

4

Skladanie objektov

Pojmy zavedené v 3. prednáške₍₁₎

- algoritmus
 - vlastnosti
 - procesor

Pojmy zavedené v 3. prednáške₍₂₎

- štruktúrované programovanie
 - základné konštrukčné prvky
 - postupnosť – sekvencia
 - vetvenie – alternatíva
 - grafické znázornenie v UML
- vetvenie v jazyku Java – príkaz if
 - blok
- vetvenie v jazyku Java – príkaz switch
- lokálna premenná

Pojmy zavedené v 3. prednáške₍₃₎

- Výrazy
 - aritmetický
 - aritmetické operátory – unárne, binárne
 - logický
 - relačné operátory
 - priorita operátorov
 - zátvorky vo výrazoch
 - typová kompatibilita

Pojmy zavedené v 3. prednáške₍₄₎

- identifikátory
 - pravidlá jazyka Java
 - konvencie Java

Cieľ prednášky

- skladanie objektov
- komunikácia medzi objektmi pomocou správ
- objektové typy
- trieda String

- príklad: digitálne hodiny

Modularizácia a abstrakcia

- jednoduchá úloha – jeden objekt
- zložitá úloha – rozklad na menšie podúlohy, časti – viac ako jeden objekt
 - rozdeľuj a panuj
 - divide and conquer
 - divide et impera
- modularizácia – rozklad na menšie časti – moduly
- abstrakcia – zanedbanie vnútorných detailov jednotlivých častí

Príklad – auto

- časti auta – motor, prevodovka, kolesá,...
 - motor – zdroj sily pohybu auta
 - prevodovka – zmena veľkosti a smeru sily prenášanej na kolesá
 - kolesá – pohyb auta po ceste, zatáčanie
 - ...

Kompozícia – skladanie objektov₍₁₎

- každá časť – špecifická úloha
- auto – spolupráca častí
- vonkajší pohľad – auto ako celok
- okolie auta – len auto ako celok

Kompozícia – skladanie objektov₍₂₎

- auto – objekt
- motor, prevodovka, kolesá – objekty
- auto – objekt – celok
- motor, prevodovka, kolesa – objekty – časti

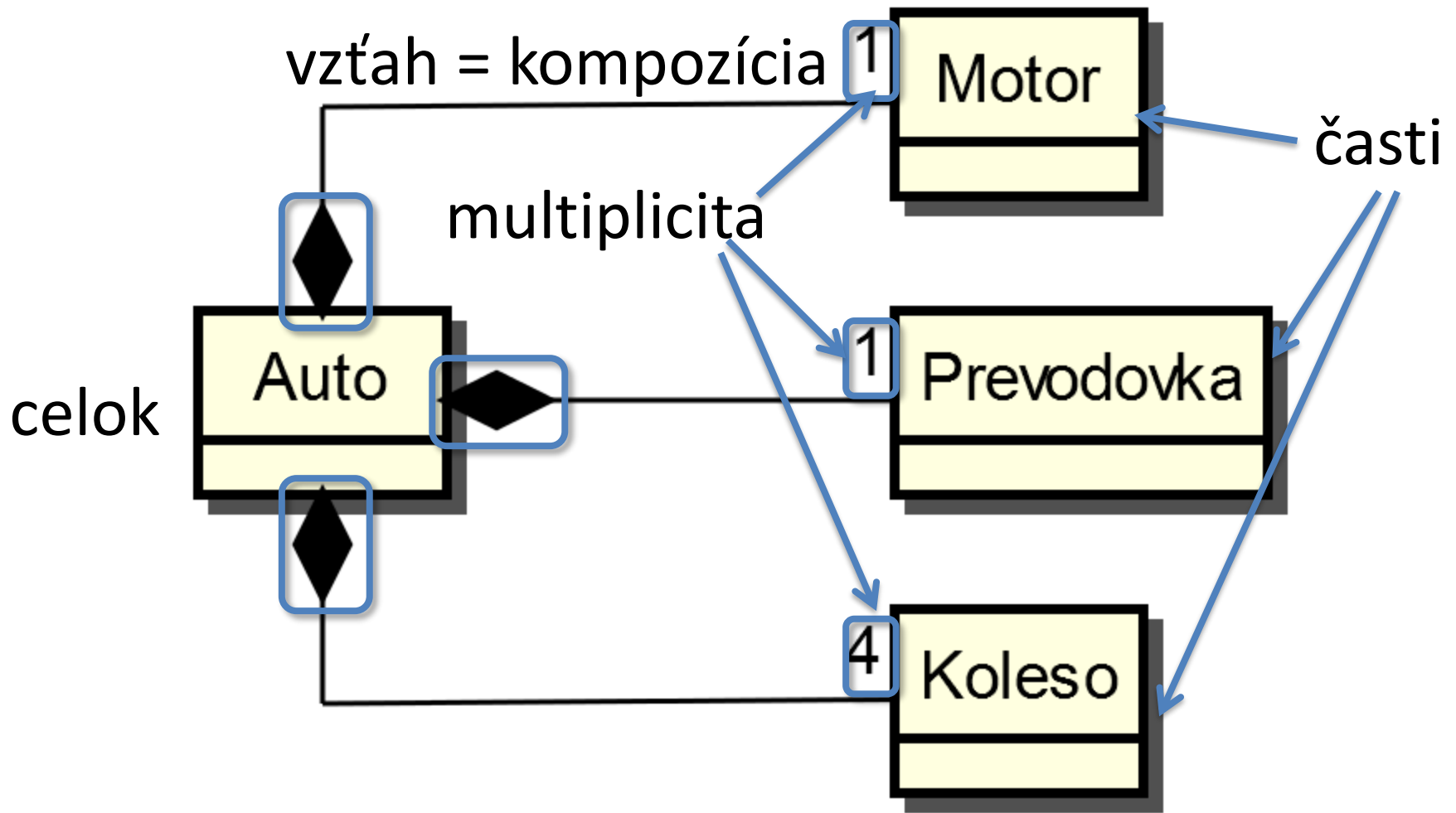
Kompozícia – skladanie objektov₍₃₎

- vytváranie zložených objektov – skladanie
- skladanie objektov – kompozícia
- kompozícia – závislosť celku a jeho častí
 - celok – nadriadený objekt
 - časť – podriadený objekt
- úloha celku – organizovanie spolupráce častí

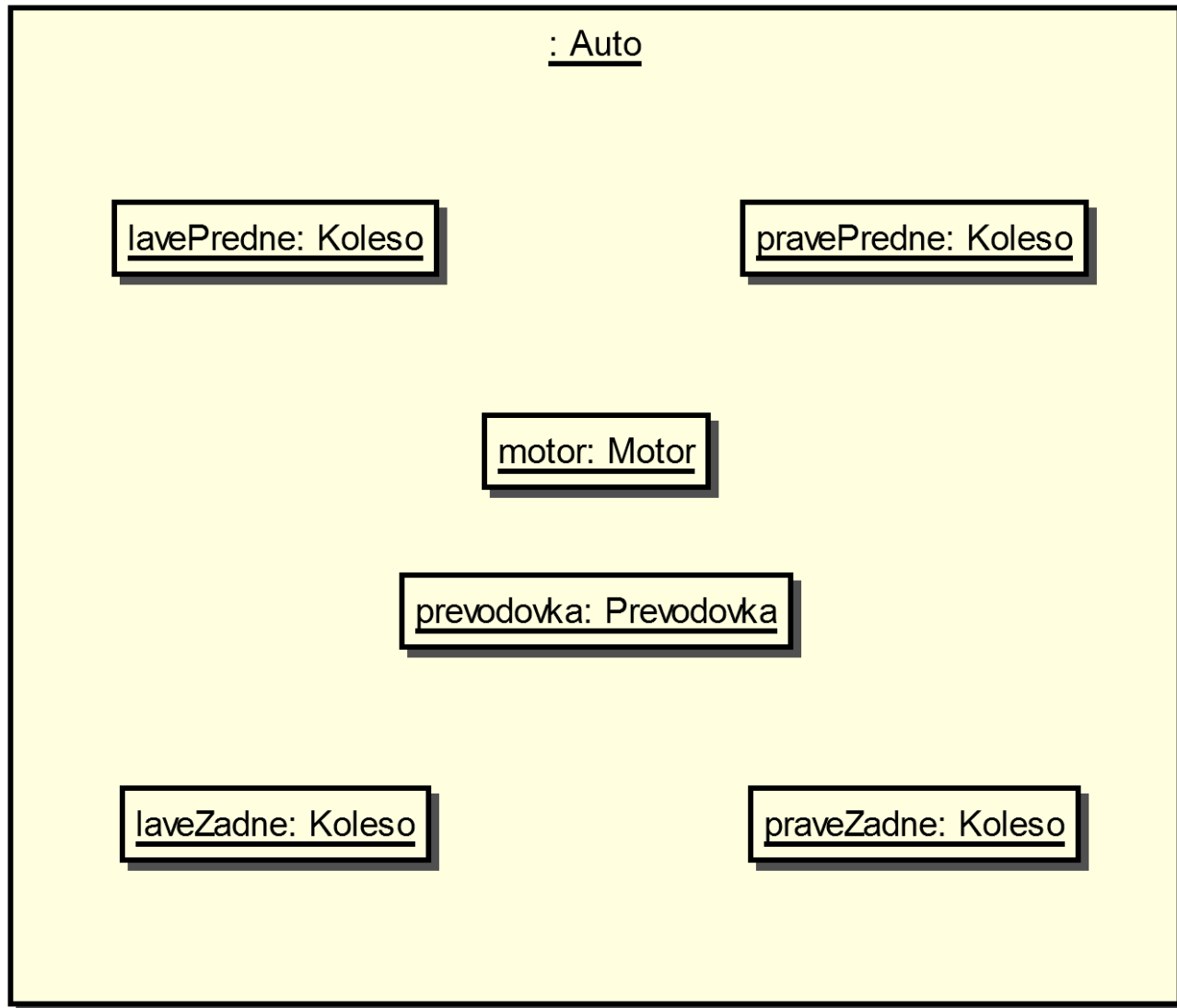
Kompozícia – skladanie objektov₍₄₎

- charakteristika kompozície
- spoločný životný cyklus
 - spoločný vznik – celok (+ časti)
 - vznik časti – súčasť vzniku celku
 - služby – len celok
 - vonkajší pohľad – rozhranie auta
 - spoločný zánik – celok (+ časti)
- zodpovednosť celku za časti

Kompozícia v UML



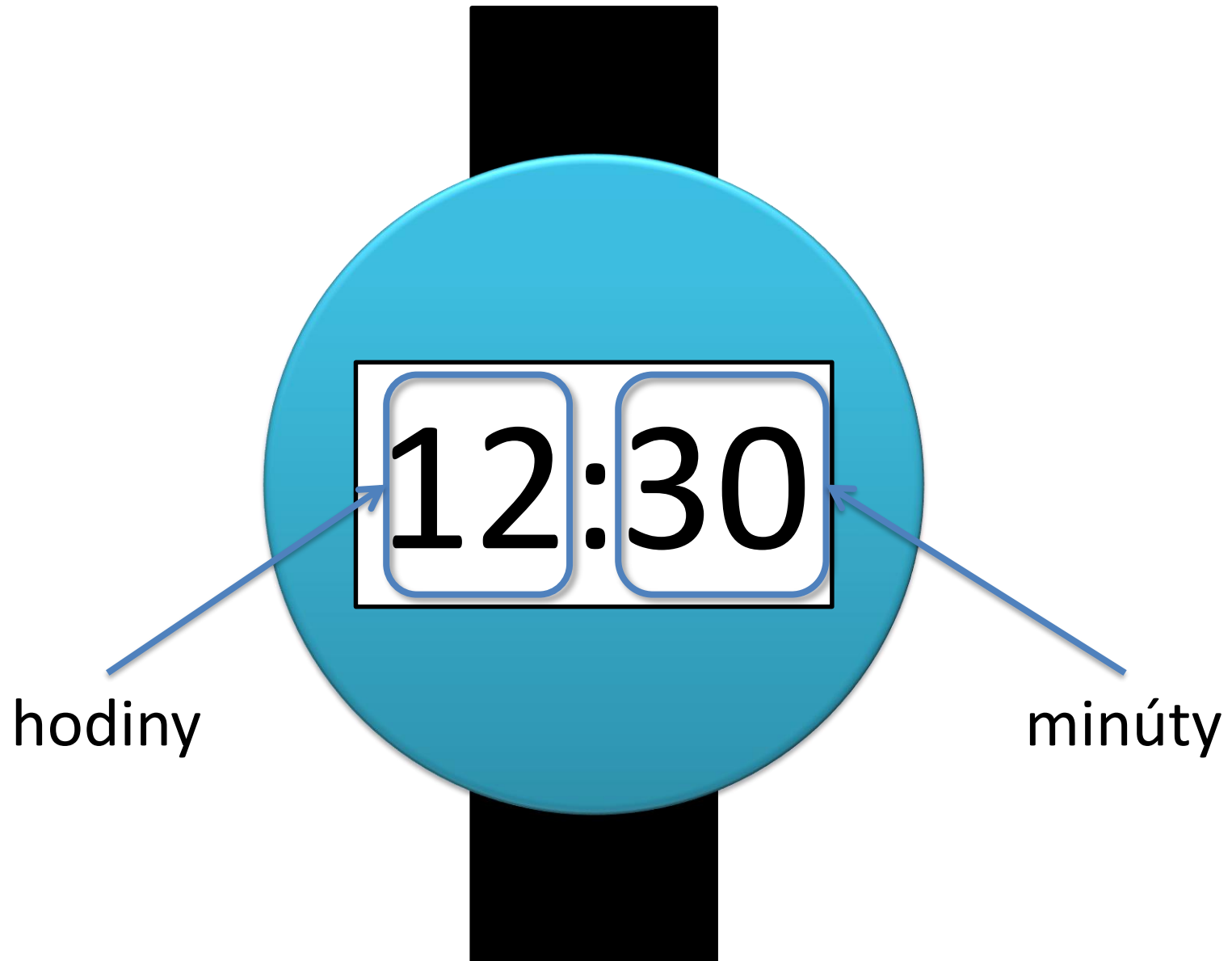
Kompozícia – diagram objektov



Projekt digitálne hodiny

- Požadované služby:
 - zobrazujú aktuálny čas 24-hod. formát
 - 00:00 – polnoc
 - 23:59 – minúta pred polnocou
- dajú sa nastaviť na požadovaný čas
- „tikajú“ – plynutie času – časový krok
 - krok 1 minúta
- trieda vytvorí inštanciu hodín
 - začiatočný čas: 00:00

Príklad: digitálne hodiny



Návrhy riešenia

- jediný objekt – podobne ako automat MHD
- kompozícia
 - digitálne hodiny – celok
 - v rozhraní bude mať požadované služby
 - minúty – časť pre prácu s minútami
 - hodiny – časť pre prácu s hodinami

Charakteristika minút

- „plynú“ – posunú sa o 1 minútu
 - najnižšia hodnota 00
 - najvyššia hodnota 59
- po uplynutí celej hodiny začínajú znova od 00
- vždy dvojciferné číslo – vedúca nula
- dajú sa nastaviť na požadovanú hodnotu z <00, 59>

Charakteristika hodín

- „plynú“ – posunú sa o 1 hodinu
 - najnižšia hodnota 00
 - najvyššia hodnota 23
- po uplynutí celého dňa začínajú znovu od 00
- vždy dvojciferné číslo – vedúca nula
- dajú sa nastaviť na požadovanú hodnotu z <00, 23>

Minúty/hodiny – rovnaké vlastnosti

- plynú
- najnižšia hodnota 00
- po dosiahnutí maxima pokračujú od 00
- formátovanie v tvare dvojciferného čísla
- dajú sa nastaviť na požadovanú hodnotu

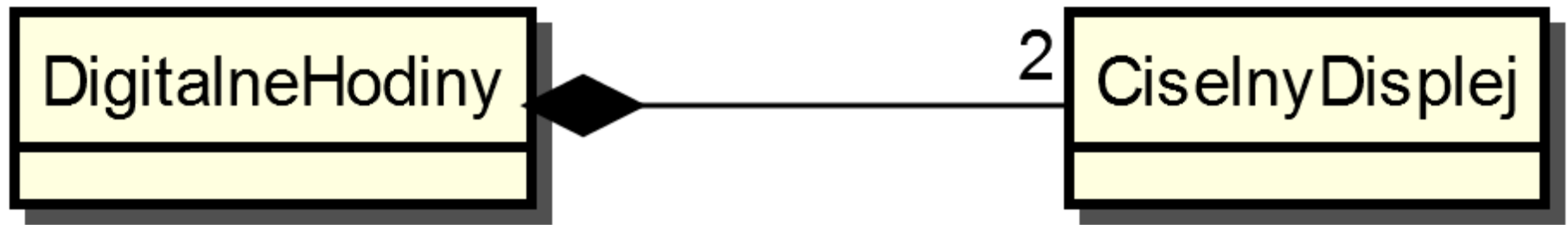
Minúty/hodiny – rozdiely

- krok pre hodiny: 1 hodina
- krok pre minúty: 1 minúta
- maximum pre hodiny: 23
- maximum pre minúty: 59

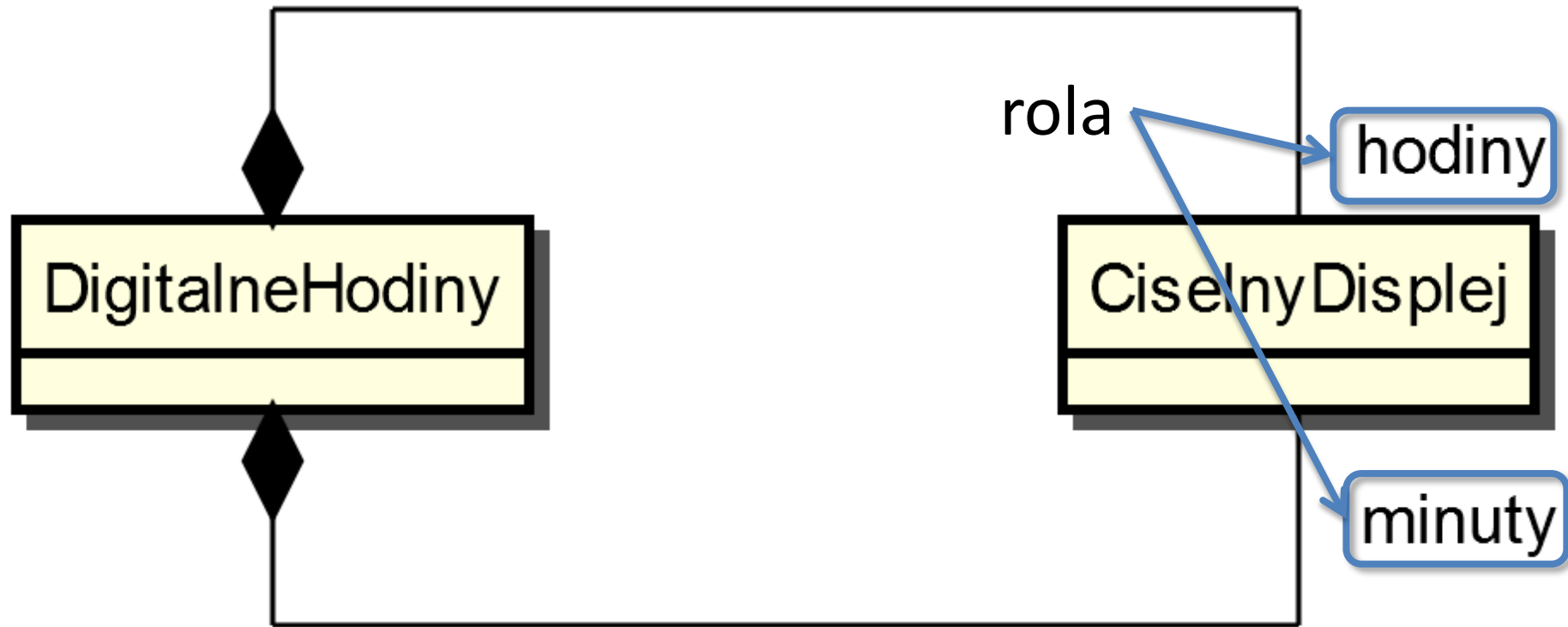
Riešenie rozdielov

- dve triedy (MinutovyDisplej, HodinovyDisplej)
- spoločná trieda (CiselnyDisplej)
 - rôznosť krokov
 - oba kroky o 1
 - rôzne jednotky – úroveň interpretácie
 - rôznosť maxima
 - nastaviteľné maximum
 - parametre konštruktora

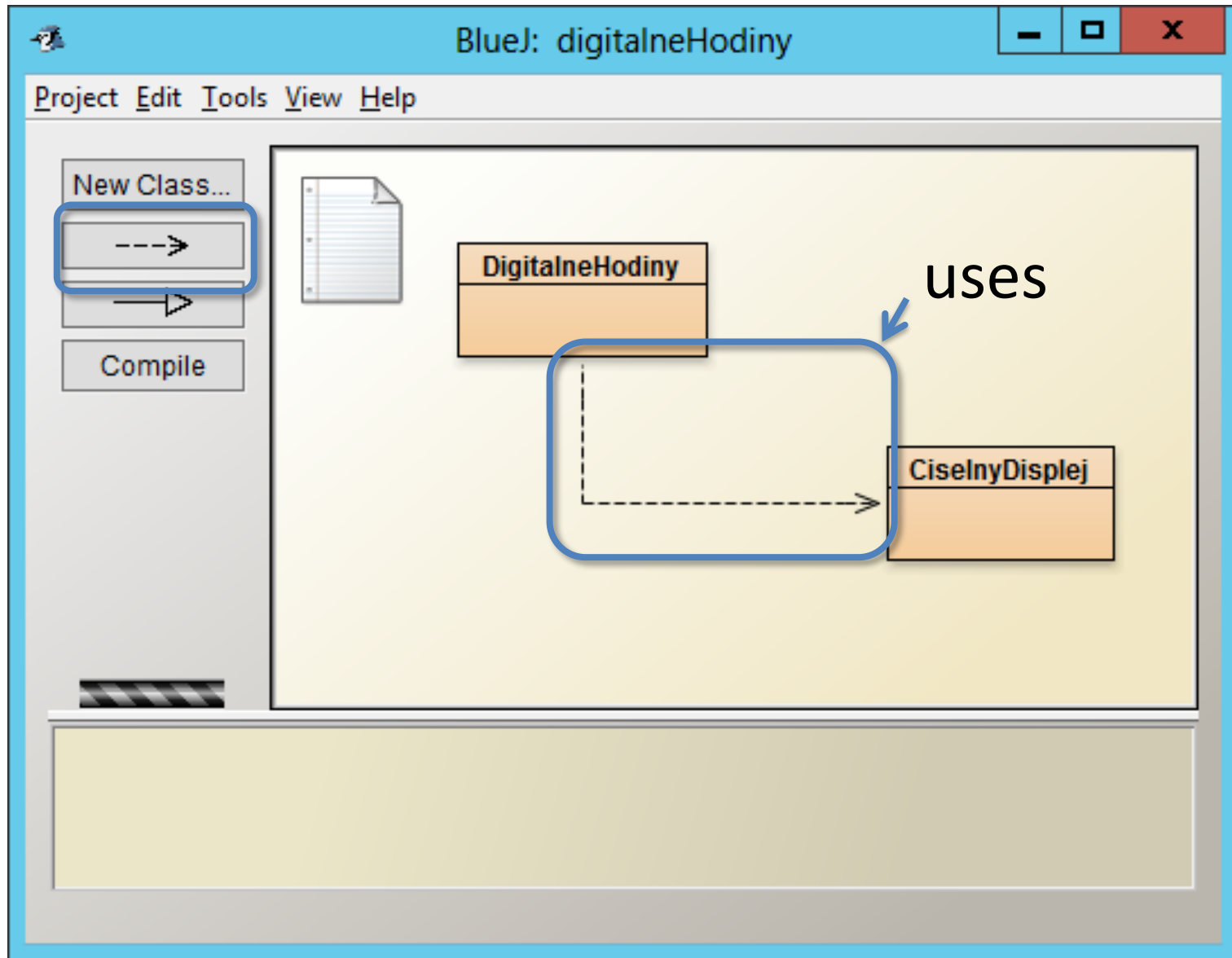
Digitálne hodiny v UML₍₁₎



Digitálne hodiny v UML₍₂₎



Digitálne hodiny v Bluej



Charakteristiky číselného displeja

- počítá kroky
- aktuálny stav vráti vo formáte dvojciferného čísla
- po dosiahnutí nastaveného maxima začína od 00
- nastaviteľný na požadovanú hodnotu z <00, max.>

Úlohy digitálnych hodín – celku

- vytvorenie displejov – častí
- vzťah hodín a minút – rôznosť jednotky
 - po uplynutí 60 minút krok pre hodiny
- nastavenie maxima

DigitalneHodiny

- + new(): DigitalneHodiny
- + tik(): void
- + setCas(hodiny: int, minuty: int): void
- + getCas(): String

DigitalneHodiny – vnútorný pohľad

DigitalneHodiny

- minuty: CiselnyDisplej
- hodiny: CiselnyDisplej

- + DigitalneHodiny()
- + tik(): void
- + setCas(hodiny: int, minuty: int): void
- + getCas(): String

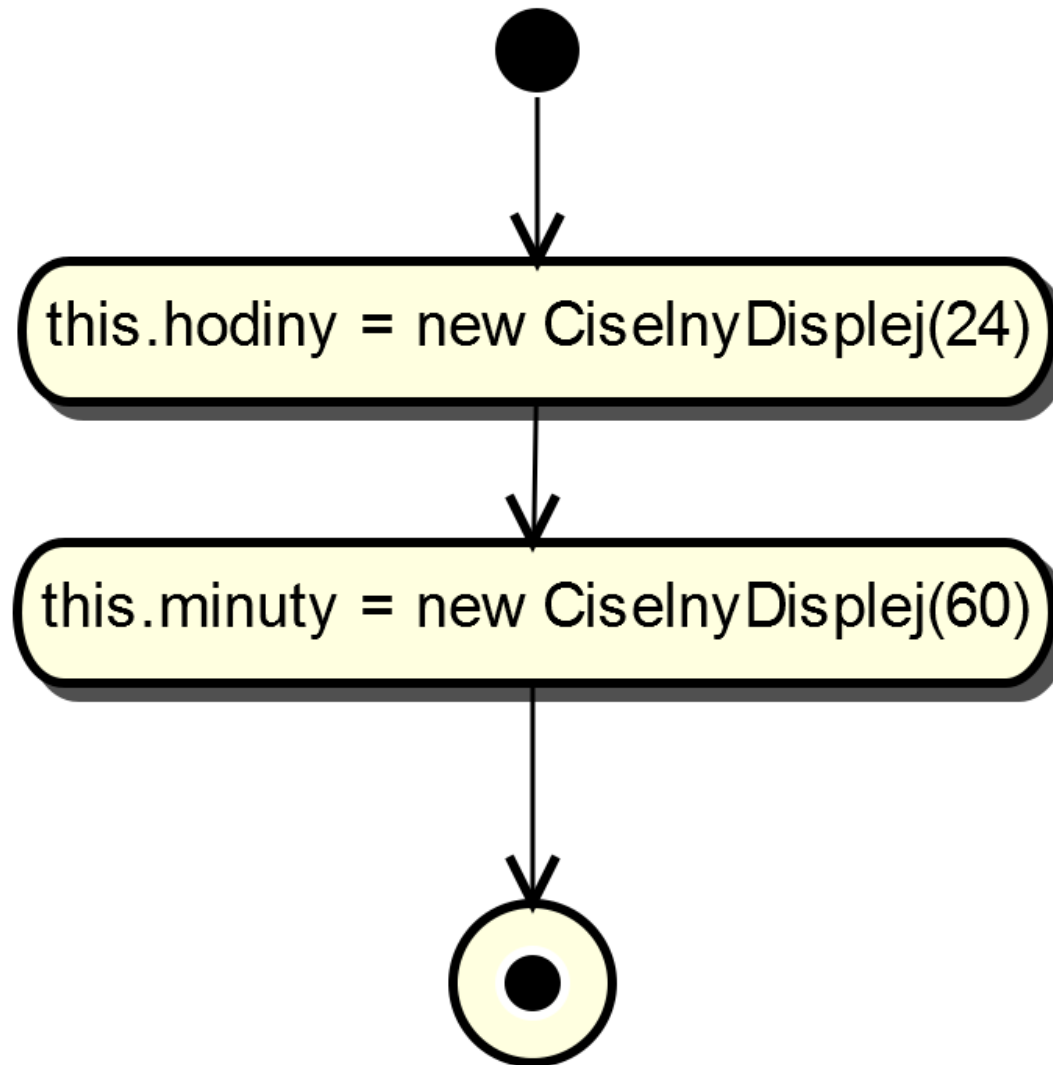
Objektové typy

- typ = názov triedy
 - trieda ako typ
- príklady:
 - typ premennej
 - hodiny : CislnyDisplej
 - typ návratovej hodnoty
 - getCas() : String

DigitalneHodiny – Java

```
public class DigitalneHodiny {  
    private CiselnýDisplej hodiny;  
    private CiselnýDisplej minuty;  
  
    ...  
}
```

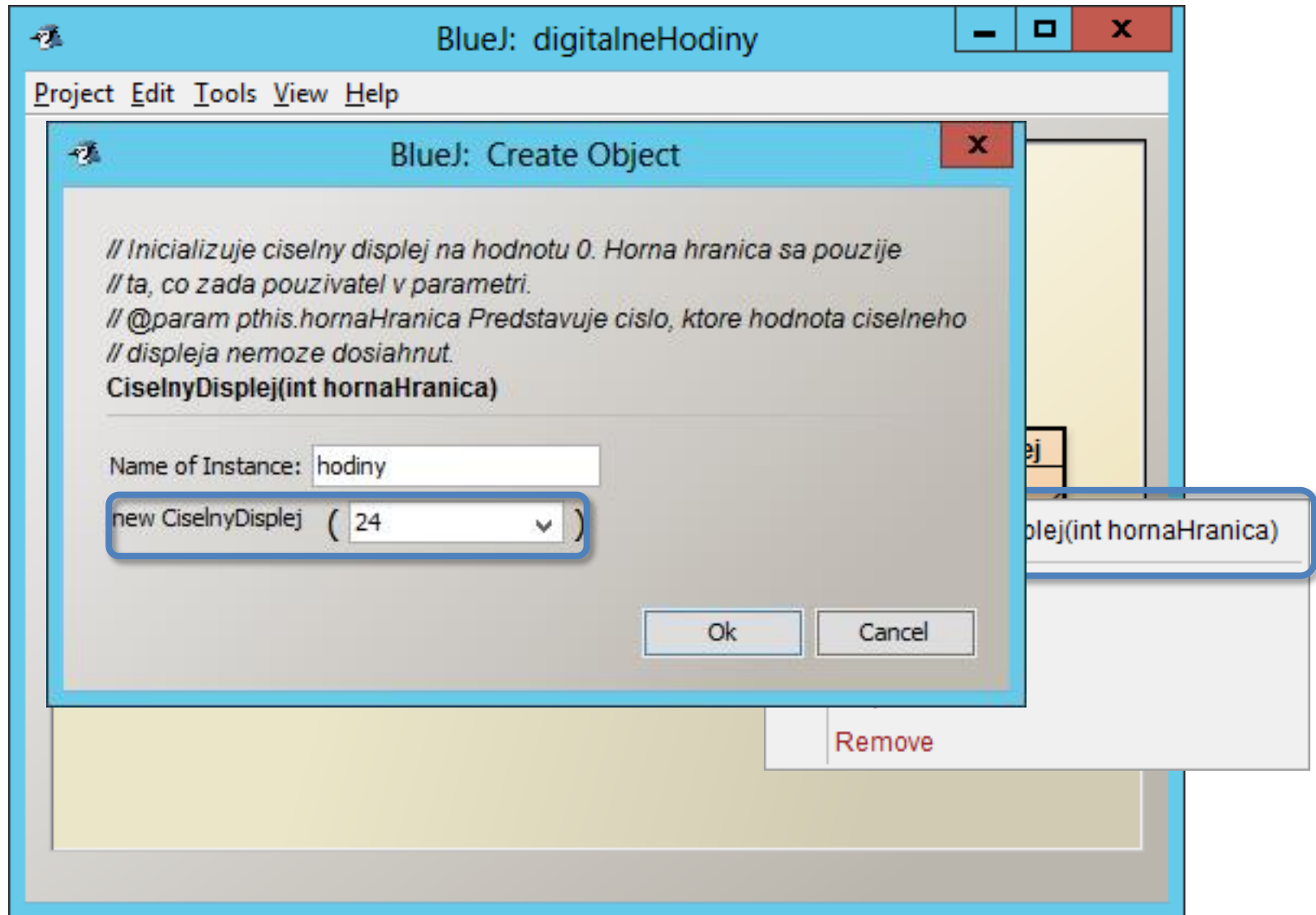
DigitalneHodiny – konštruktor – UML



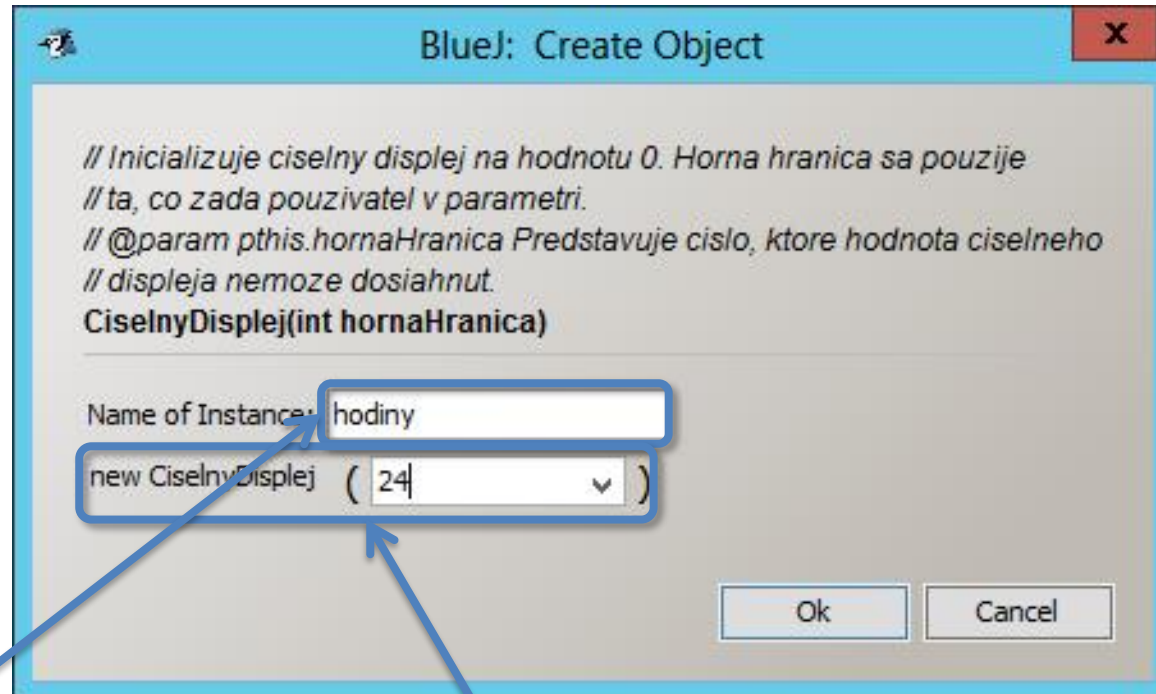
DigitalneHodiny – konštruktor – Java

```
public DigitalneHodiny() {  
    this.hodiny = new CiselyDisplej(24);  
    this.minuty = new CiselyDisplej(60);  
}
```

Správa triede „new“

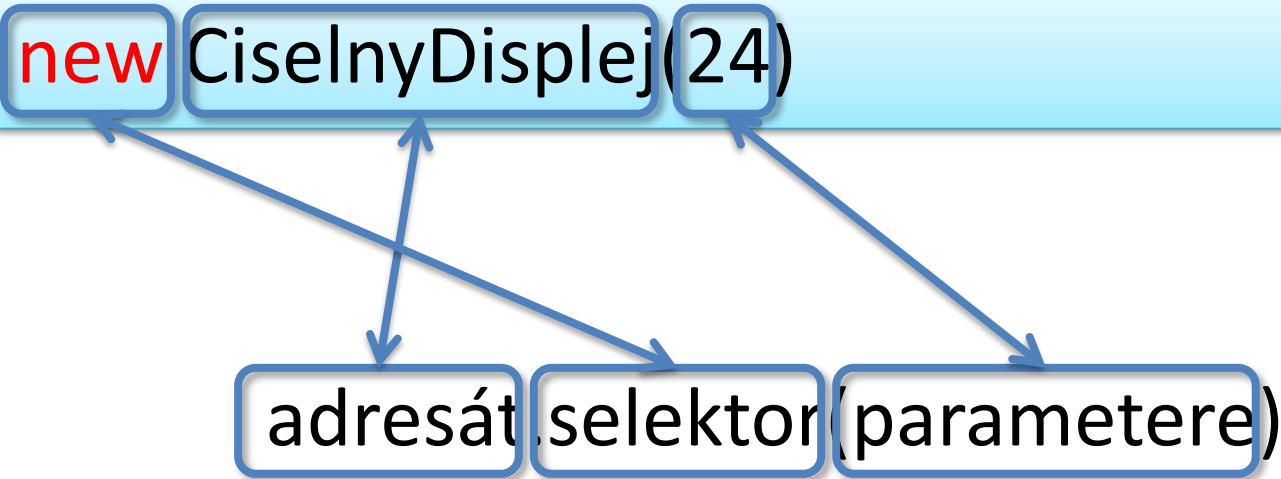


Správa triede „new“



this.hodiny = **new** CiselnyDisplej(24);

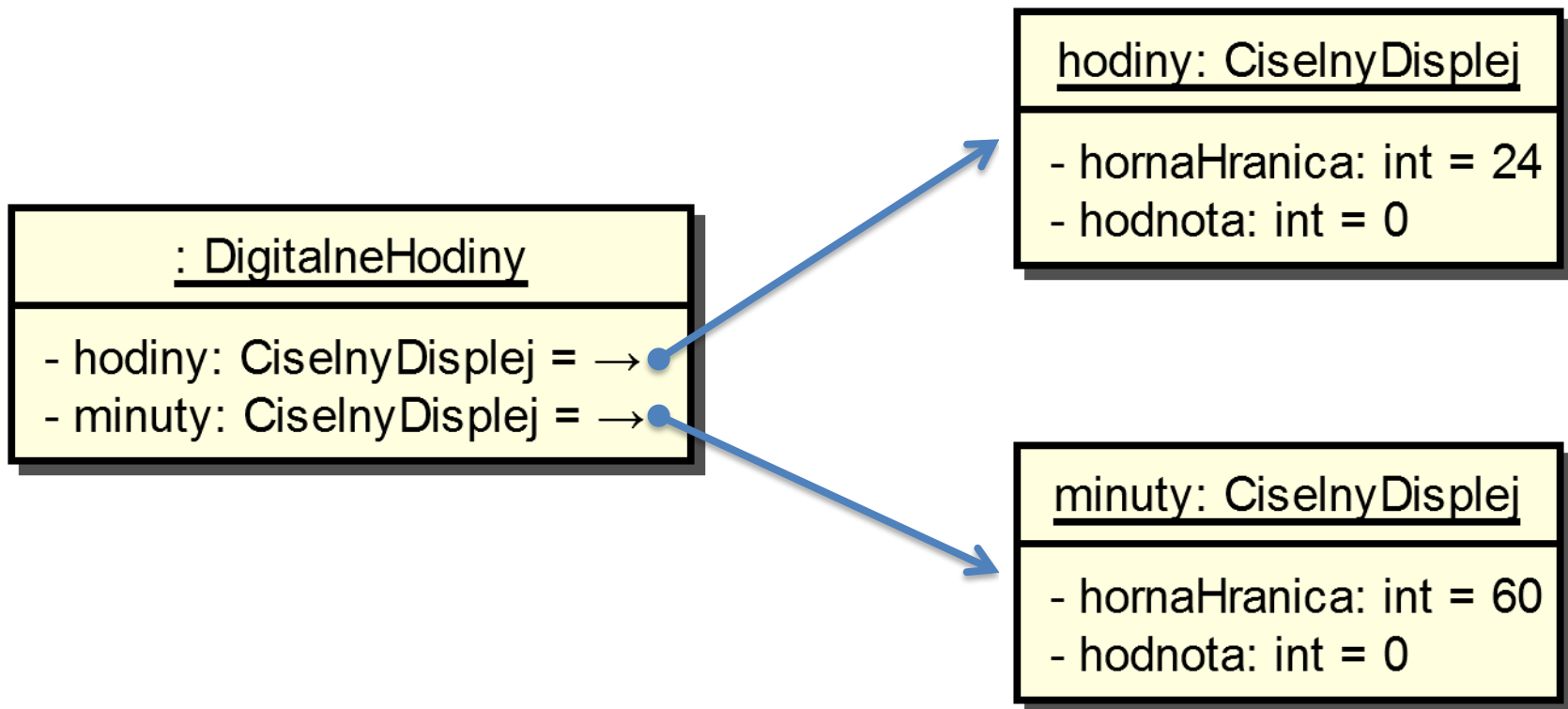
Štruktúra správy „new“



new = operátor v jazyku Java

DigitalneHodiny – konštruktor

```
this.hodiny = new CiselyDisplej(24);  
this.minuty = new CiselyDisplej(60);
```



Referencie

- premenná uchováva hodnotu
- hodnota objektového typu – [referencia](#)
- rozdiel v používaní primitívnych typov a objektových typov
 - primitívne typy – hodnota vo výrazoch
 - objektové typy
 - hodnota vo výrazoch
 - [adresát v správe](#)

DigitalneHodiny – diagram objektov₍₁₎

: DigitalneHodiny

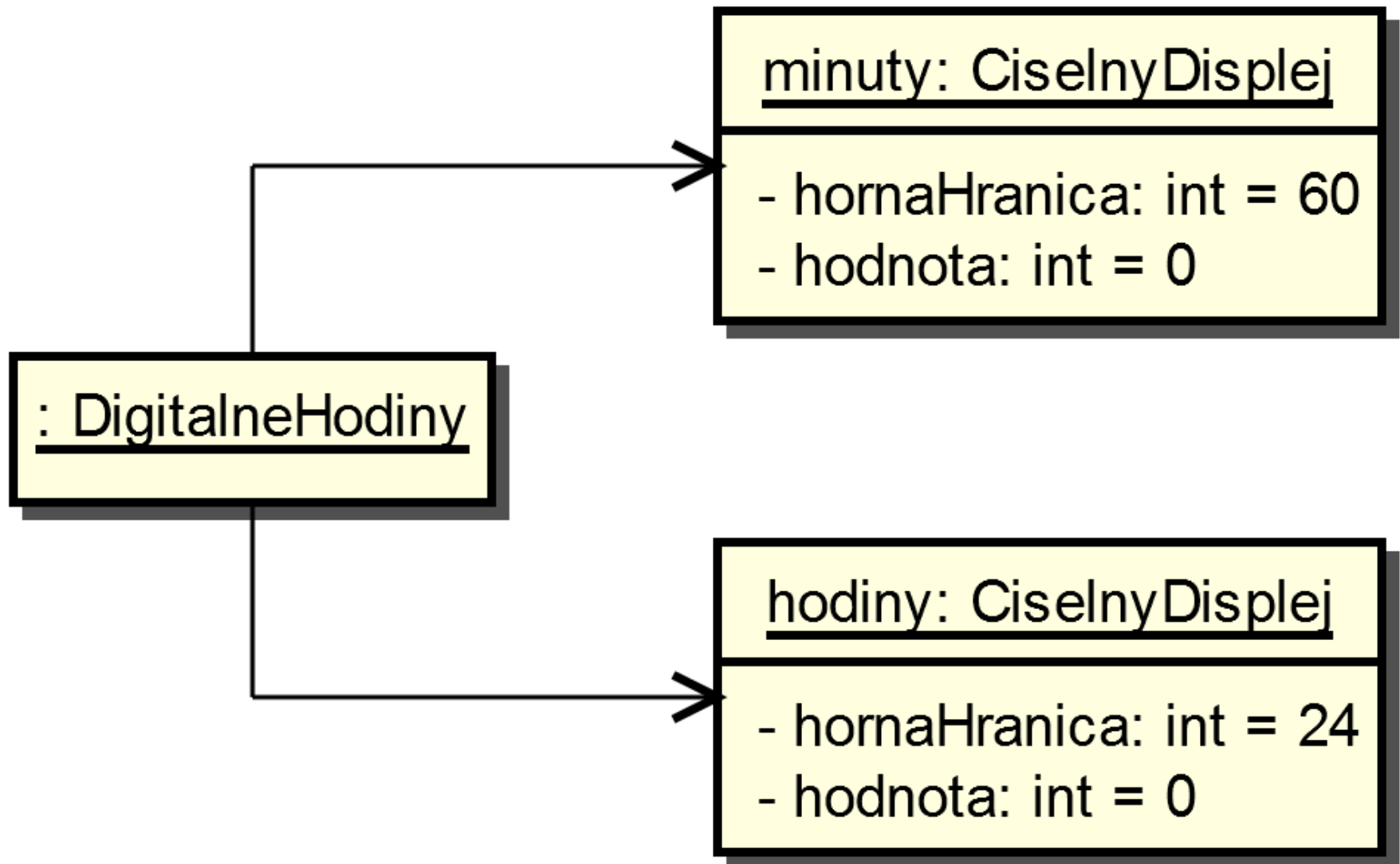
hodiny: CiselyDisplej

- hornaHranica: int = 24
- hodnota: int = 0

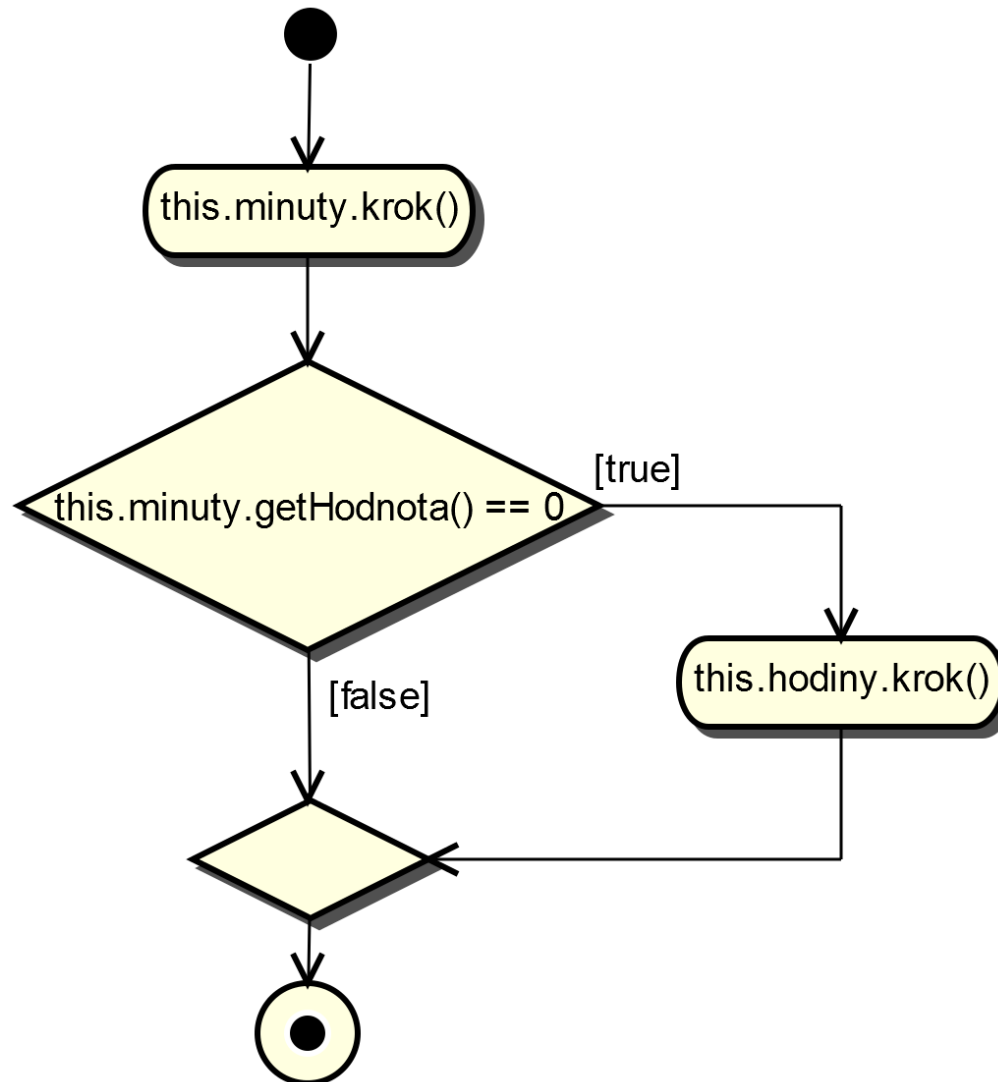
minuty: CiselyDisplej

- hornaHranica: int = 60
- hodnota: int = 0

DigitalneHodiny – diagram objektov₍₂₎



DigitalneHodiny – tik – UML



DigitalneHodiny – tik – UML

```
public void tik() {  
    this.minuty.krok();  
  
    if (this.minuty.getHodnota() == 0) {  
        this.hodiny.krok();  
    }  
}
```

Príkaz na poslanie správy

- objekt.sprava(skutočneParametre);
 - každý zo skutočných parametrov môže byť výraz
- vždy samostatný príkaz
- príklad:

```
this.minuty.krok();
```

adresát

selektor

zoznam skutočných
parametrov

Správy s návratovou hodnotou

- typ návratovej hodnoty
 - primitívny
 - objektový
- správa s návratovou hodnotou – výraz

aritmetický výraz

```
(this.minuty.getHodnota() == 0)
```

logický výraz

Objektový výraz

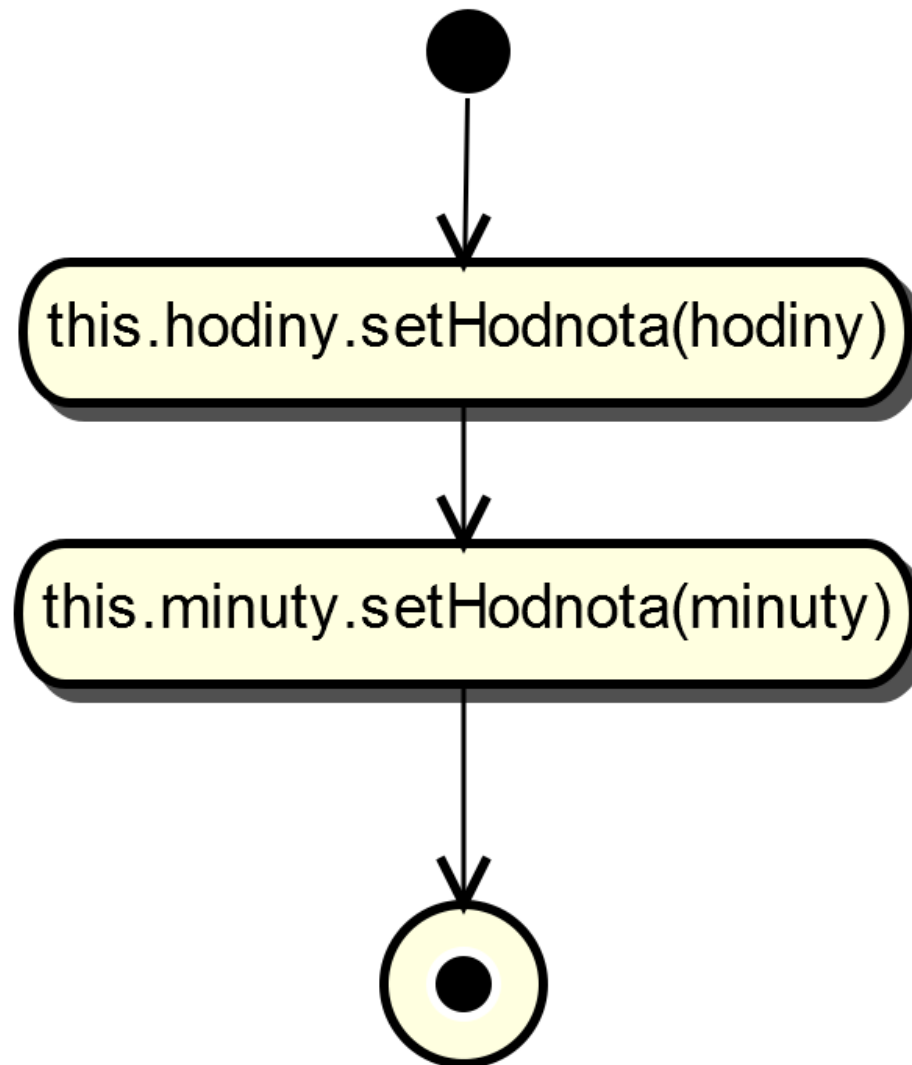
- výsledok vyhodnotenia objektového výrazu – referencia na objekt
- návratová hodnota správy „new“ – referencia na inštanciu danej triedy

```
this.hodiny = new CiselnýDisplej(24);
```

objektový výraz



DigitalneHodiny – setCas – UML




DigitalneHodiny – setCas – Java

```
public void setCas(int hodina, int minuta) {  
    this.hodiny.setHodnota(hodina);  
    this.minuty.setHodnota(minuta);  
}
```

Metóda getCas

```
public String getCas()
```



A blue arrow originates from the `String` type in the method signature above and points to a black-bordered box containing the text `13:15`.

13:15

Literál typu String

- reťazcový literál

"ľubovoľný text v Unicode uzavretý do dvojice úvodzoviek"

- prázdny reťazec

""

- literál – realizovaný ako inštancia triedy String

Premenná typu String

- atribút

```
private String priezvisko;
```

- formálny parameter

```
public void setPriezvisko(String priezvisko)
```

- lokálna premenná

```
String priezvisko;
```

Spájanie reťazcov₍₁₎

- Java používa operátor + pre spájanie reťazcov

- spojením dvoch reťazcov

"Žilinská" + " univerzita"

- vzniká nový reťazec (nová inštancia triedy String)

"Žilinská univerzita"

Spájanie reťazcov₍₂₎

- reťazcový výraz

prvyOperand + druhyOperand

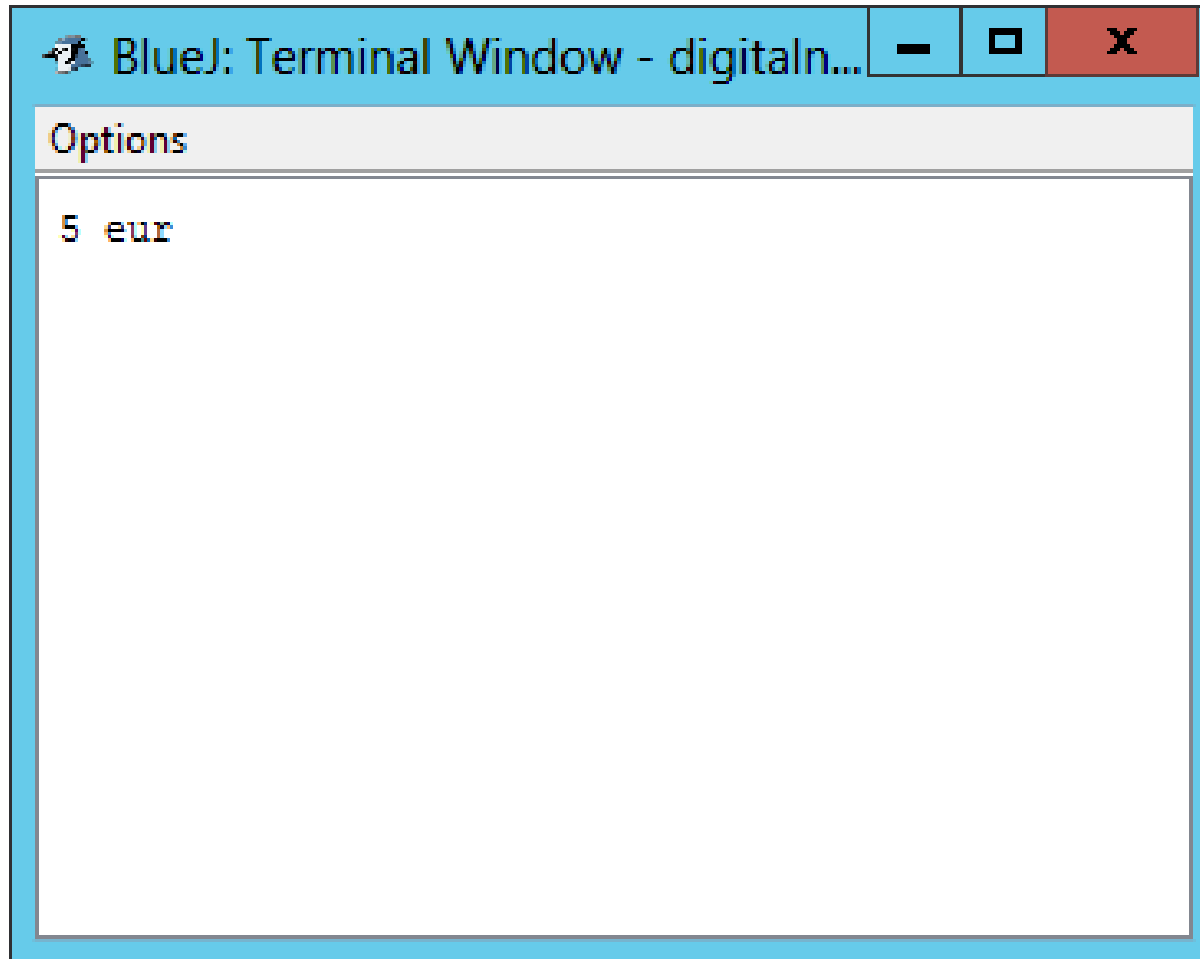
- aspoň jeden operand je reťazcový
- hodnota reťazcového výrazu – reťazec
- reťazcový operand
 - literál
 - premenná typu String
 - návratová hodnota typu String
 - reťazcový výraz

Spájanie reťazcov₍₃₎

- iný ako reťazcový operand sa automaticky prekonvertuje na reťazec
- vyhodnocovanie výrazu – zľava doprava

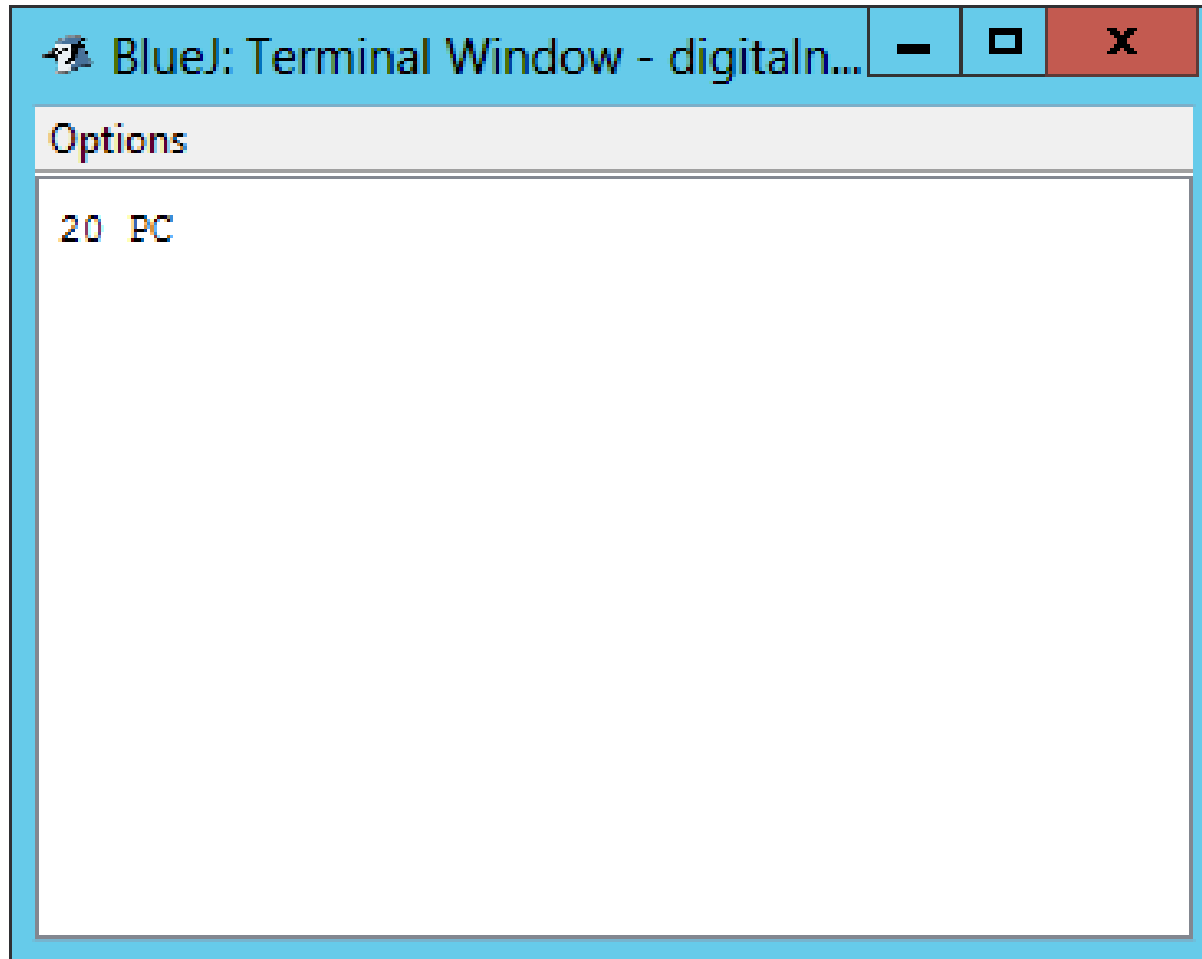
Spájanie reťazcov₍₁₎

```
System.out.println(5 + " eur");
```



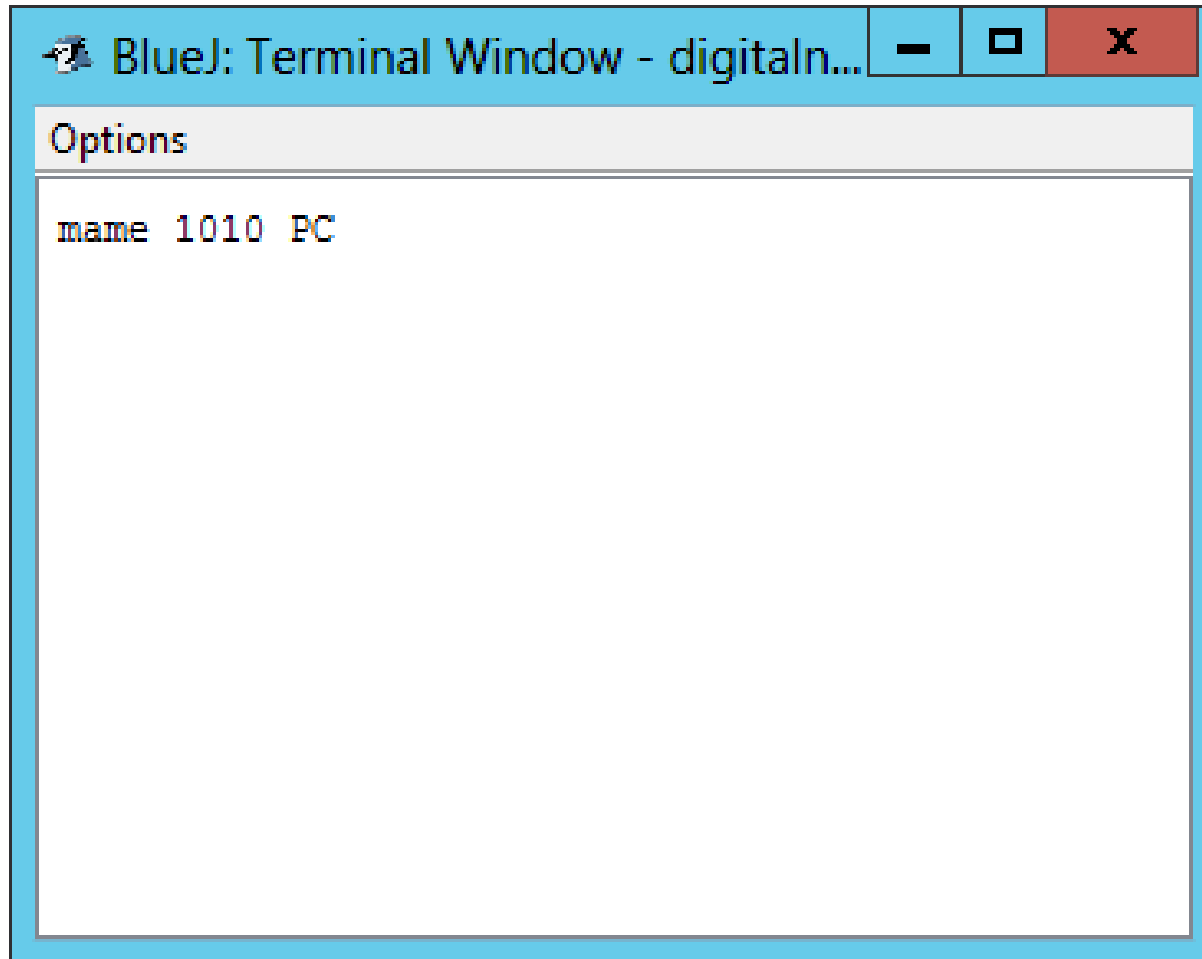
Spájanie reťazcov₍₂₎

```
System.out.println(10 + 10 + " PC");
```



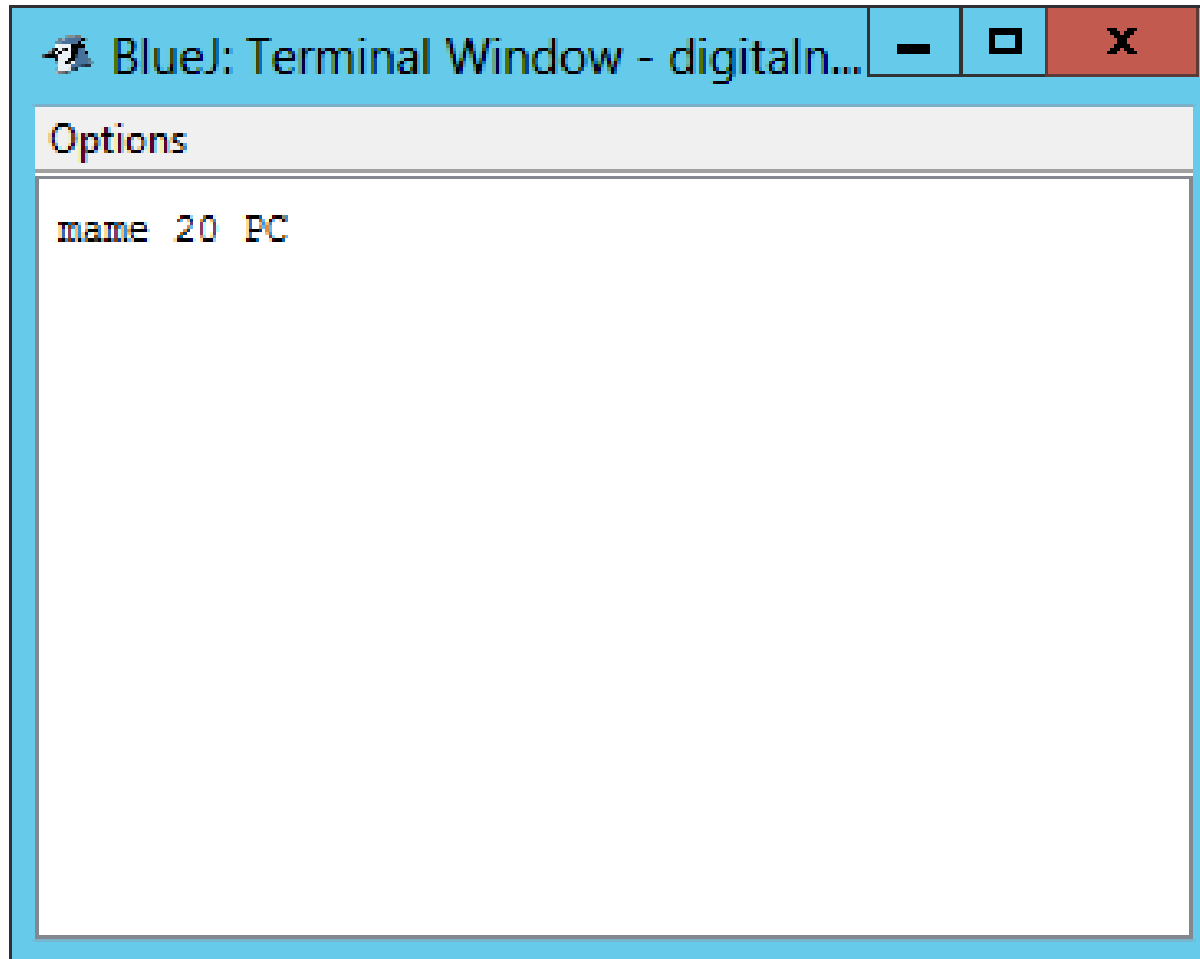
Spájanie reťazcov₍₃₎

```
System.out.println("mame "+ 10 + 10 + " PC");
```



Spájanie reťazcov₍₄₎

```
System.out.println("mame "+ (10 + 10) + " PC")
```



Metóda getCas

```
public String getCas() {  
    return this.hodiny.getHodnotaAkoRetazec() + ":"  
        + this.minuty.getHodnotaAkoRetazec();  
}
```

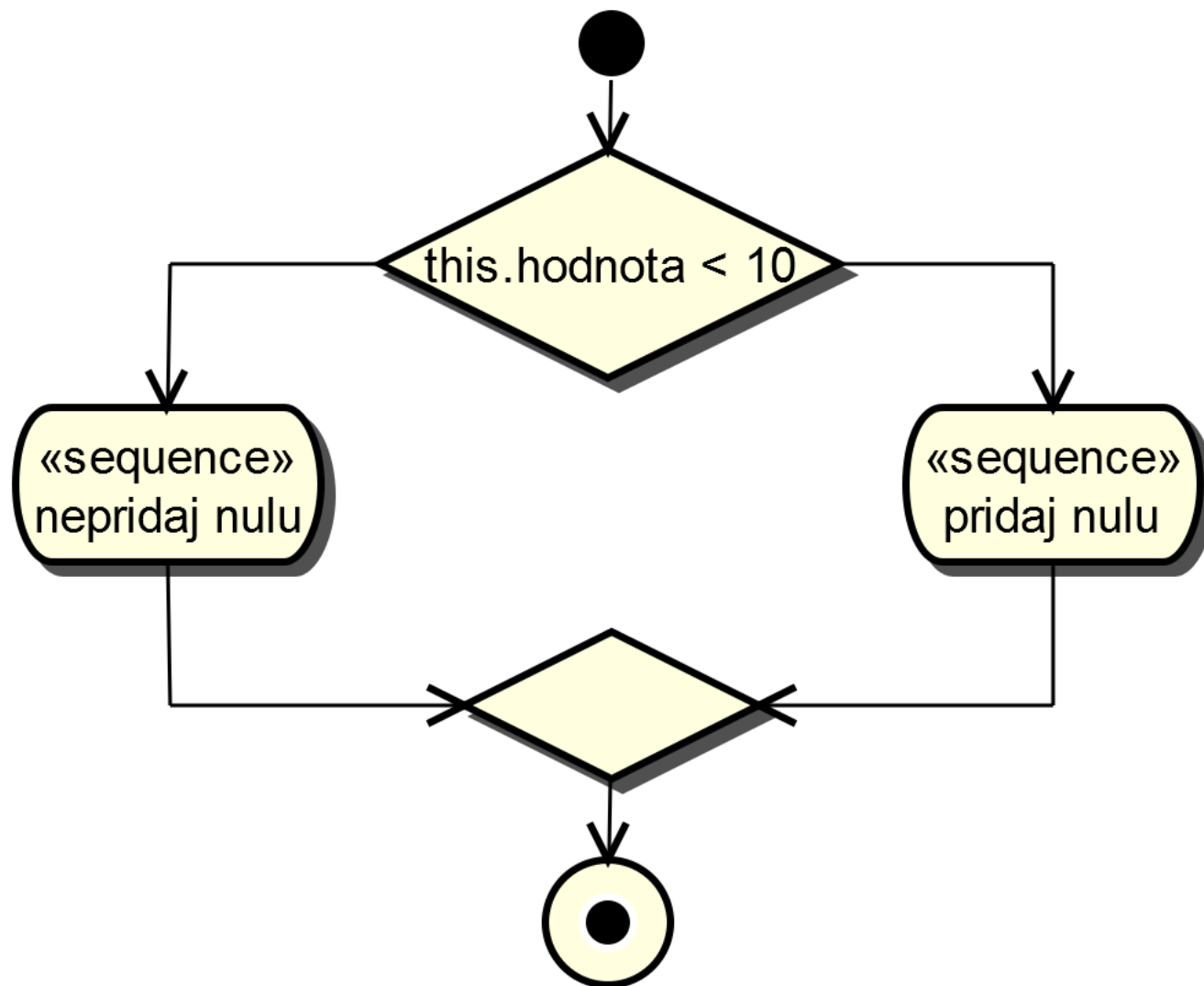
CiselnyDisplej – getHodnotaAkoRetazec

```
public String getHodnotaAkoRetazec ()
```

- vracia hodnotu ako dvojciferné číslo
- nula na začiatku

06

Riešenie problému s nulou – UML



Riešenie problému s nulou – Java₍₁₎

```
public String getHodnotaAkoRetazec() {  
    String hodnotaAkoRetazec;  
    if (this.hodnota < 10) {  
        hodnotaAkoRetazec = "0" + this.hodnota;  
    } else {  
        hodnotaAkoRetazec = "" + this.hodnota;  
    }  
    return hodnotaAkoRetazec;  
}
```

Riešenie problému s nulou – Java₍₂₎

```
public String getHodnotaAkoRetazec() {  
    if (this.hodnota < 10) {  
        return "0" + this.hodnota;  
    } else {  
        return "" + this.hodnota;  
    }  
}
```

Ďalšie spôsoby spájania reťazcov

- `StringBuilder`
- `String.format/System.out.format`

StringBuilder

- objekt na vytváranie dlhších reťazcov

StringBuilder
+ <u>new(): StringBuilder</u> + append(value): void + toString(): String

StringBuilder – použitie

```
int pocet = 10 + 10;  
StringBuilder retazec = new StringBuilder();  
retazec.append("Mame ");  
retazec.append(pocet);  
retazec.append(" PC");  
System.out.println(retazec.toString());
```

Dôvod pre použitie StringBuilder

- reťazcový operátor + sa aj tak prekladá na použitie StringBuilder
- rýchlejšie spájanie viac reťazcov
 - Shlemiel The Painter Algorithm

StringBuilder vs. operátor +

```
String a = "a";  
// StringBuilder sb = new StringBuilder();  
// sb.append(a); sb.append(a);  
// String b = sb.toString();  
String b = a + a;  
  
// StringBuilder sb = new StringBuilder();  
// sb.append(b); sb.append(a);  
// String c = sb.toString();  
String c = b + a;  
...
```

String.format

- formátovanie reťazca

```
String.format("formátovací reťazec", hodnoty)
```

Formátovací znak	Význam
%d	Celé číslo
%s	Reťazec
%f	Desatinné číslo
%n	Koniec riadku

Formát	Význam
%5 znak	Zarovnanie vpravo na 5 znakov
%-5 znak	Zarovnanie vľavo na 5 znakov
%5.2f	Zarovnanie vpravo na 5 znakov, 2 des. miesta

String.format – použitie

```
int pocet = 10 + 10;  
String retazec = String.format("Mame %d PC%n",  
                                pocet);  
System.out.println(retazec);  
  
// resp. iba pre vypis  
System.out.format("Mame %d PC%n", pocet);
```

Metóda krok

```
public void krok() {  
    this.hodnota = (this.hodnota + 1)  
                    % this.hornaHranica;  
}
```

Prirad'ovací príkaz

```
this.hodnota = (this.hodnota + 1) % this.hornaHranica;
```

0 + 1

1 % 24

1

```
this.hodnota = 1
```

```
int hodnota = this.hodnota + 1;
```

```
this.hodnota = hodnota % this.hornaHranica;
```

Metóda krok₍₁₎

hodiny: CiselyDisplej

- hornaHranica: int = 24
- hodnota: int = 0

`this.hodnota = (this.hodnota + 1) % this.hornaHranica;`

hodiny: CiselyDisplej

- hornaHranica: int = 24
- hodnota: int = 1

Metóda krok₍₂₎

hodiny: CiselyDisplej

- hornaHranica: int = 24
- hodnota: int = 1

`this.hodnota = (this.hodnota + 1) % this.hornaHranica;`

hodiny: CiselyDisplej

- hornaHranica: int = 24
- hodnota: int = 2

Metóda krok₍₃₎

hodiny: CiselyDisplej

- hornaHranica: int = 24
- hodnota: int = 23

```
this.hodnota = (this.hodnota + 1) % this.hornaHranica;
```

hodiny: CiselyDisplej

- hornaHranica: int = 24
- hodnota: int = 0

Vďaka za pozornosť