

# Informatika 1

## Algoritmus



# Pojmy zavedené v 2. prednáške (1)

- trieda ako
  - inštancia metatriedy
  - priamo definovaný objekt – Java
- delenie jazykov
  - založené na objektoch
  - založené na triedach – Java
- UML – grafické znázornenie triedy
  - názov, atribúty, metódy
  - rôzne pohľady podľa účelu
- UML – grafické znázornenie inštancie

# Pojmy zavedené v 2. prednáške (2)

- definícia triedy (trieda ako šablóna) – Java
  - definícia atribútov
  - definícia konštruktorov
  - definícia metód
- parametre konštruktorov a metód
  - formálne – v definícii metódy
  - skutočné – v správe
- návratová hodnota metódy
  - typ void

# Pojmy zavedené v 2. prednáške (3)

- primitívne dátové typy
  - pre celé čísla
  - pre reálne čísla
  - znaky
  - logické hodnoty
- objektový typ String
- literály
  - primitívnych dátových typov
  - String

# Pojmy zavedené v 2. prednáške (4)

- identifikátory
  - pravidlá jazyka Java
  - konvencie Java

# Pojmy zavedené v 2. prednáške (5)

- priradovací príkaz
  - výraz, operátor priradenia
- príkaz return
- príkazy pre tlač do okna terminálu
- lokálna premenná
- komentáre
  - jednoriadkové - // komentár
  - viacriadkové /\* komentár \*/
    - dokumentačné /\*\* komentár \*/

# Cieľ prednášky

- aritmetický výraz
- algoritmus
  - definícia a vlastnosti
  - základné konštrukčné prvky
    - vetvenie
    - cyklus
  - znázornenie v UML
- Príklad: automat na cestovné lístky

# Aritmetický výraz

- predpis na výpočet číselnej hodnoty
- obvykle má tvar matematického výrazu
- aritmetický výraz môže mať formu:
  - bez operátorov
    - číselný literál
    - číselný parameter
    - číselný atribút
  - s aritmetickým operátorom
    - unárnyOperátor operand
    - operand binárnyOperátor operand



# Príklady výrazov

**this**.trzba + **this**.cenaListka

**this**.vlozenaCiastka - **this**.cenaListka

0

cenaListka

**this**.pocetPredanychListkov + 1

# Unárne aritmetické operátory

- tvar:

operátor operand
------------------

- operand môže byť ľubovoľný aritmetický výraz
- unárne aritmetické operátory:
  - - (mínus) – unárne mínus – zmena znamienka
  - + (plus) – unárne plus

# Binárne aritmetické operátory

- tvar:

prvýOperand operátor druhýOperand

- operand môže byť ľubovoľný aritmetický výraz
- binárne aritmetické operátory:
  - + (plus) – súčet
  - - (mínus) – rozdiel
  - \* (hviezdička) – súčin
  - / (lomka) – podiel
  - % (percento) – zvyšok po delení

# Priorita operátorov (1)

- operátory sa vyhodnocujú v nasledujúcom poradí:
  - unárne  $+$ ,  $-$
  - binárne  $*$ ,  $/$ ,  $\%$
  - binárne  $+$ ,  $-$
- teda rovnako ako v matematike

## Priorita operátorov (2)

- poradie vyhodnocovania sa dá ovplyvňovať zátvorkami – spôsobom obvyklým v matematike
- zátvorky treba používať vždy, keď to zlepší čitateľnosť výrazu

# Zátvorky

$$\frac{a.b}{c.d}$$

~~$a * b / c * d$~~

$(a * b) / (c * d)$

# Typ hodnoty aritmetického výrazu

- operand v aritmetickom výraze môže reprezentovať len číselnú hodnotu

operandy	byte, short, int	long	float	double
byte, short, int	int	long	float	double
long	long	long	float	double
float	float	float	float	double
double	double	double	double	double

# Typová kompatibilita – príkaz návratu

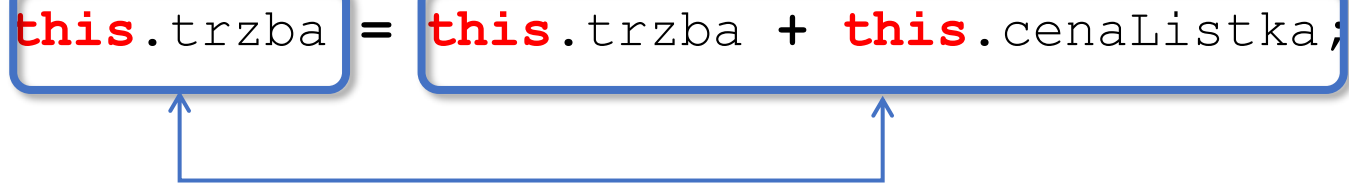
```
public int getCenaListka() {  
    return this.cenaListka;  
}
```

typ výrazu musí byť konvertovateľný na typ návratovej hodnoty



# Typová kompatibilita – priradenie

```
this.trzba = this.trzba + this.cenaListka;
```



typ výrazu na pravej strane priradovacieho príkazu musí byť konvertovateľný na typ ľavej strany priradovacieho príkazu

# Typová kompatibilita – konverzie

- nazývané aj implicitné pretypovanie
- prvý stĺpec – cieľový typ konverzie
- prvý riadok – zdrojový typ konverzie

operandy	byte	short	int	long	float	double
byte	A	N	N	N	N	N
short	A	A	N	N	N	N
int	A	A	A	N	N	N
long	A	A	A	A	N	N
float	A	A	A	A	A	N
double	A	A	A	A	A	A

# Typová kompatibilita príklad

```
private int hodnotaInt;  
private byte hodnotaByte;  
private double hodnotaDouble;  
  
this.hodnotaInt = 5;  
  
this.hodnotaInt = this.hodnotaByte;  
  
this.hodnotaInt = this.hodnotaDouble;
```

# Explicitná konverzia

- (explicitné pretypovanie)
- použiteľná v prípade, keď konverziu prekladač nedovolí
- programátor sa zaručí, že pretypovanie bude fungovať správne

```
this.hodnotaInt = (int) this.hodnotaDouble;
```

# Algoritmy

- telo konštruktora obsahuje tie príkazy, ktoré predstavujú inicializáciu práve vytváratej inštancie
- telá metód obsahujú tie príkazy, ktoré urobí objekt ako reakciu na prijatie rovnomennej správy
- telá metód a konštruktorov predstavujú algoritmy

# Algoritmus

- algoritmus

- popis pracovného postupu, ktorým sa rieši určitá skupina úloh.
- presne definovaná konečná postupnosť príkazov (krokov), vykonávaním ktorých pre každé prípustné vstupné hodnoty získame po konečnom počte krokov odpovedajúce výstupné hodnoty.

- algoritmizácia

- tvorivý proces hľadania a vytvárania algoritmu.

# Vlastnosti algoritmu

- determinovanosť – po vykonaní každého kroku musí byť jednoznačne určený krok nasledujúci
- rezultatívnosť – pre rovnaké vstupné údaje musí algoritmus dať rovnaké výstupné údaje
- konečnosť – vykonávanie algoritmu má vždy konečný počet krokov
- hromadnosť – nie je riešením jedinej úlohy, ale všetky úlohy danej kategórie, ktoré sa líšia len hodnotami vstupných údajov

# Processor

- algoritmus vykonáva objekt, ktorý sa nazýva procesor.
- algoritmus musí byť vyjadrený v jazyku, ktorému procesor rozumie a vie vykonávať príkazy zapísané pomocou toho jazyka.
- predpokladom je, že procesor „neuvažuje“ – príkazy algoritmu vykonáva mechanicky.



# Vyjadrenie algoritmu

- postup vyjadrený v prirodzenom jazyku – procesorom je človek.
- znázornený graficky v podobe diagramu – používa najčastejšie človek, keď sa chce vyjadriť nezávisle od prirodzeného jazyka autora.
- zapísaný v programovacom jazyku – ak je procesorom počítač.

# Štruktúrované programovanie

- doteraz – všeobecne o algoritmoch
- ich zápis môže mať rôzne formy v rôznych programovacích jazykoch
  - neštruktúrované programovanie
  - štruktúrované programovanie
  - deklaratívne programovanie
  - funkcionálne programovanie
  - ...
- odteraz len štruktúrované programovanie
- podporuje ho aj Java

# Základné konštrukčné prvky

- prvkami sú príkazy
- jednoduché
  - priradovací príkaz
  - príkaz návratu
- štruktúrované
  - postupnosť (sekvencia)
  - vetvenie
  - cyklus

# Štruktúrované príkazy

- postupnosť (sekvencia)
  - určuje poradie vykonávania príkazov
  - príkazy môžu byť jednoduché aj štruktúrované
- vetvenie – výber jednej alternatívy
  - alternatívne vetvy algoritmu
  - vetva sa uplatní, ak je splnená podmienka
- cyklus – opakovanie časti algoritmu
  - opakovaná časť – telo cyklu
  - opakovanie na základe podmienky cyklu

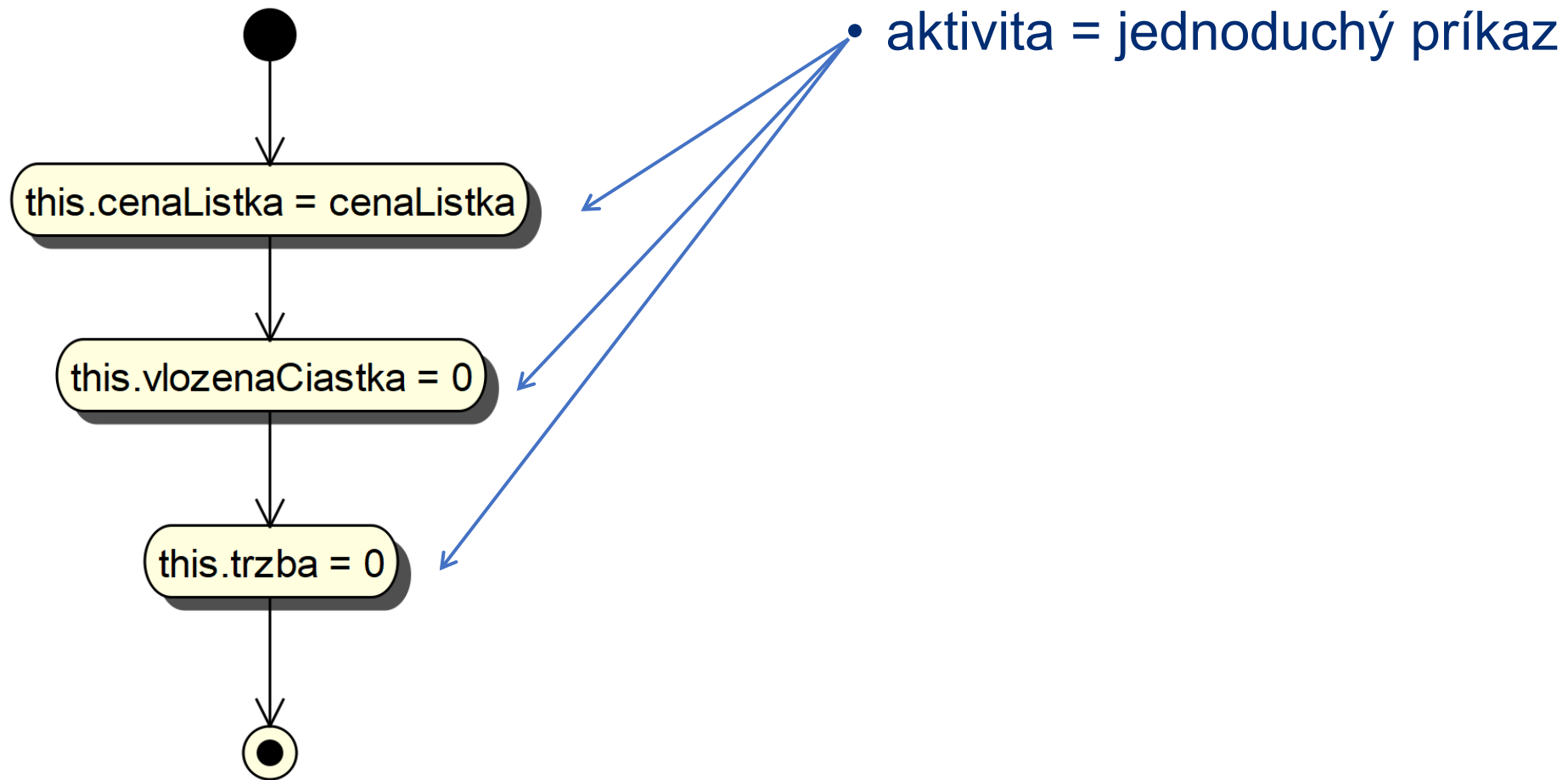
# Primitívny automat MHD

- telá konštruktora a všetkých metód sú postupnosti (sekvencie) jednoduchých príkazov.
- príklad – konštruktor
- telo konštruktora tvoria 3 príkazy – nastavenie automatu do začiatočného stavu.

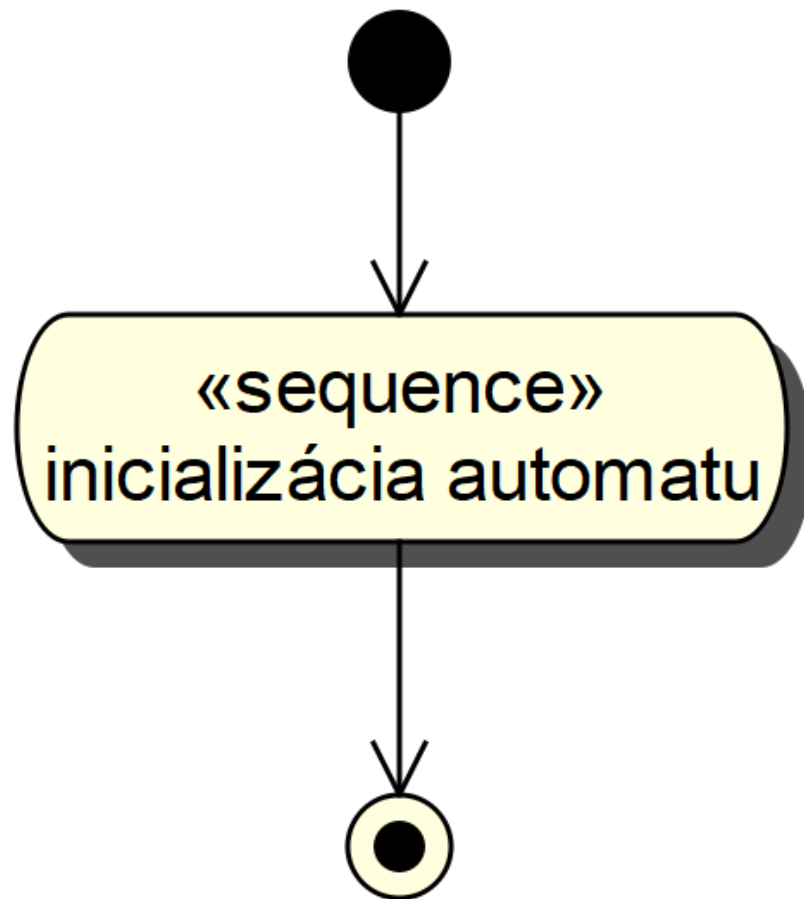
# Java – postupnosť príkazov

```
this.cenaListka = cenaListka;  
this.vlozenaCiastka = 0;  
this.trzba = 0;
```

# Diagram aktivít UML – postupnosť (1)



## Diagram aktivít UML – postupnosť (2)



- aktivita = príkaz – sekvencia



# Vetvenie

- prostriedok na rozhodovanie v algoritme
- jedna alebo skupina podmienok
  - ich vyhodnotením sa určí, aké príkazy má procesor v danej situácii vykonávať
- Java
  - úplný príkaz if
  - neúplný príkaz if
  - príkaz switch

# Vetvenie v jazyku Java – úplný príkaz if

```
if (podmienka) {  
    // príkazy vetvy, ak podmienka platí (true)  
} else {  
    // príkazy vetvy, ak podmienka neplatí (false)  
}  
// príkazy, ktoré sa vykonajú vždy
```

# Vetvenie v jazyku Java – príkaz if

```
if (podmienka) {  
    // príkazy vetvy, ak podmienka platí (true)  
}  
// príkazy, ktoré sa vykonajú vždy
```

# Formulácia podmienky

- podobná matematickej forme
- logický výraz – jeho hodnota je typu boolean
- príklad:

```
hodnotaMince > 0
```

- > (väčší ako) – relačný operátor
- ak je relácia splnená, výsledok má hodnotu true
- inak má hodnotu false

# Relačné operátory (1)

- sú vždy binárne

```
prvýOperand operátor druhýOperand
```

- oba operandy sú aritmetické výrazy
- priorita relačných operátorov je nižšia ako priorita aritmetických operátorov
- výsledok logického výrazu je vždy typu boolean

## Relačné operátory (2)

matematika	Java
$x > y$	<code>x &gt; y</code>
$x \geq y$	<code>x &gt;= y</code>
$x < y$	<code>x &lt; y</code>
$x \leq y$	<code>x &lt;= y</code>
$x = y$	<code>x == y</code>
$x \neq y$	<code>x != y</code>

(font Cascadia Code, ale nefunguje v BlueJ)

# Blok

- časť kódu programu uzavretá do dvojice zátvoriek {}
- bloky sa do seba vnášajú
- telo triedy obsahuje bloky – telá konštruktorov a metód
- vetva v príkaze if tiež môže byť blok

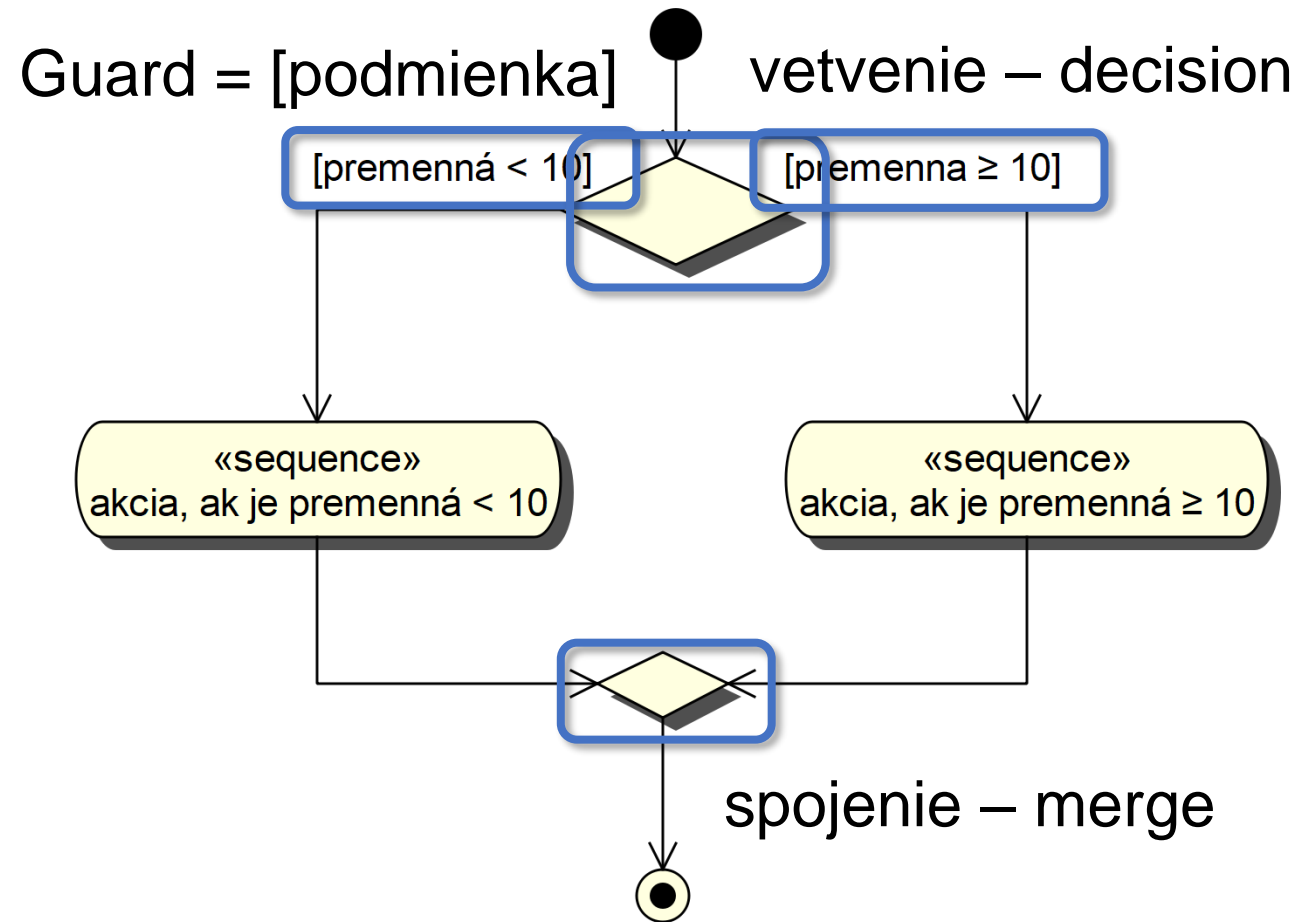
# Príkaz if bez bloku

```
if (podmienka)
    // jeden príkaz - podmienka platí (true)
else
    // jeden príkaz - podmienka neplatí (false)
// tu už začína nasledujúci príkaz
```

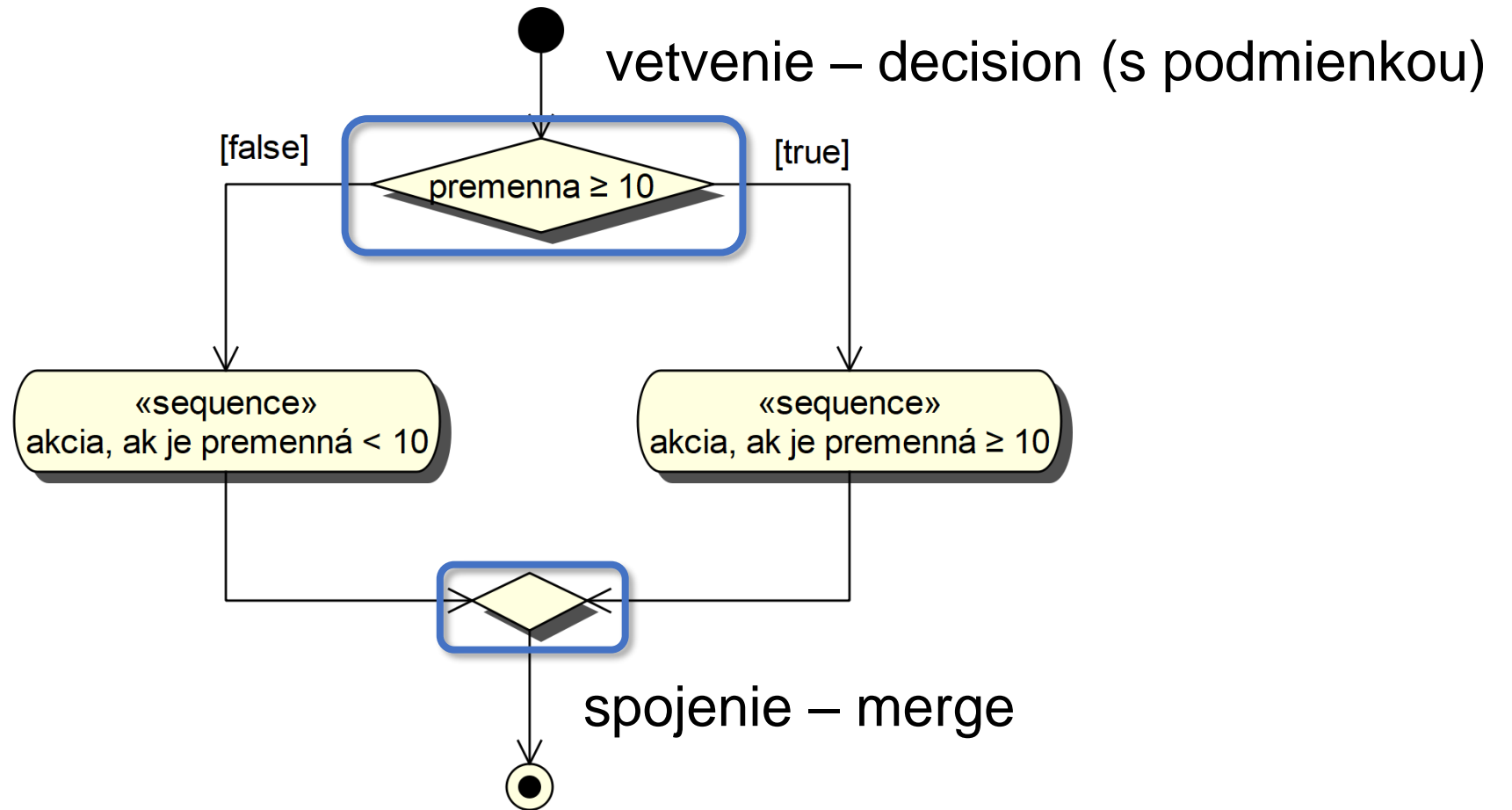
```
if (podmienka)
    // jeden príkaz - podmienka platí (true)
// tu už začína nasledujúci príkaz
```



# Vetvenie v diagrame aktivít



# Vetvenie v diagrame aktivít – iný zápis



# Príklad

- doplnenie kontroly, či vložená čiastka pokrýva cenu lístka
- treba sa rozhodnúť
  - ak vložená čiastka dosahuje cenu lístka
    - lístok sa vytlačí
  - ak vložená čiastka nedosahuje cenu lístka
    - lístok sa nevytlačí, vypíše sa dôvod

# Metóda tlacListok

```
public void tlacListok() {  
    if (this.cenaListka <= this.vlozenaCiastka) {  
        // tlac listka - vynechane prikazy  
        this.trzba = this.trzba + this.cenaListka;  
        this.vlozenaCiastka = this.vlozenaCiastka - this.cenaListka;  
    } else {  
        System.out.println("Ciastka je mensia ako cena.");  
    }  
}
```

# Ukončenie algoritmu pomocou príkazu return

- príkaz return – posledný vykonaný príkaz v algoritme
- možnosť skombinovať s vetvením
  - podmienené predčasné ukončenie algoritmu
- dá sa využiť aj vo void metóde
  - bez parametra – neoznačuje návratovú hodnotu
- !!! používať iba na zjednodušenie programu

## Metóda tlacListok po zjednodušení s return

```
public void tlacListok() {  
    if (this.cenaListka > this.vlozenaCiastka) {  
        System.out.println("Ciastka je mensia ako cena.");  
        return;  
    }  
  
    // tlac listka - vynechane prikazy  
    this.trzba = this.trzba + this.cenaListka;  
    this.vlozenaCiastka = this.vlozenaCiastka - this.cenaListka;  
}
```

# Iný príklad – vetvenie v konštruktore

- doplnenie kontroly, či je zadaná cena lístka kladná
- použijeme vetvenie
  - 1. vetva
    - podmienka: cena lístka  $> 0$
    - cena sa použije na inicializáciu atribútu
  - 2. vetva
    - podmienka: cena lístka  $\leq 0$
    - ???
    - atribút sa inicializuje preddefinovanou hodnotou

# Konštruktor AutomatMHD

```
public AutomatMHD(int cenaListka) {  
    if (cenaListka > 0) {  
        this.cenaListka = cenaListka;  
    } else {  
        this.cenaListka = 100;  
    }  
    this.vlozenaCiastka = 0;  
    this.trzba = 0;  
}
```



# Viaccestné vetvenie – príkaz switch

- príkaz switch – viaccestné vetvenie

```
switch (výraz) {  
    // možnosti a vetvy  
}
```

- výraz
  - musí byť typu byte, short, int, char, alebo String
  - porovnáva s možnosťami

# Príkaz swich – case (návestie)

- možnosti musia byť konštantné výrazy
  - konvertovateľné na typ výrazu v príkaze switch

```
case moznost1:  
case moznost2:  
...  
    // príkazy vetvy
```

# Príkaz switch – default (návestie)

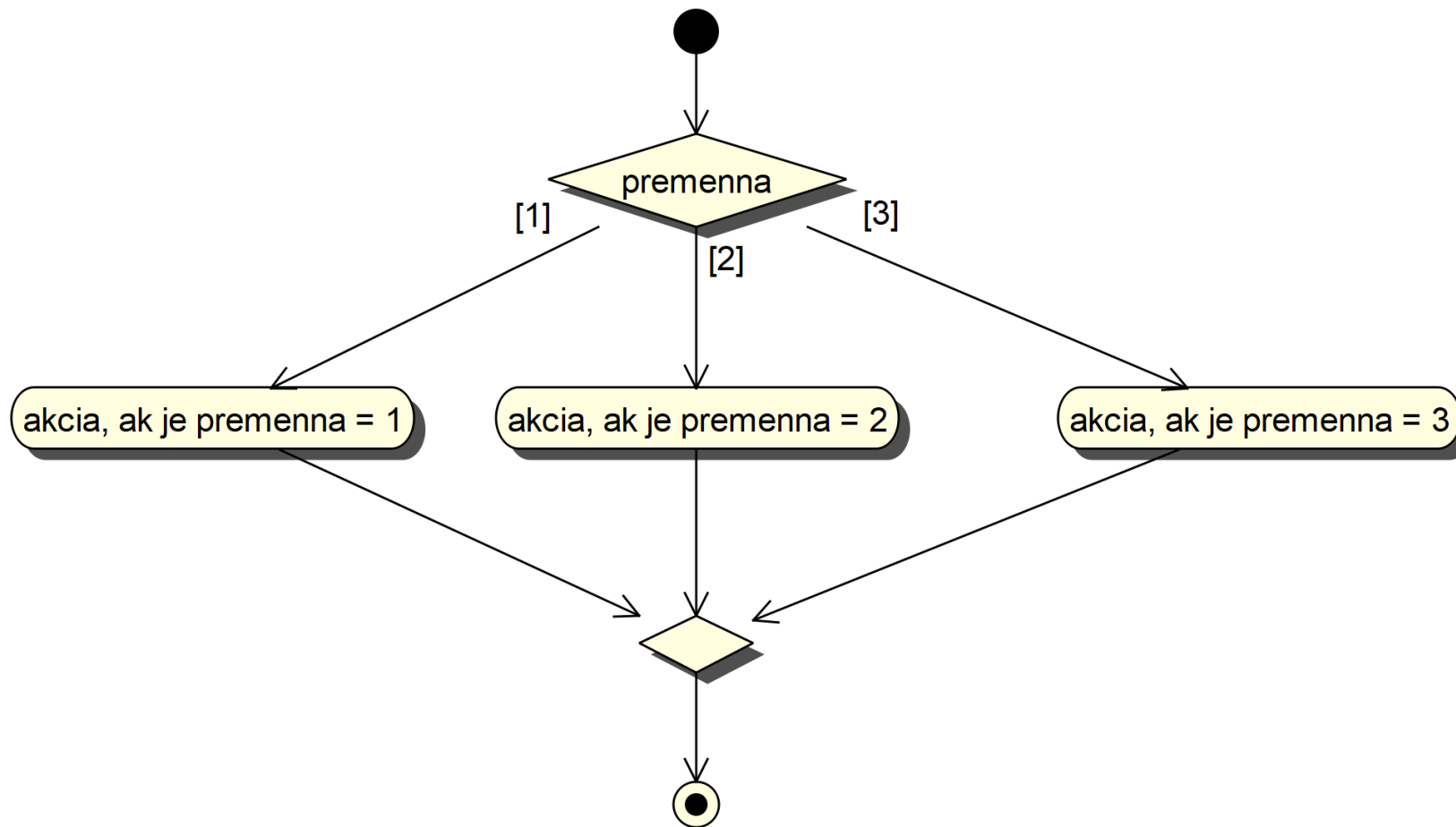
- default – vetva, ktorá sa uplatní, ak sa nenájde príslušná možnosť
- obdoba else v príkaze if

```
default:  
    // príkazy vetvy
```

# Ukončenie vetvy

- každá vetva má byť ukončená
  - príkaz return – ukončenie vykonávania metódy
  - príkaz break – ukončenie vykonávania príkazu switch
- chýbajúce ukončenie
  - !!! prekladač jazyka Java neupozorní
  - vykonávanie pokračuje ďalšou vetvou

# Viaccestné vetvenie v diagrame aktivít



# Príklad

- chceme polovičné, alebo dvoj pásmové lístky
- rozhodovanie medzi viac možnosťami

# Získanie ceny lístka v metóde tlač lístok (1)

```
public void tlacListok(String typ) {  
    int cenaListka;  
    switch (typ) {  
        case "zakladny":  
            cenaListka = this.cenaListkaZakladna;  
            break;  
        case "studentsky":  
        case "polovicny":  
            cenaListka = this.cenaListkaZakladna / 2;  
            break;  
        ...  
    }
```

## Získanie ceny lístka v metóde tlač lístok (2)

```
...  
    case "dvojpasmovy":  
        cenaListka = this.cenaListkaDvojpasmove;  
        break;  
    case "studentsky dvojpasmovy":  
    case "polovicny dvojpasmovy":  
        cenaListka = this.cenaListkaDvojpasmove / 2;  
        break;  
    default:  
        System.out.println("Nespravny typ listka.");  
        return;  
}  
...
```



# Cyklus

- zopakovanie časti algoritmu podľa zadaných pravidiel
  - vypísanie všetkých poznámok v diári
  - sčítanie čísel od 1 po dané číslo
  - vyhľadanie knihy v knižnici
- rôzne typy cyklov – rôzne pravidlá
- pravidlá sa vyhodnocujú počas vykonávania algoritmu procesorom

# Cyklus for

- pravidlo:
  - inicializuj premennú cyklu na zadanú hodnotu
  - vykonávaj kým platí podmienka
    - príkazy tela cyklu
    - príkaz kroku
- využíva sa na pevný počet opakovaní

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

# Cyklus for – inicializácia

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

- Definícia a inicializácia premennej cyklu
- TypPrvku premennaCyklu = zaciatoznaHodnota
- zaciatoznaHodnota -> 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 ++

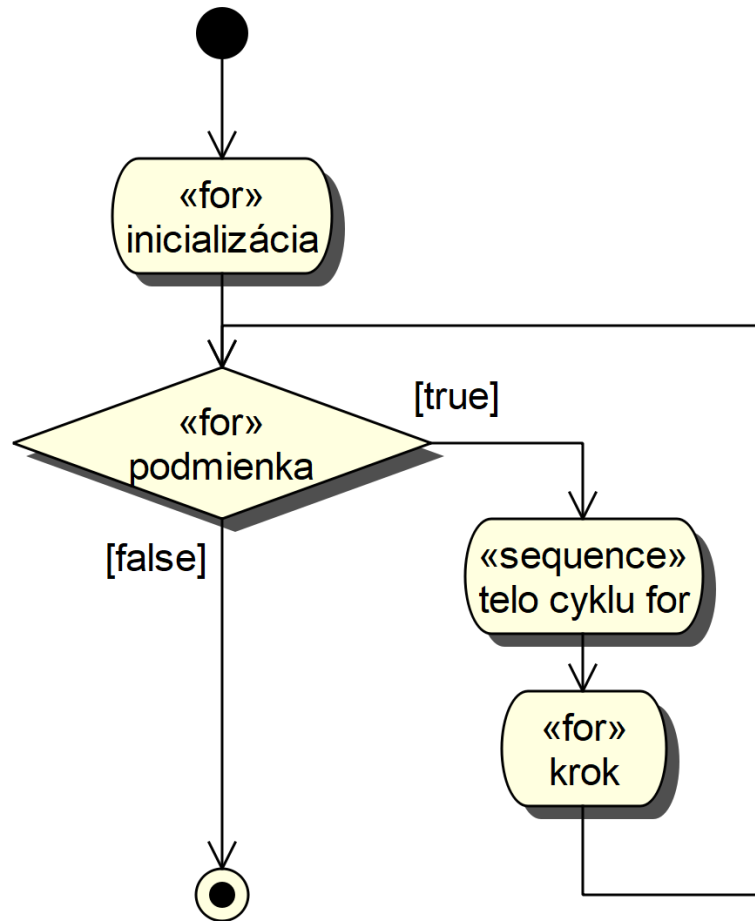
```
for (int i = 0; i < 24; i++) {  
    // telo cyklu  
}
```

- je ekvivalent

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



# Cyklus for v diagrame aktivít



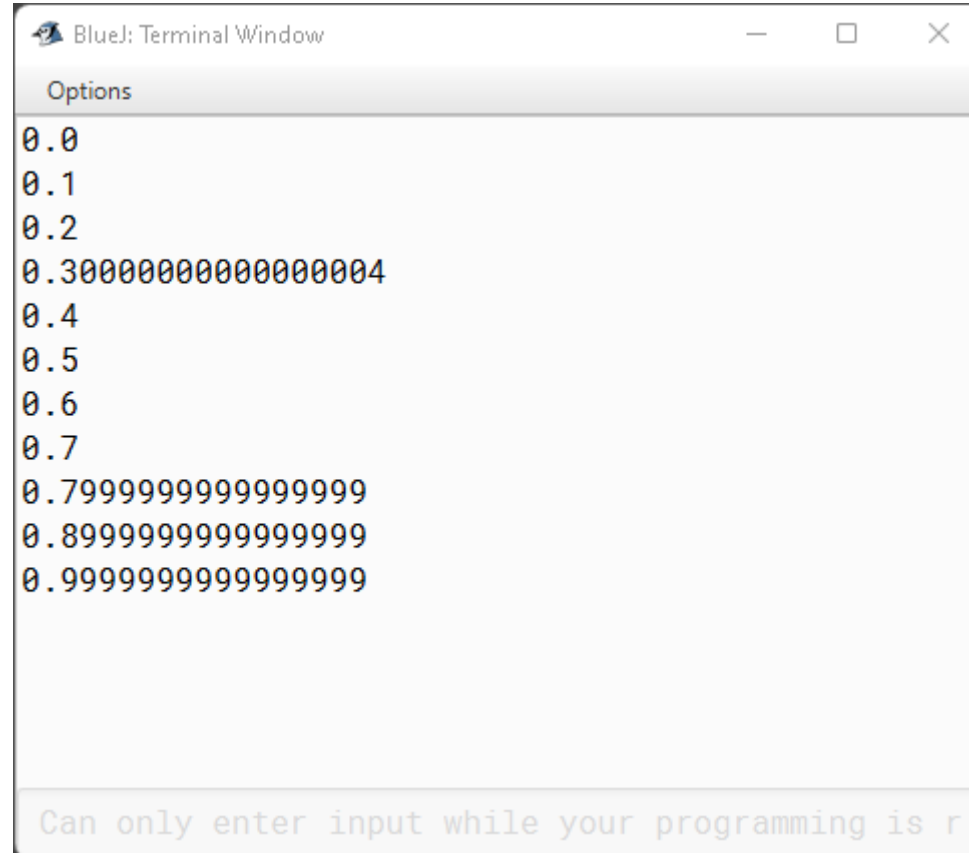
# Cyklus for a double (1)

- 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

Options

```
0.0  
0.1  
0.2  
0.30000000000000004  
0.4  
0.5  
0.6  
0.7  
0.7999999999999999  
0.8999999999999999  
0.9999999999999999
```

Can only enter input while your programming is r

# Príklad

- vytlačenie viac lískov naraz
- používateľ zadá počet lískov o ktoré má záujem
- pevný počet opakovaní – cyklus for

# Metóda tlacListky

```
public void tlacListky(int pocet) {  
    for (int i = 0; i < pocet; i++) {  
        // tlac listka - vynechane prikazy  
        this.trzba = this.trzba + this.cenaListka;  
        this.vlozenaCiastka = this.vlozenaCiastka - this.cenaListka;  
    }  
}
```

# Cyklus while

- pravidlo:
  - vykonávajú telo cyklu kým platí podmienka

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

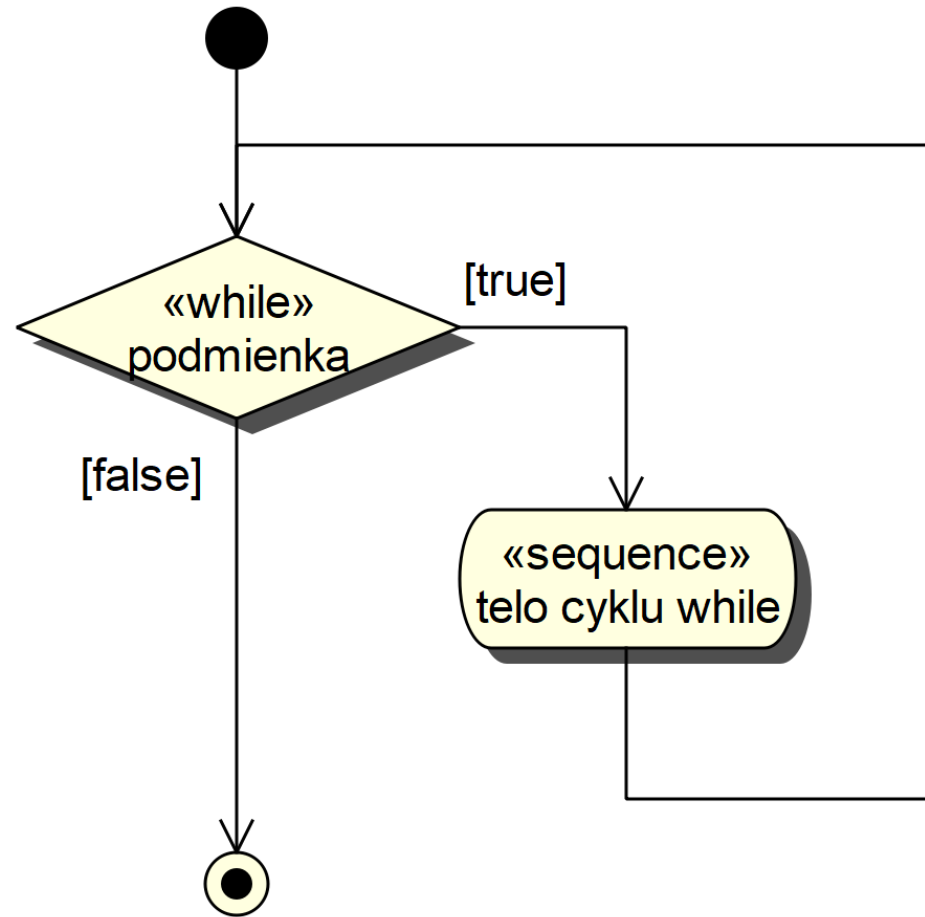
# Cyklus for – ekvivalent cyklu while

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

- je ekvivalent

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

# Cyklus while v diagrame aktivít





# Príklad

- vytlačenie lístkov podľa vloženej čiastky
- nepoznáme počet opakovaní, poznáme podmienku – cyklus while
  - podmienka cyklu: vložená čiastka je väčšia ako cena lístka

# Metóda tlacListky

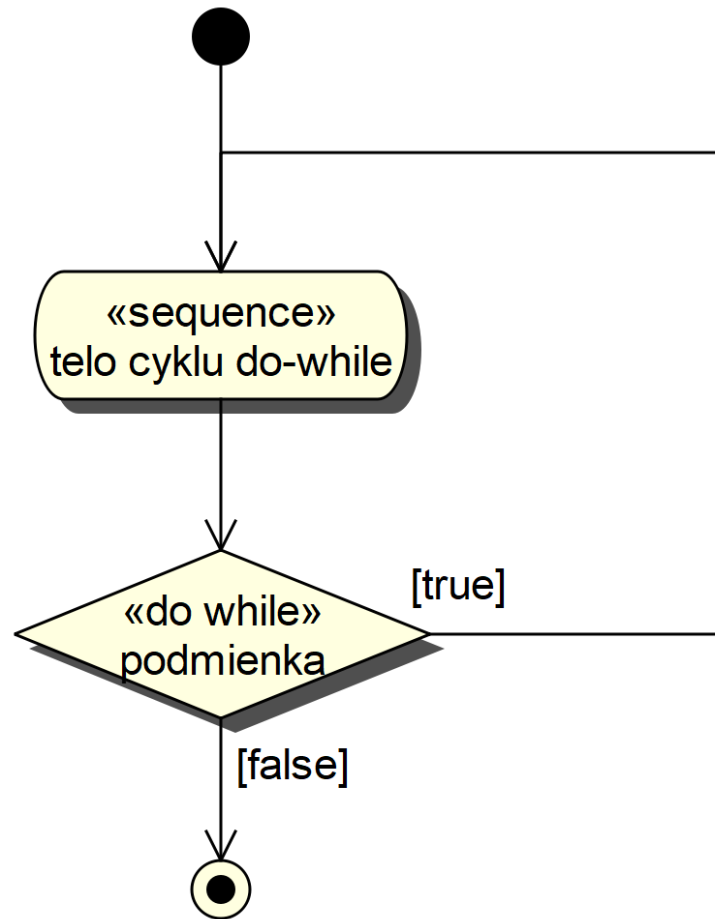
```
public void tlacListky() {  
    while (this.vlozenaCiastka > this.cenaListka) {  
        // tlac listka - vynechane prikazy  
        this.trzba = this.trzba + this.cenaListka;  
        this.vlozenaCiastka = this.vlozenaCiastka - this.cenaListka;  
    }  
}
```

# Cyklus do-while

- pravidlo:
  - vykonávajú kým platí podmienka
  - telo sa vykoná aspoň jeden krát
  - Pascalisti: nemýliť s repeat-until

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

# Cyklus do-while v diagrame aktivít



# while vs. do-while



# Príkazy break a continue v kontexte cyklu

- príkaz break – bezpodmienečné ukončenie cyklu
- príkaz continue – prechod na ďalšie opakovanie cyklu (podľa pravidla)
- oba porušujú princípy štruktúrovaného programovania
  - nevykoná sa príkaz bloku celý

# Príkaz break – kvíz

```
for (int i = 0; i < 10; i++) {  
    if (i == 5) {  
        break;  
    }  
    // pokračuje príkazy  
}
```

1. Koľko krát sa zopakuje cyklus?
2. Koľko krát sa vykonajú príkazy skryté za komentár?

## Príkaz continue – kvíz

```
for (int i = 0; i < 10; i++) {  
    if (i == 5) {  
        continue;  
    }  
    // pokračuje príkazy  
}
```

1. Koľko krát sa zopakuje cyklus?
2. Koľko krát sa vykonajú príkazy skryté za komentár?