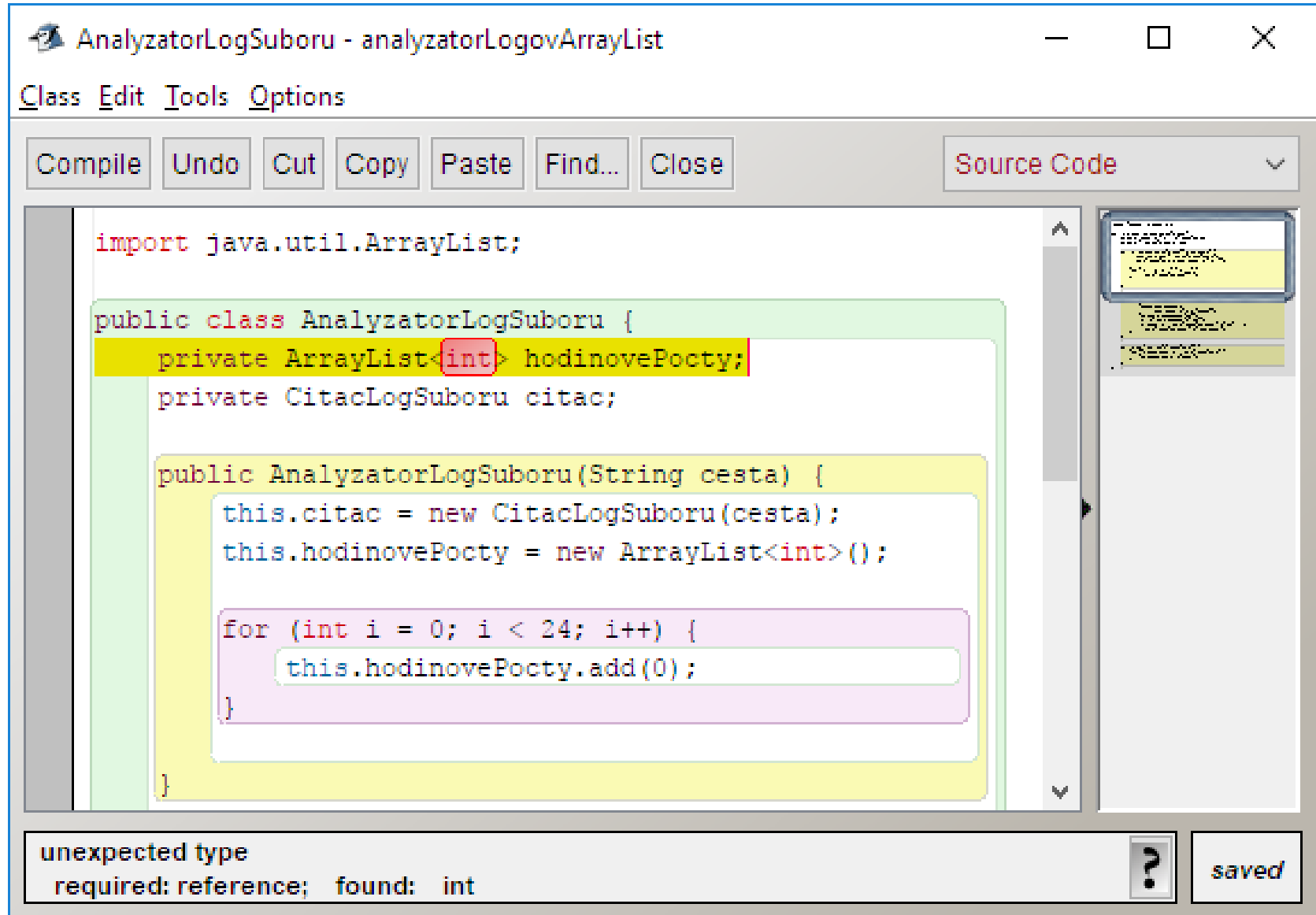
Práca s log súborom – rozhrania



Analýza

- potrebujeme ukladať údaje pre hodiny 0-23
 - počet požiadaviek v danú hodinu
- kontajner – ArrayList
 - prvky: celé čísla

Chyba pri vytváraní kontajnera



Problém s vytváraním kontajnera

- typ prvkov ArrayList – objektový typ
- naše prvky – čísla – primitívny typ
- riešenie – obaľovacia trieda

Obal'ovacia trieda CeleCislo

```
public class CeleCislo {  
    private int cislo;  
  
    public CeleCislo(int cislo) {  
        this.cislo = cislo;  
    }  
  
    public int getCislo() {  
        return this.cislo;  
    }  
}
```

Použitie obalovacej triedy

```
ArrayList<CeleCislo> cc = new ArrayList<CeleCislo>();  
cc.add(new CeleCislo(5));
```

```
// Vypíše 5
```

```
System.out.println(cc.get(0).getCislo());
```

Obalovacie triedy v jazyku Java

primitívny typ	obaľovacia trieda
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Použitie štandardnej obalovacej triedy

```
int celeInt = 15;
```

```
Integer obaleneCislo = new Integer(celeInt);
```

```
int druheCeleInt = obaleneCislo.intValue();
```

Automatické konverzie

- typová kompatibilita
 - primitívny typ \leftrightarrow obal'ovacia trieda
- boxing
 - prevod primitívneho typu na objektový
- unboxing
 - prevod objektového typu na primitívny

Boxing/unboxing

```
int celeInt = 15;
```

```
Integer obaleneCislo = celeInt;
```

```
int druheCeleInt = obaleneCislo;
```

```
obaleneCislo = obaleneCislo + 1;
```

Obal'ovacia trieda a ArrayList

```
ArrayList<Integer> ciska = new ArrayList<Integer>();
```

```
ciska.add(15);
```

```
ciska.add(cislo);
```


Analýza

- `ArrayList<Integer>` – kontajner na čísla
- hodiny 0 až 23 – 24 prvkov
- začiatočná hodnota každého prvku – 0
- vkladanie v cykle

AnalyzatorLogSuboru – rozhranie

AnalyzatorLogSuboru

- + new(cesta: String): AnalyzatorLogSuboru
- + analyzujData(): void
- + vytlacHodinovePocty(): void

Vnútorný pohľad

AnalyzatorLogSuboru

- hodinovePocty: ArrayList<Integer>
- citac: CitacLogSuboru

- + AnalyzatorLogSuboru(cesta: String)
- + analyzujData(): void
- + vytlacHodinovePocty(): void

Trieda AnalyzatorLogSuboru

```
import java.util.ArrayList;

public class AnalyzatorLogSuboru {
    private ArrayList<Integer> hodinovePocty;
    private CitacLogSuboru citac;

    ...
}
```

AnalyzatorLogSuboru – konštruktor₍₁₎

```
public AnalyzatorLogSuboru(String cesta) {  
    this.citac = new CitacLogSuboru(cesta);  
    this.hodinovePocty = new ArrayList<Integer>();  
  
    ...  
}
```

AnalyzatorLogSuboru – konštruktor₍₂₎

```
// inicializacia  
int i = 0;  
while (i < 24) {  
    this.hodinovePocty.add(0);  
    i++;  
}
```

Cyklus for

- ďalší z cyklov
- pravidlo:
 - inicializuj premennú cyklu na zadanú hodnotu
 - vykonávaj kým platí podmienka
 - príkazy tela cyklu
 - príkaz kroku

```
for (inicializácia; podmienka; krok) {  
    // telo cyklu  
}
```

Cyklus for – inicializácia

```
for (int i = 0; i < 24; i++) {  
    // telo cyklu  
}
```

- deklarácia a inicializácia premennej cyklu
- TypPrvku premennaCyklu = zaciatočnaHodnota
- zaciatočnaHodnota -> ľubovoľný výraz

Cyklus for – podmienka

```
for (int i = 0; i < 24; i++) {  
    // telo cyklu  
}
```

- relačný výraz
- všeobecne – logický výraz
- podmienka skončenia cyklu
 - false – cyklus končí

Cyklus for – krok

```
for (int i = 0; i < 24; i++) {  
    // telo cyklu  
}
```

- príkaz, ktorým meníme premennú cyklu

Operátor ++

```
premenna ++;
```

- je ekvivalentné

```
premenna = premenna + 1;
```

- operátor inkrementácie – zväčšenia
- aplikovateľný na všetky číselné typy

Operátor --

```
premenna --;
```

- je ekvivalentné

```
premenna = premenna - 1;
```

- operátor dekrementácie – zmenšenia
- aplikovateľný na všetky číselné typy

Cyklus for a operátor ++

```
for (int i = 0; i < 24; i++) {  
    // telo cyklu  
}
```

- je ekvivalent

```
for (int i = 0; i < 24; i = i + 1) {  
    // telo cyklu  
}
```

Cyklus for – ekvivalent cyklu while

```
for (int i = 0; i < 24; i++) {  
    this.hodinovePocty.add(0);  
}
```

- je ekvivalent

```
int i = 0;  
while (i < 24) {  
    this.hodinovePocty.add(0);  
    i++;  
}
```

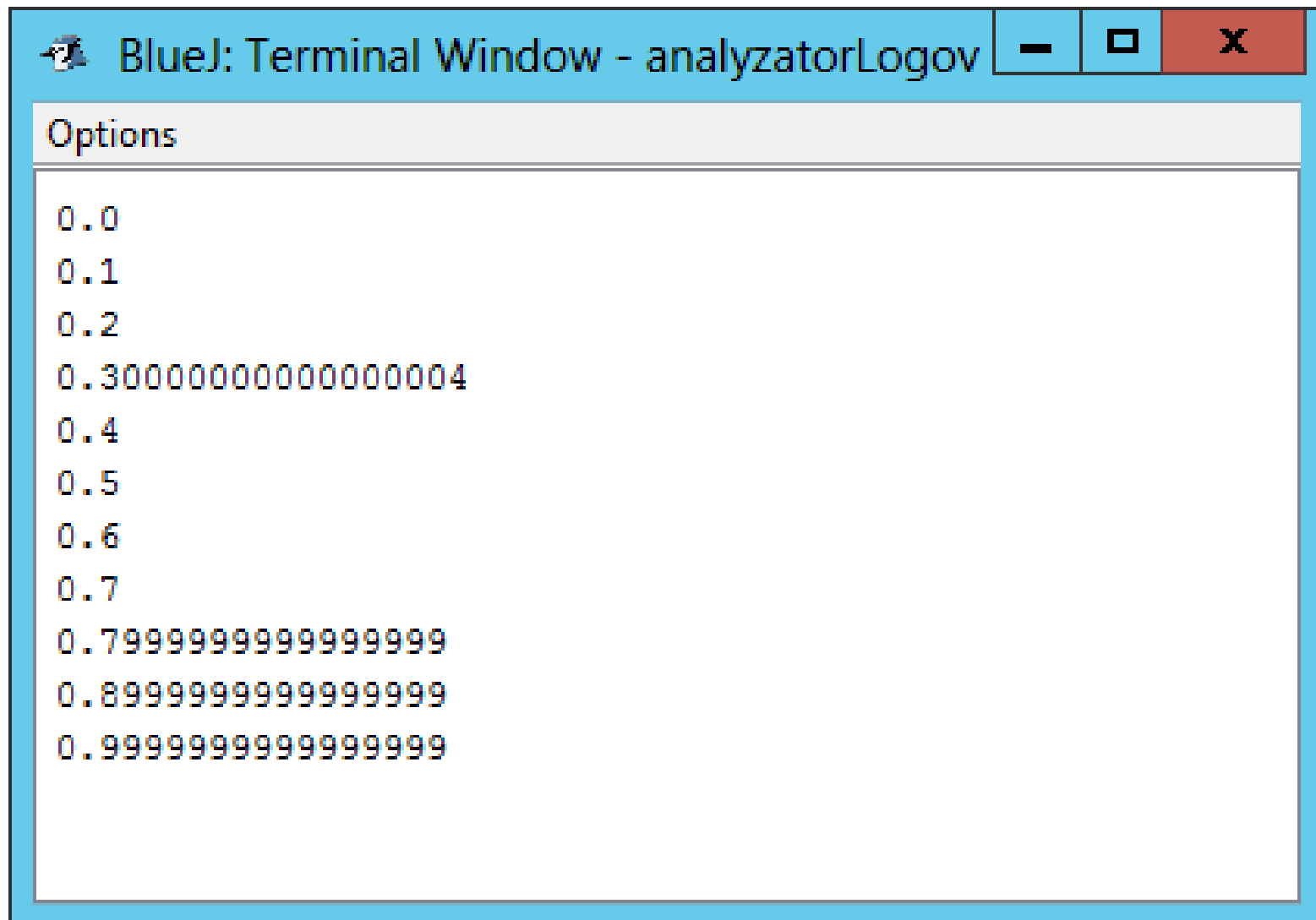
Cyklus for a double₍₁₎

- aký bude výstup?

```
for (double i = 0; i < 1; i = i + 0.1) {  
    System.out.println(i);  
}
```

- POZOR !!!

Cyklus for a double₍₂₎



```
BlueJ: Terminal Window - analyzatorLogov
Options
0.0
0.1
0.2
0.30000000000000000004
0.4
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
```


AnalyzatorLogSuboru – konštruktor

```
public AnalyzatorLogSuboru(String cesta) {  
    this.citac = new CitacLogSuboru(cesta);  
    this.hodinovePocty = new ArrayList<Integer>();  
  
    for (int i = 0; i < 24; i++) {  
        this.hodinovePocty.add(0);  
    }  
}
```

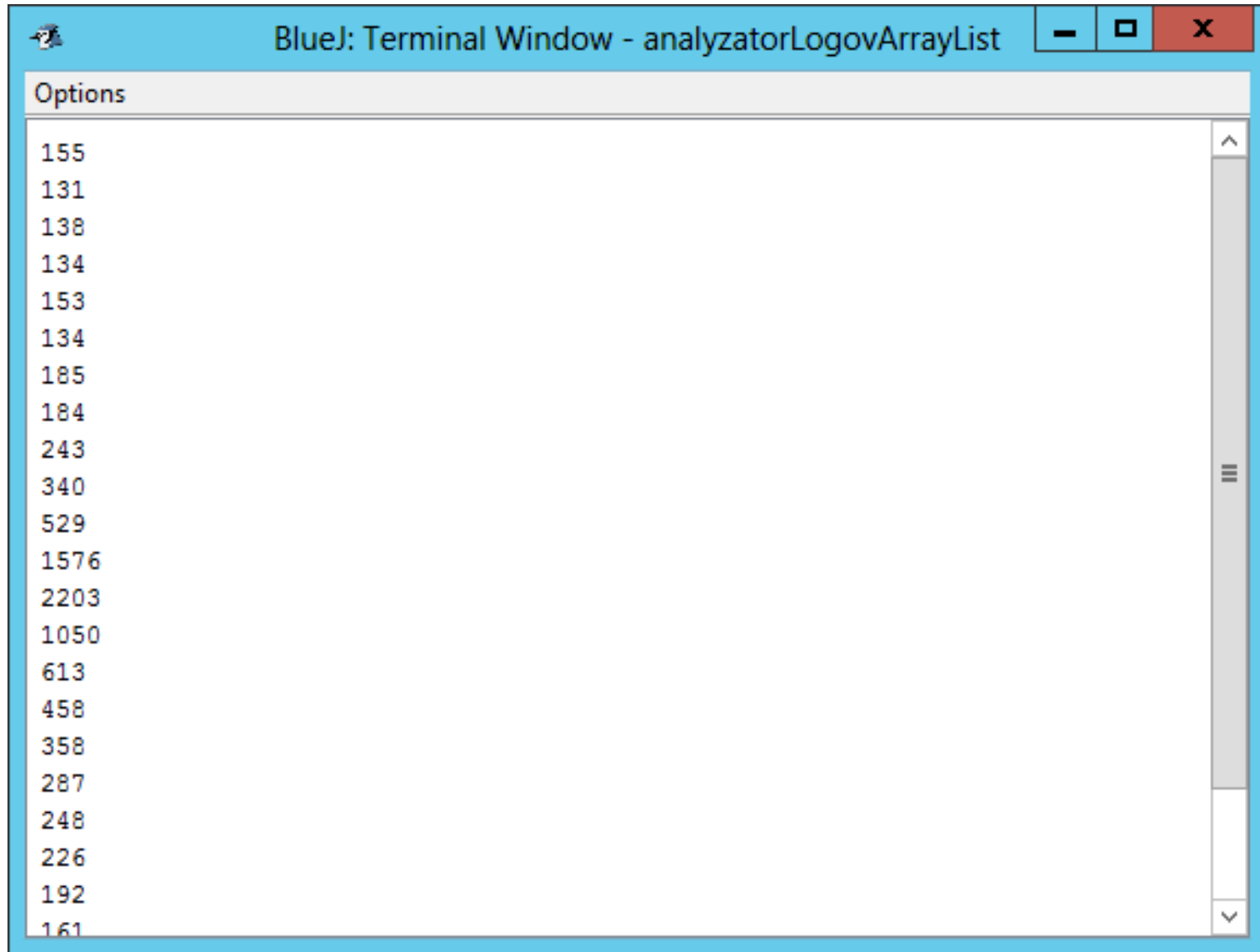
AnalyzatorLogSuboru – analyzujData

```
public void analyzujData() {  
    while (this.citac.maDalsiZaznam()) {  
        ZaznamLogu zaznam =  
            this.citac.citajDalsiZaznam();  
        Cas cas = zaznam.getCas();  
        int hodina = cas.getHodina();  
        int pocet = this.hodinovePocty.get(hodina) + 1;  
        this.hodinovePocty.set(hodina, pocet);  
    }  
}
```

Metóda vytlačHodinovePocty

```
public void vytlacHodinovePocty() {  
    for (Integer pocet : this.hodinovePocty) {  
        System.out.println(pocet);  
    }  
}
```

Výstup

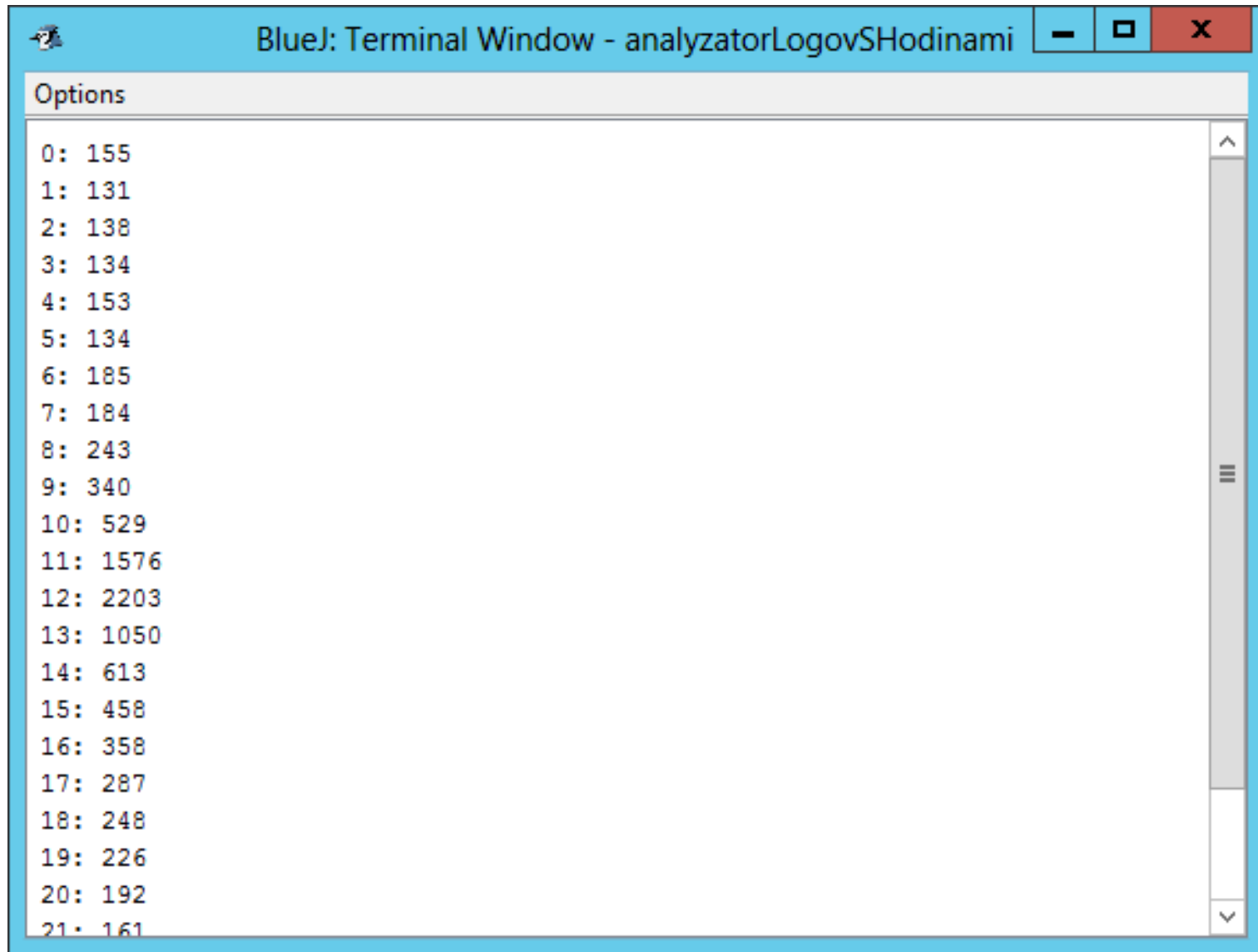


BlueJ: Terminal Window - analyzatorLogovArrayList

Options

```
155
131
138
134
153
134
185
184
243
340
529
1576
2203
1050
613
458
358
287
248
226
192
161
```

Požadovaný výstup



The screenshot shows a BlueJ Terminal Window titled "BlueJ: Terminal Window - analyzatorLogovSHodinami". The window contains a list of 22 numbers, each preceded by an index from 0 to 21. The numbers are: 155, 131, 138, 134, 153, 134, 185, 184, 243, 340, 529, 1576, 2203, 1050, 613, 458, 358, 287, 248, 226, 192, and 161. The window has a standard Windows-style title bar with minimize, maximize, and close buttons. A vertical scrollbar is visible on the right side of the text area.

```
Options
0: 155
1: 131
2: 138
3: 134
4: 153
5: 134
6: 185
7: 184
8: 243
9: 340
10: 529
11: 1576
12: 2203
13: 1050
14: 613
15: 458
16: 358
17: 287
18: 248
19: 226
20: 192
21: 161
```

Tlač s hodinami

```
public void vytlacHodinovePocty() {  
    int hodina = 0;  
    for (Integer pocet : this.hodinovePocty) {  
        String riadok = hodina + ": " + pocet;  
        System.out.println(riadok);  
        hodina++;  
    }  
}
```

Tlač s hodinami pomocou cyklu for

```
public void vytlacHodinovePocty() {  
    for (int hodina = 0; hodina < 24; hodina++) {  
        String riadok = hodina + ": " +  
                        this.hodinovePocty.get(hodina);  
        System.out.println(riadok);  
    }  
}
```

Kontejnery s fixným počtom prvkov

- Diár – príklad na použitie kontejnerov s premenlivým počtom prvkov
 - zmena počtu prvkov v priebehu životného cyklu
- existujú situácie, keď sa počet prvkov nemení
 - počet hodín dňa je konštantný
- kontejnery pre takéto situácie – polia
 - iná syntax – historické dôvody
 - pole – najstarší kontejner

Pole ako kontajner – rovnaké vlastnosti

- pole – objektový typ – referencia na pole
- hodnota null – aj pre pole
- práca s poľom ako celkom – referencia na pole
- môže obsahovať ľubovoľný typ prvkov

Pole ako kontajner – odlišné vlastnosti

- definícia poľa
- vytvorenie poľa
- prístup k prvkom
- nie je generický typ

Java – definícia poľa

```
typPrvkov[] menoPola;
```

- typ prvkov – ľubovoľný primitívny alebo objektový typ
- príklady:

```
int[] pocetPristupov;  
AutomatMHD[] automaty;
```

Java – vytvorenie poľa

```
menoPola = new typPrvkov[pocetPrvkov];
```

- správa new – špecifická forma
- prvky sú inicializované na hodnotu 0
- príklady:

```
pocetPristupov = new int[24];  
automaty = new AutomatMHD[2];
```

Java – definícia a vytvorenie poľa

```
typPrvkov[] menoPola = new typPrvkov[pocetPrvkov];
```

- spojenie definície a inicializácie do jedného príkazu
- príklady:

```
int[] pocetPristupov = new int[24];  
AutomatMHD[] automaty = new AutomatMHD[2];
```

Inicializácia poľa vymenovaním prvkov

```
typPrvkov[] menoPola = {zoznamPrvkov};
```

- zoznamPrvkov – čiarkou oddelený zoznam prvkov vytváraného poľa
- príklady:

```
int[] mince = {1, 2, 5, 10, 20, 50};  
AutomatMHD[] automaty = {ulSmrekova, ulPolna};  
AutomatMHD[] automaty = {  
    new AutomatMHD(50), new AutomatMHD(20)  
};
```

Java – prístup k prvkom poľa₍₁₎

- pomenovanie prvkov – meno poľa + index
– premenná

```
menoPola[indexPrvku]
```

- $0 \leq \text{index prvku} < \text{počet prvkov}$
- prvok poľa – premenná
- operácie – pravidlá pre typ prvkov
- index – celočíselný aritmetický výraz

Java – prístup k prvkom poľa₍₂₎

- príklad zápis do poľa:

```
pocetPristupov[0] = 5;  
automaty[0] = new Automat(50);
```

- príklad čítanie z poľa:

```
System.out.println(pocetPristupov[0]);  
automaty[0].vytlacListok();
```

- príklad kombinovaný:

```
pocetPristupov[0]++;
```


Java – pole a cyklus for

- dĺžka poľa = počet prvkov

`menoPola.length`

- výsledok – int

```
for (int i = 0; i < zoznam.length; i++) {  
    System.out.println(i + ": " + zoznam[i]);  
}
```

Java – pole a cyklus foreach

- pole je možné prechádzať cyklom foreach

```
for (int prvok : zoznam) {  
    System.out.println(prvok);  
}
```

Java – pole

- relačné operátory == a !=
 - porovnanie referencií
- príkaz priradenia
 - priradenie referencie, nie kópia všetkých prvkov poľa

```
int[] poleDruhe = pole;  
pole[1] = 5; // !!! zmenia sa obe polia
```

Trieda AnalyzatorLogSuboru

```
public class AnalyzatorLogSuboru {  
    private int[] hodinovePocty;  
    private CitacLogSuboru citac;  
  
    ...  
}
```

AnalyzatorLogSuboru – konštruktor

```
public AnalyzatorLogSuboru(String cesta) {  
    this.hodinovePocty = new int[24];  
    this.citac = new CitacLogSuboru(cesta);  
}
```

AnalyzatorLogSuboru – analyzujData

```
public void analyzujData() {  
    while (this.citac.maDalsiZaznam()) {  
        ZaznamLogu zaznam =  
                                this.citac.citajDalsiZaznam();  
        Cas cas = zaznam.dajCas();  
        int hodina = cas.dajHodinu();  
        this.hodinovePocty[hodina]++;  
    }  
}
```

Typy a polia

- definícia premennej

```
typPremennej nazovPremennej
```

- návratová hodnota

```
typ nazovMetody(parametre)
```

- typy
 - primitívne – int, float, boolean, ...
 - objektové – názov triedy: String
 - objektové – polia: typPrvku[]: int[], String[]

Pole a prvok poľa

```
int[] mince = {1, 2, 5, 10, 20, 50};  
AutomatMHD[] automaty = {ulSmrekova, ulPolna};
```

- int[] – int
 - mince – mince[3]
- AutomatMHD[] – AutomatMHD
 - automaty – automaty[1]
 - ulPolna – automaty
 - ulPolna – automaty[1]

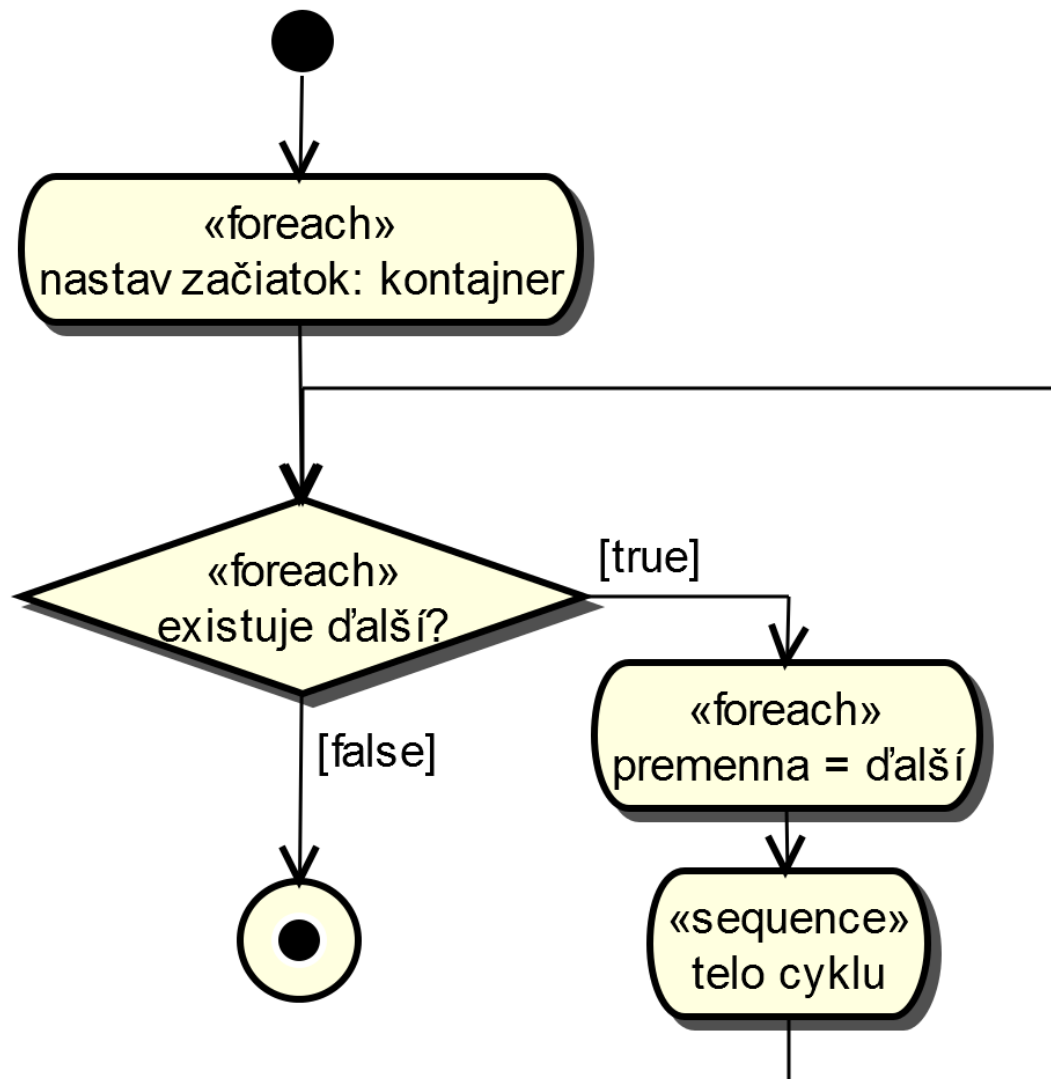
Cykly v UML

- Algoritmy – Diagramy aktivít
- Cykly
 - Žiadny špeciálny zápis
 - Modelovanie pomocou podmienky
 - Typ cyklu – stereotyp

Cyklus foreach

```
for (TypPrvku premenna : kontajner) {  
    // telo cyklu  
}
```

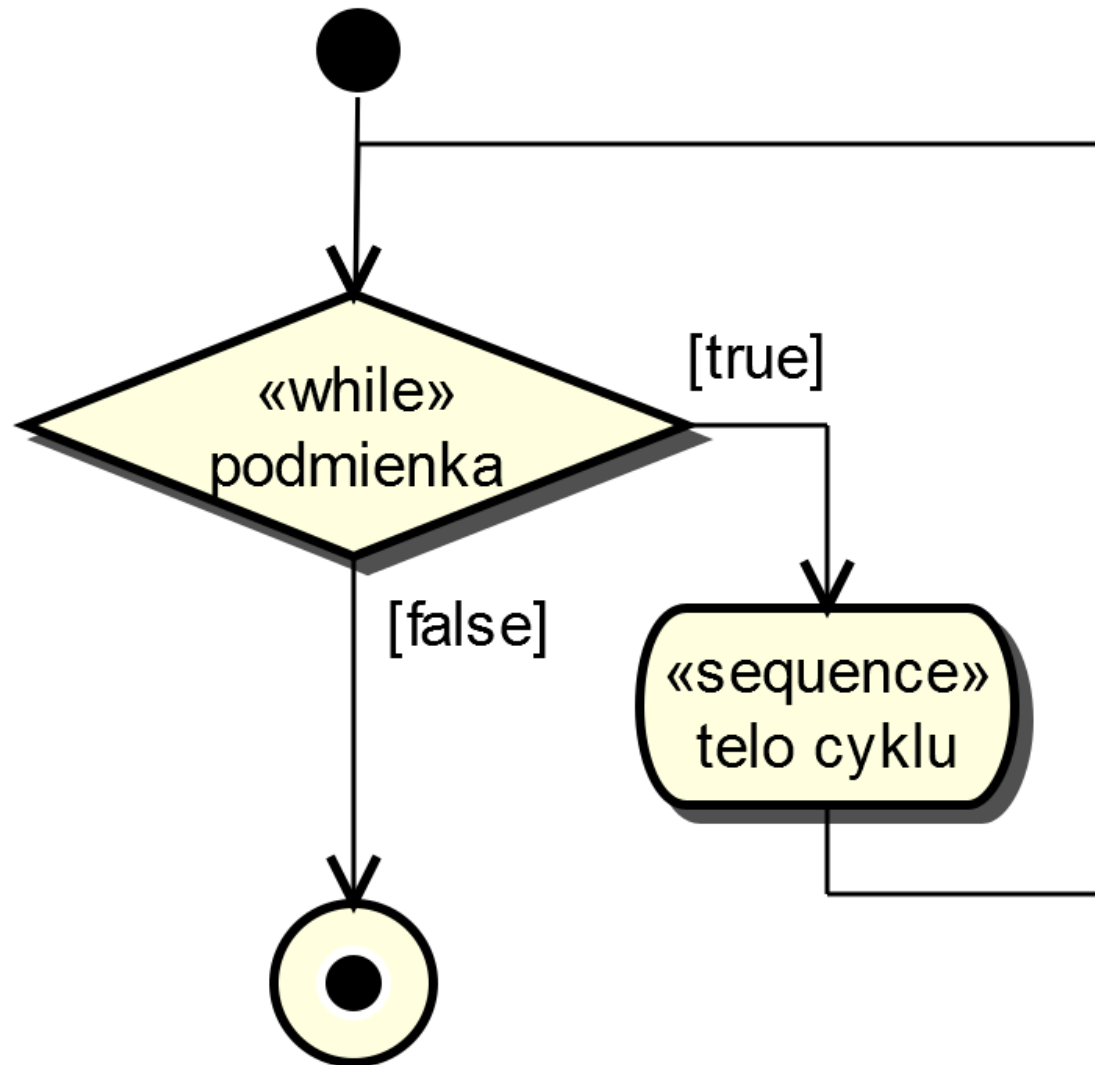
Cyklus foreach v UML



Cyklus while

```
while (podmienka) {  
    // telo cyklu  
}
```

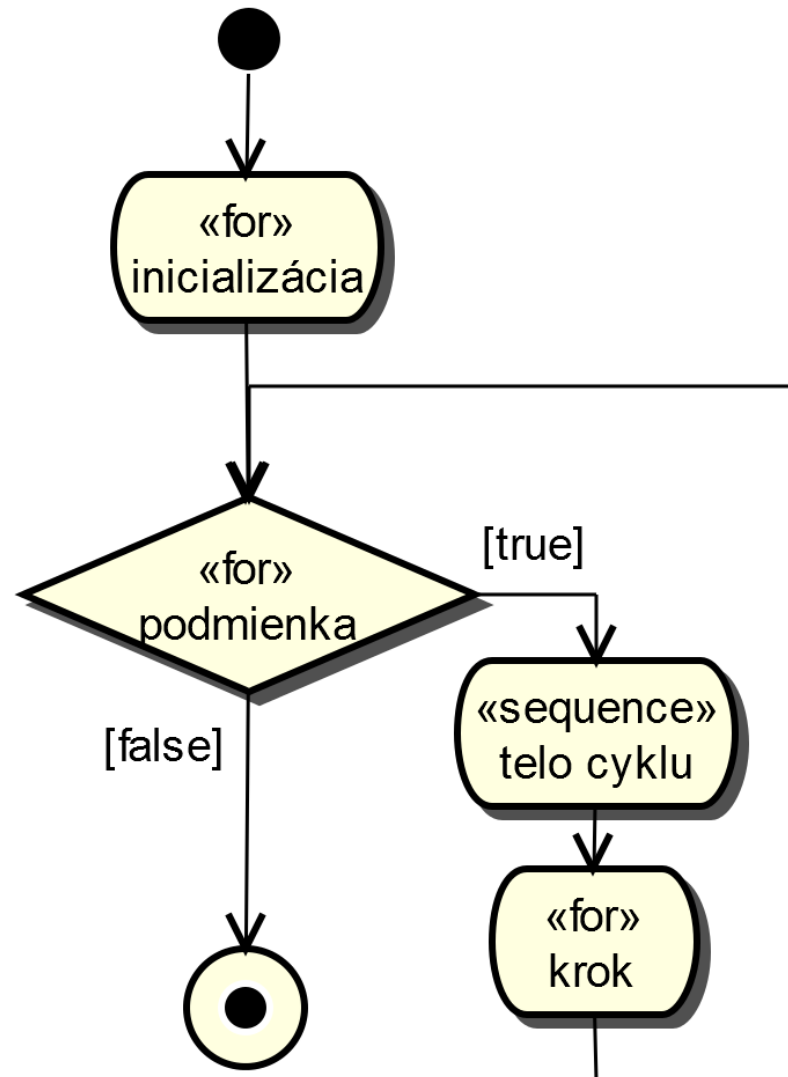
Cyklus while v UML



Cyklus for

```
for (inicializácia; podmienka; krok) {  
    // telo cyklu  
}
```

Cyklus for v UML

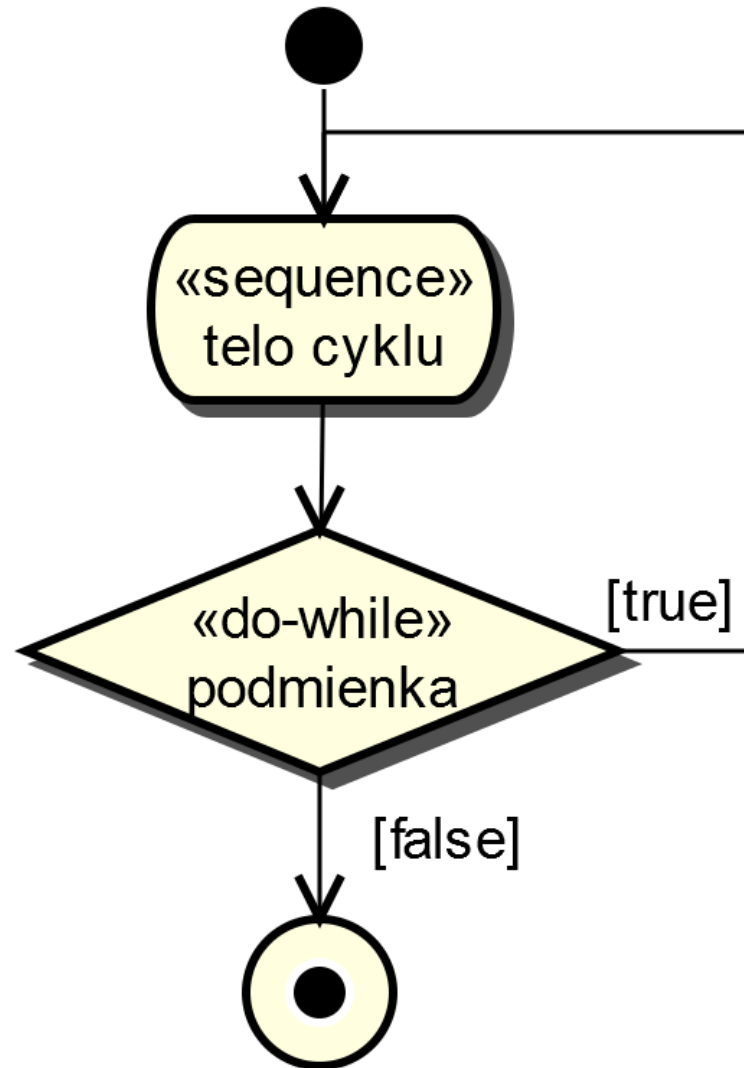


Cyklus do-while

- Další z cyklov
- Pravidlo:
 - vykonávajúj kým platí podmienka
 - telo sa vykoná aspoň jeden krát

```
do {  
    // telo cyklu  
} while (podmienka);
```


Cyklus do-while v UML



Vďaka za pozornosť