

Informatika 1

Testovanie a ladenie



Pojmy zavedené v 4. prednáške (1)

- modularizácia a abstrakcia
- kompozícia – skladanie objektov
 - spoločné životné cykly
 - UML

Pojmy zavedené v 4. prednáške (2)

- objektové typy
- referencie

Pojmy zavedené v 4. prednáške (3)

- príkaz na poslanie správy
 - príkaz na poslanie správy "new" triede
 - príkaz na poslanie správy bez návratovej hodnoty
 - príkaz na poslanie správy s návratovou hodnotou
- spracovanie správ a vykonávanie metód na zásobníku
- objektový výraz
 - výstupný parameter správy objektového typu
 - špeciálne: reťazcový výraz

Pojmy zavedené v 4. prednáške (4)

- reťazce
 - reťazcové literály
 - spájanie reťazcov
 - reťazcové výrazy
 - trieda String
 - trieda StringBuilder
 - správa String.format

Cieľ prednášky

- chyby v softvéri
- vyhľadávanie a odstraňovanie chýb
- overovanie správnej funkcie softvéru

- príklad: digitálne hodiny

Typy chýb

- syntaktické chyby
- behové chyby
- logické chyby

Syntaktické chyby

- zistí a hlási prekladač
- nedodržanie formálnych pravidiel programovacieho jazyka – syntax jazyka
- preklepy pri písaní zdrojového textu
- jasné chyby – na mieste kurzora
- nejasné chyby – nie na riadku s kurzorom
- !!! čítať texty chybových hlásení

Syntaktické chyby – príklad (1)

```
public CiselnýDisplej (int hornaHranica) {  
    this.hornaHranica = hornaHranica;  
    this.hodnota = 0  
}
```

';' expected

```
/**  
 * Vrať aktuálnu hodnotu číselného displeja vo forme celého čísla typu  
 * int.  
 */  
public int getHodnota () {  
    return this.hodnota;  
}
```

Syntaktické chyby – príklad (2)

```
public int getHodnota() {  
    return this.hodnota;  
}
```

```
/**
```

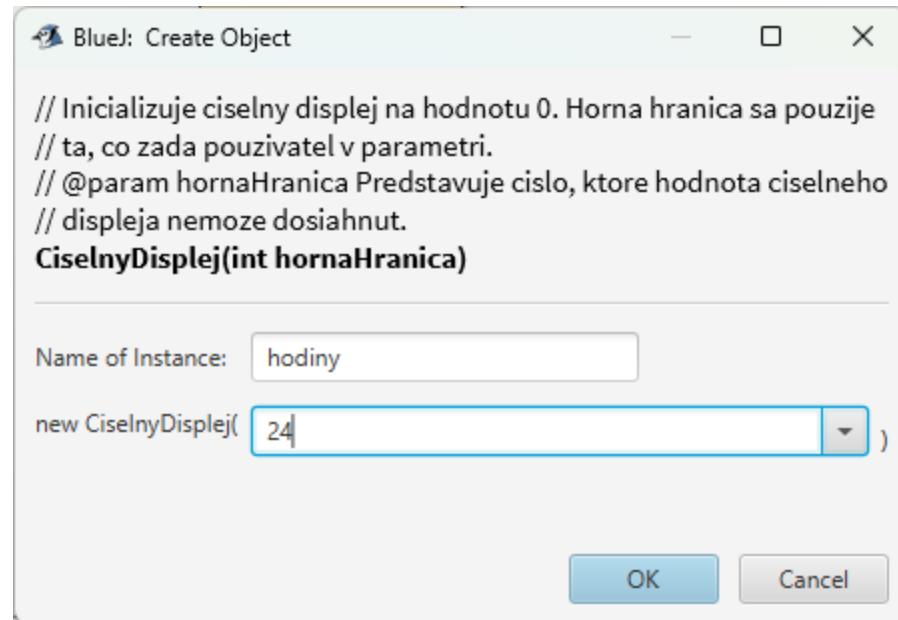
```
 * Zvacsi hodnotu na ciselnom displeji o hodnotu jedna. Ak dosiahne hornu  
 * hranicu, pokračuje znovu od nuly.  
 */
```

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

Behové chyby

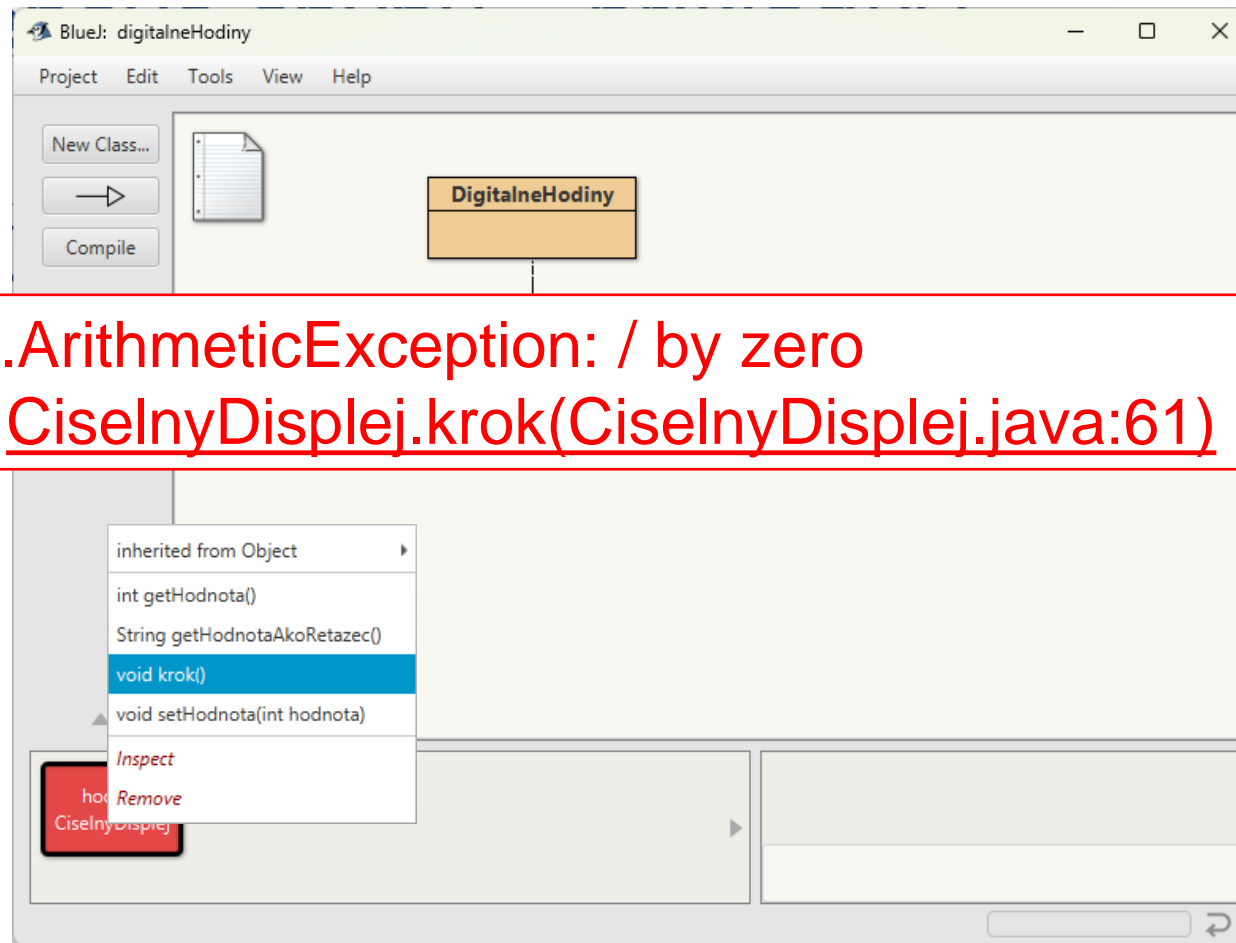
- zistí a „hlási“ procesor pri vykonávaní programu
- hlási = program „havaruje“
- procesor nemôže vykonať požadovaný príkaz
- delenie nulou, správa neexistujúcemu objektu, ...
- zákernosť behových chýb
 - nemusia sa prejaviť pri každom spustení programu
 - „zavlečená“ – skutočná chyba je niekde skôr

Behové chyby – príklad (1)



Behové chyby – príklad (2)

java.lang.ArithmeticException: / by zero
at CiselnyDisplej.krok(CiselnyDisplej.java:61)



Behové chyby – príklad (3)

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

java.lang.ArithmeticException: / by zero
at CiselnyDisplej.krok(CiselnyDisplej.java:61)

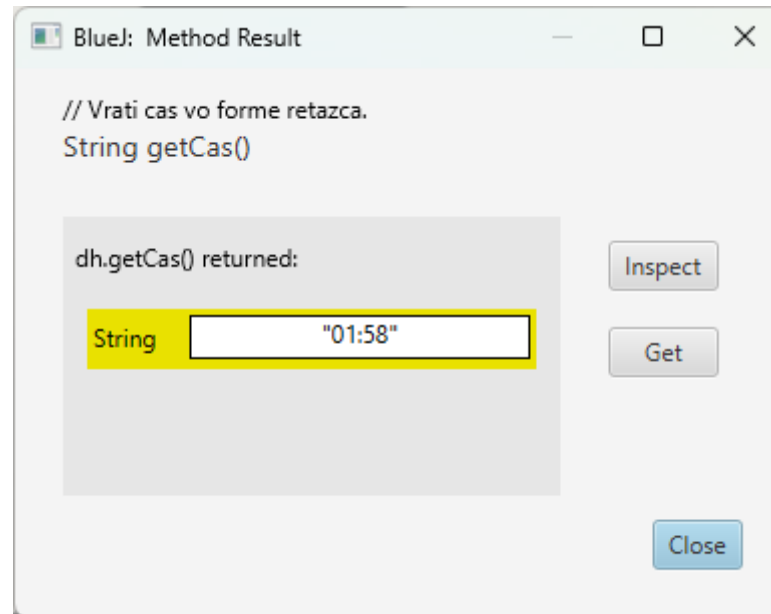
Behové chyby – príklad (4)

```
public CiselnýDisplej(int hornaHranica) {  
    this.hornaHranica = 0;  
    this.hodnota = 0;  
}
```

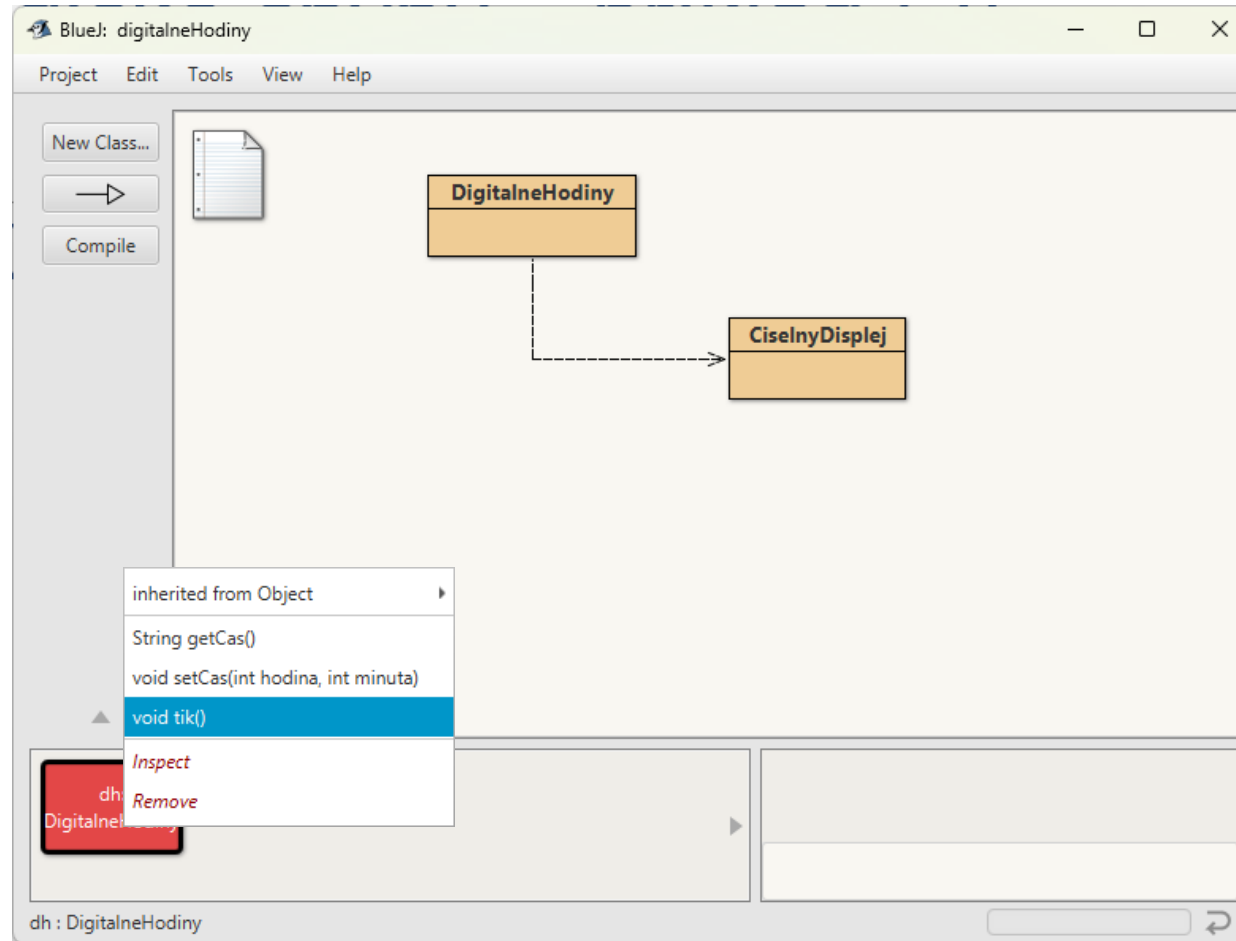
Logické chyby

- môže zistiť a „hlási“ používateľ programu
- program pracuje, ale jeho výsledky sú nesprávne
- najzákernejšie chyby

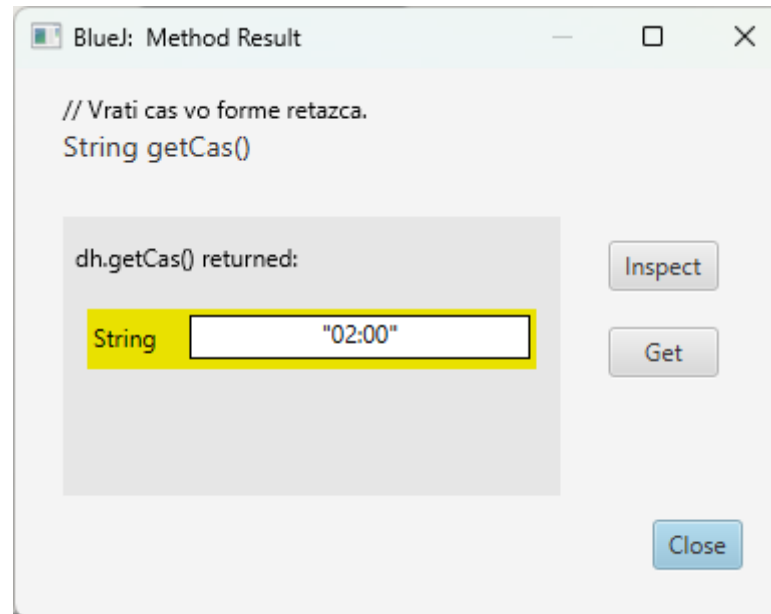
Logické chyby – príklad (1)



Logické chyby – príklad (2)



Logické chyby – príklad (3)



Logické chyby – príklad (4)

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

Logické chyby – príklad (5)

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

Logické chyby – príklad (6)

```
public CiselnýDisplej(int hornaHranica) {  
    this.hornaHranica = hornaHranica;  
    this.hodnota = 0;  
}
```

Logické chyby – príklad (7)

```
public DigitalneHodiny() {  
    this.hodiny = new CiselnýDisplej (23) ;  
    this.minuty = new CiselnýDisplej (59) ;  
}
```

Príklady „veľkých“ chýb v softvéri (1)

- Mars Climate Orbiter

- misia na orbitu Marsu, 1998-1999
- 9 mesiacov cesta
- \$327,6 miliónov
- zhorel v atmosfére Marsu
- nesprávne jednotky

- Mariner 1

- misia na pozorovanie Venuše, 1962
- trvala 294,5 s – odklonila sa z trajektórie a bola zničená
- \$18,5 miliónov
- programátor zle prepísal do kódu rukou písaný vzorec

Príklady „veľkých“ chýb v softvéri (2)

- bug v procesore Intel Pentium®
 - chyba pri delení v špecifických situáciách, 1993-1994
 - Intel bol nútený procesory vymeniť
 - \$ 475 000 000
- (takmer) krach firmy Knight
 - obchody na burze, 2012
 - pre chybu softvér nakúpil nevýhodné akcie
 - \$ 440 000 000
- Therac-25
 - Rádioterapia, 1985-1987
 - softvér nesprávne kontroloval ochranu pacientov pred radiáciou
 - 4 ľudia zomreli, 2 ľudia mali zdravotné následky z ožiarenia

Moja najväčšia chyba v softvéri

- vymazanie všetkých vlakov budúročného grafikonu ŽSR, 2012
- algoritmus na odstránenie vymazaných vlakov zo súboru

```
while (povodnySubor.suDalsieVlaky()) {  
    Vlak v = povodnySubor.citajDalsiVlak();  
    if (v.jeVymazany()) {  
        break; // ma byt continue  
    }  
    novySubor.ulozVlak(v);  
}  
povodnySubor.vymaz();  
novySubor.premenuj(povodnySubor.getNazov());
```

Techniky boja s chybami

- testovanie (testing)
- ladenie (debugging)
- písanie čitateľného kódu (readable code)

Testovanie

- proces overovania správneho fungovania programu
- testovanie fungovania časti aplikácie – testovanie komponentov
 - „komponent“ – skupina tried, trieda, metóda, skupina metód
- testovanie interakcie častí aplikácie – integračné testovanie
- testovanie funkčnosti celej aplikácie – systémové testovanie
- testovanie vhodnosti aplikácie pre používateľa – akceptačné testovanie

Biela a čierna skrinka

- testovanie bielej skrinky
 - k dispozícii aj vnútorný pohľad
 - využívajú sa znalosti o implementácii
 - napr. kontrola stavu objektu, kontrola podmienok podmienených príkazov a cyklov, ...
- testovanie čiernej skrinky
 - k dispozícii je iba rozhranie
 - kontrola reakcií na správu
 - kontrola zhody očakávaných a získaných výsledkov

Pozitívne a negatívne testovanie

- pozitívne testovanie

- kontrola prípadov, v ktorých sa očakáva úspešný výsledok
- operácie nesmú zlyhať pre žiadnu z povolených vstupných hodnôt

- negatívne testovanie

- testovanie prípadov, v ktorých sa očakáva zlyhanie
- informovanie o chybe – kontrola
- objekt sa nesmie dostať do nekorektného stavu, ani ak dostane neplatné vstupy

Spôsoby testovania

- manuálne testovanie
- automatizované testovanie

Manuálne testovanie (1)

- tester v úlohe používateľa (procesora)
- ideálne: tester nie je autor programu

Manuálne testovanie (2)

- prechádzanie zdrojového kódu
 - vizuálne prechádzanie štruktúrou programu
 - kontrola algoritmov
 - kontrola stavu objektu v rôznych fázach algoritmu vykonávanej testovanej metódy
- priama komunikácia s objektom
 - napr. v prostredí BlueJ
 - biela skrinka – využitie funkcie objekt inspector

Automatizované testovanie

- na testovanie sa vytvorí špecializovaný program – test
- test posiela správy testovanému programu, kontroluje odpovede
- výsledky prezentuje testerovi

Dôvody automatizovaného testovania

- testy sa vykonávajú opakovane
- manuálne testy
 - zdĺhavé – náročné na čas
 - náchylné na chyby – ľudský činiteľ
- automatizované testy
 - rýchle vykonanie testu
 - vždy rovnaký postup
 - automatizácia rutinnej práce

Testy regresie

- zásah do programu
 - rozšírenie programu
 - oprava chyby v programe
- zistiť, či nebola narušená zvyšná funkcionality programu
 - opakovať všetky doteraz napísané testy

Testovacie triedy

- unit test
- autori: Beck, Gamma
- automatizované testovanie častí programu
- priama podpora v rôznych programovacích jazykoch
- Java – knižnica JUnit

Testovacia trieda v JUnit

- jedna trieda = niekoľko testov jednej jednotky
 - názov končiaci slovom Test
- jedna metóda = jeden test
 - verejná metóda
 - bez parametrov a návratovej hodnoty
 - Metóda musí byť označená ako @Test

Príklad testu v JUnit (1)

```
import org.junit.Assert;  
import org.junit.Test;  
  
public class DigitalneHodinyTest {  
    ...  
}
```

Príklad testu v JUnit(2)

```
@Test
```

```
public void casPoInicializacii() {
```

```
    DigitalneHodiny dh = new DigitalneHodiny();
```

```
    Assert.assertEquals("00:00", dh.getCas());
```

```
}
```


Príklad testu v JUnit(3)

```
@Test
public void prechodCezHodinu() {
    DigitalneHodiny dh = new DigitalneHodiny();
    dh.setCas(0, 58);
    Assert.assertEquals("00:58", dh.getCas());
    dh.tik();
    Assert.assertEquals("00:59", dh.getCas());
    dh.tik();
    Assert.assertEquals("01:00", dh.getCas());
}
```

Správa assertEquals/assertNotEquals

```
Assert.assertEquals(ocakavana, skutocna);
```

- assert = tvrdiť, uistiť sa
- vyhodnocuje rovnosť parametrov
 - áno - test pokračuje
 - nie - test končí chybou
- assertEquals môže byť v každom teste použitý ľubovoľný počet krát
- assertEquals = opak assertEquals

Správa assertTrue/assertFalse

```
Assert.assertTrue(logickyVyras) ;
```

- assert = tvrdiť, uistiť sa
- vyhodnocuje hodnota pravdivostného výrazu
 - true - test pokračuje
 - false - test končí chybou
- assertFalse = opak assertTrue

Prípravky

- rôzne testy môžu pracovať s rovnakými objektmi
- prípravky (fixtures) – objekty prístupné vo všetkých testoch v jednom unit teste
- reprezentované atribútmi testovacej triedy
- vytvárajú sa v špeciálnej metóde setUp
- vytvoria sa pred spustením každého testu

Príklad testu v JUnit, Fixtures

```
private DigitalneHodiny dh;
```

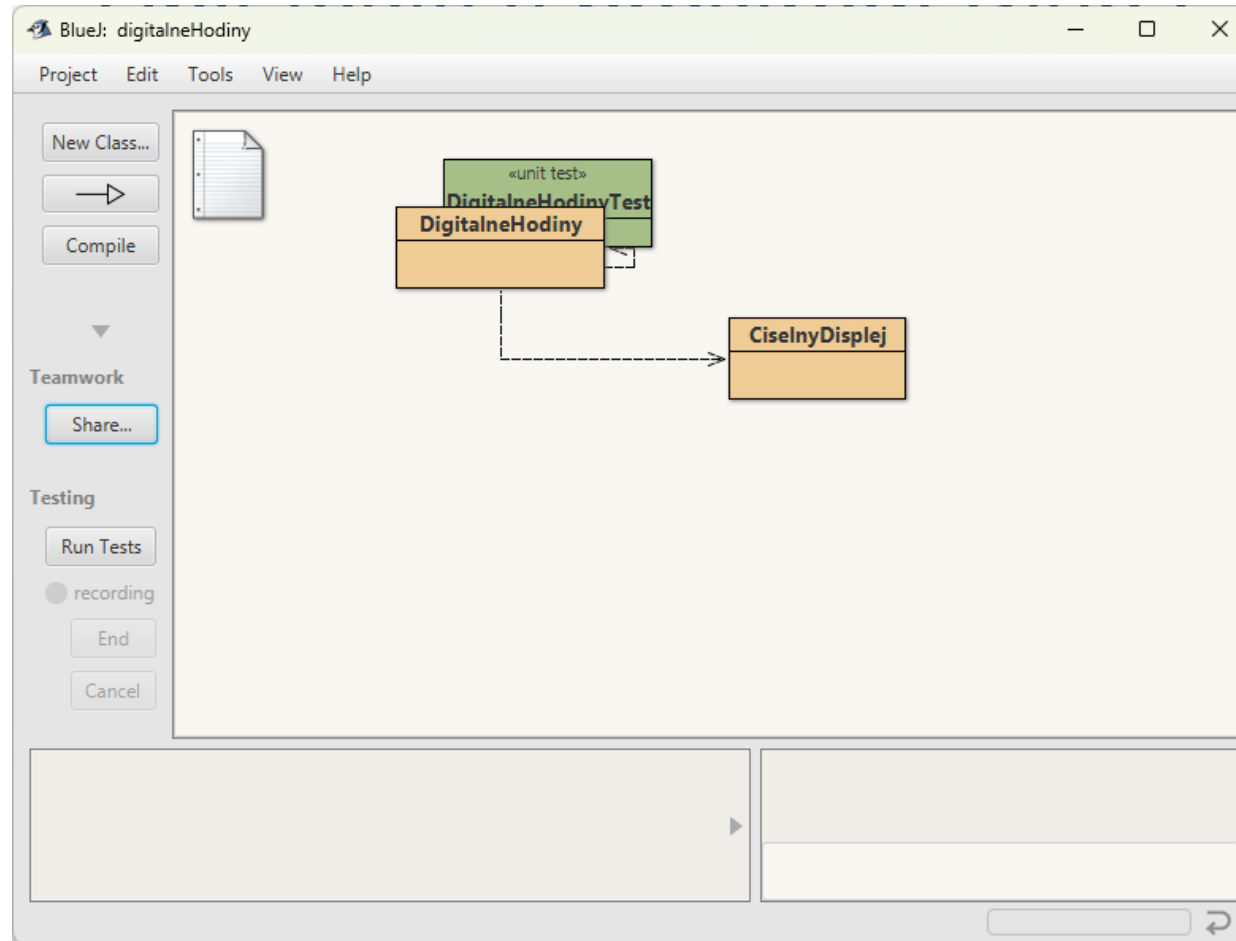
```
@Before
```

```
public void setUp() {  
    this.dh = new DigitalneHodiny();  
}
```

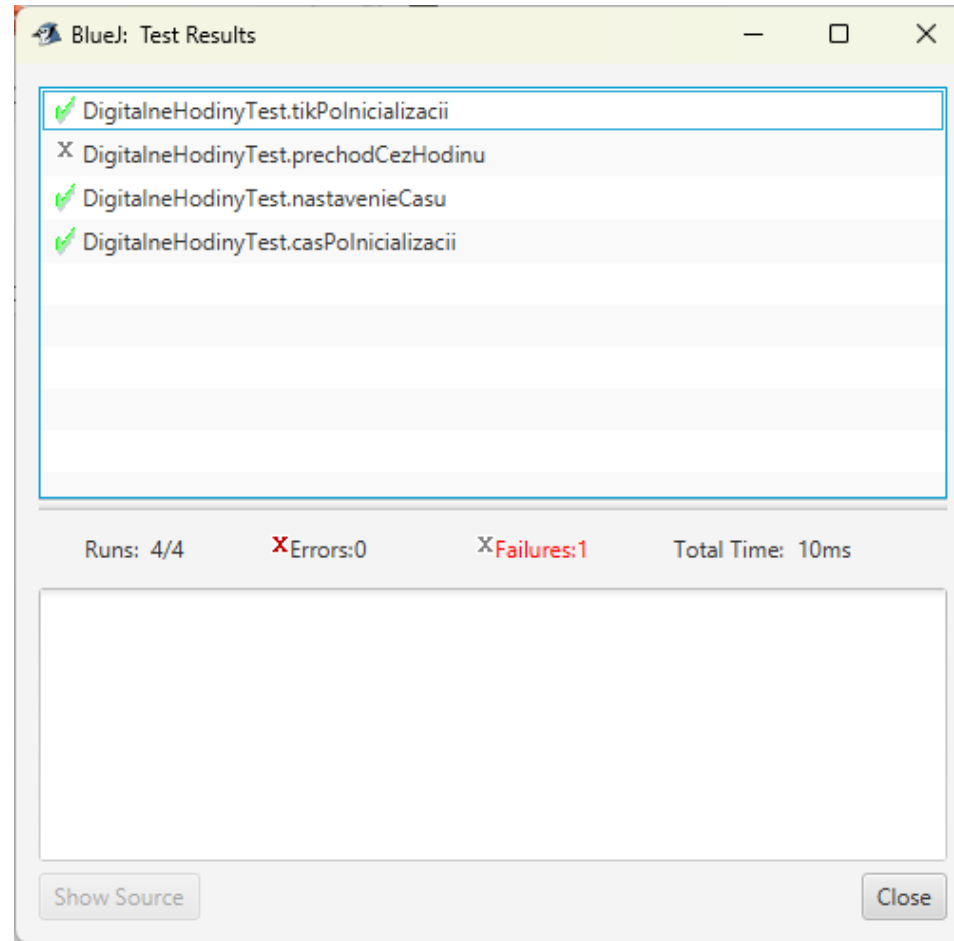
```
@Test
```

```
public void casPoInicializacii() {  
    Assert.assertEquals("00:00", this.dh.getCas());  
}
```

JUnit testy v prostredí BlueJ (1)



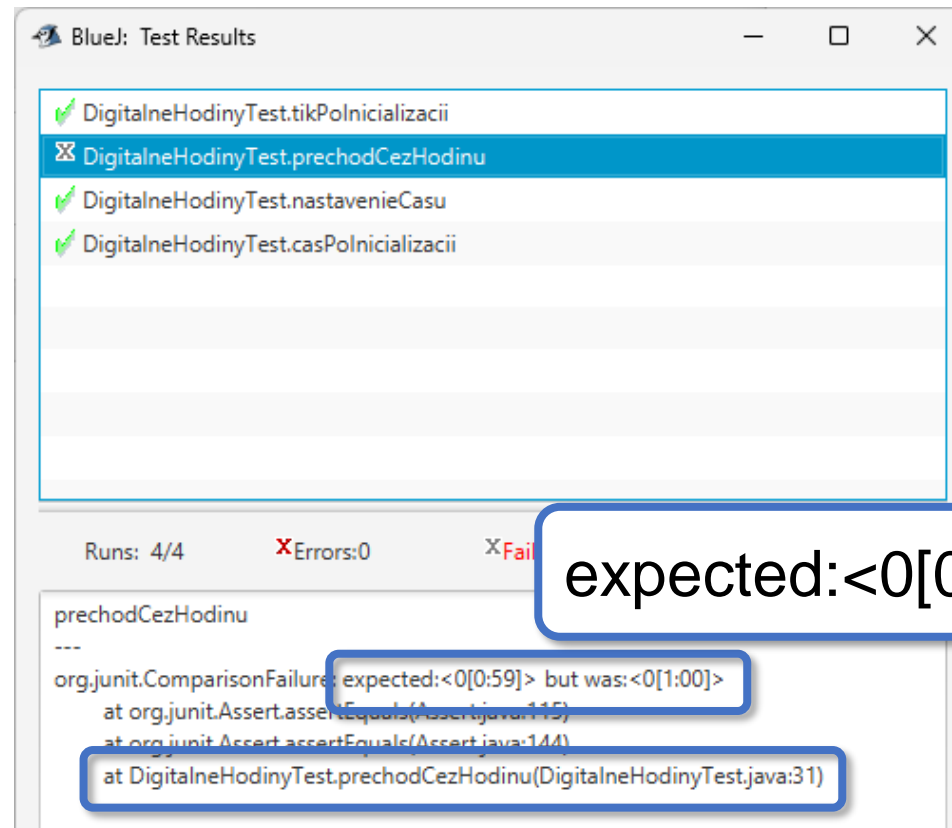
JUnit testy v prostredí BlueJ (2)



Stavy vykonaných testových metód

- ✓ daná testová metóda bola vykonaná a test prešiel bez chýb
- ✗ Daná testová metóda bola vykonaná a test zlyhal na niektorý assert
- ✗ Daná testová metóda bola vykonaná a test skončil behovou chybou

Detaily chyby v teste v prostredí BlueJ



expected:<0[0:59]> but was:<0[1:00]>

at DigitalneHodinyTest.prechodCezHodinu(DigitalneHodinyTest.java:31)

Hranice testovania

- úplne otestovať každý program vo všeobecnosti nie je možné
 - úspešný test nedokazuje, že program neobsahuje žiadnu chybu
 - čím viac chýb sa v programe nájde, tým viac ich program obsahuje
 - paradox pesticídov
-
- kombinácia viacerých spôsobov

Ladenie

- testovanie pomôže nájsť, že existuje chyba
- ladenie pomôže nájsť, kde sa tá chyba nachádza

Spôsoby ladenia

- manuálne prechádzanie kódu
- ladiace výpisy
- debugger

Manuálne prechádzanie kódu

- programátor otvorí zdrojový kód
- vizuálne prechádza zdrojový kód a hľadá chybu
 - manuálne vykonáva príkazy – je v úlohe procesora
 - zaznamenáva aktuálne hodnoty premenných
 - vyhodnocuje aktuálnu správnosť algoritmu
- jeden z najčastejších spôsobov ladenia

Ladiace výpisy

- rozšírenie programu o výpisy aktuálneho stavu objektov a algoritmov pomocou správy `System.out.println`
- programátor vo výpise vidí, kde sa objekty/algoritmy dostali do nesprávneho stavu
- ladiace výpisy môžu byť podmienené
 - zapoznámkovanie
 - ako vetva neúplného podmieneného príkazu
- ladiace výpisy môžu byť do súboru

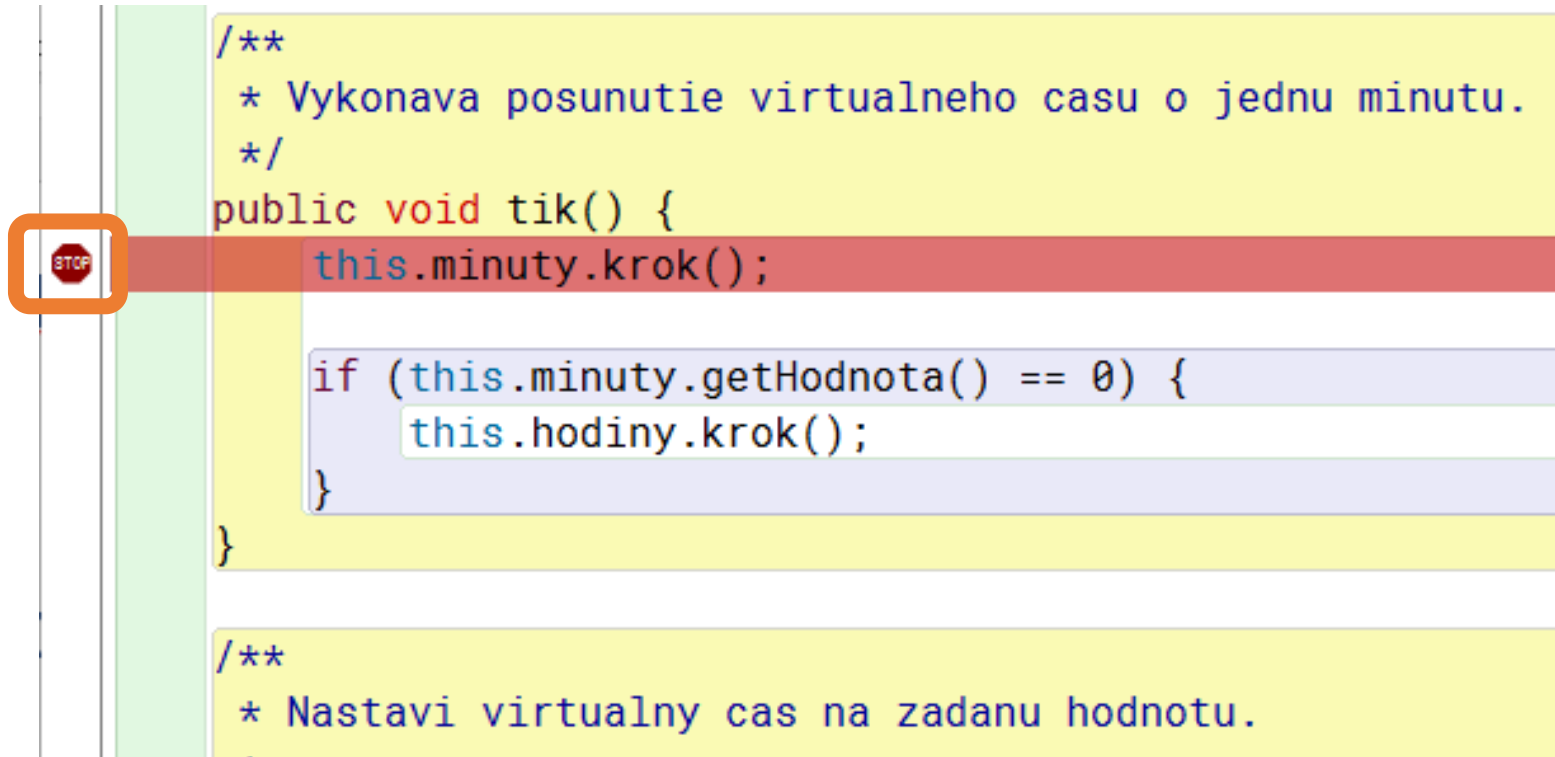
Debugger

- bug (chyba) = chyba v programe
- debugger – program asistujúci pri hľadaní chýb
 - zobrazuje hodnoty všetkých dostupných premenných
 - označuje príkaz, ktorý má byť aktuálne vykonaný
- „krokovanie“ programu
- možnosť nastavenia zarážok (breakpoint)
- programátor vyhodnocuje správnosť dosiahnutého stavu

Debugger v BlueJ (1)

- Nastavenie zarážky

- myšou
- Ctrl+B



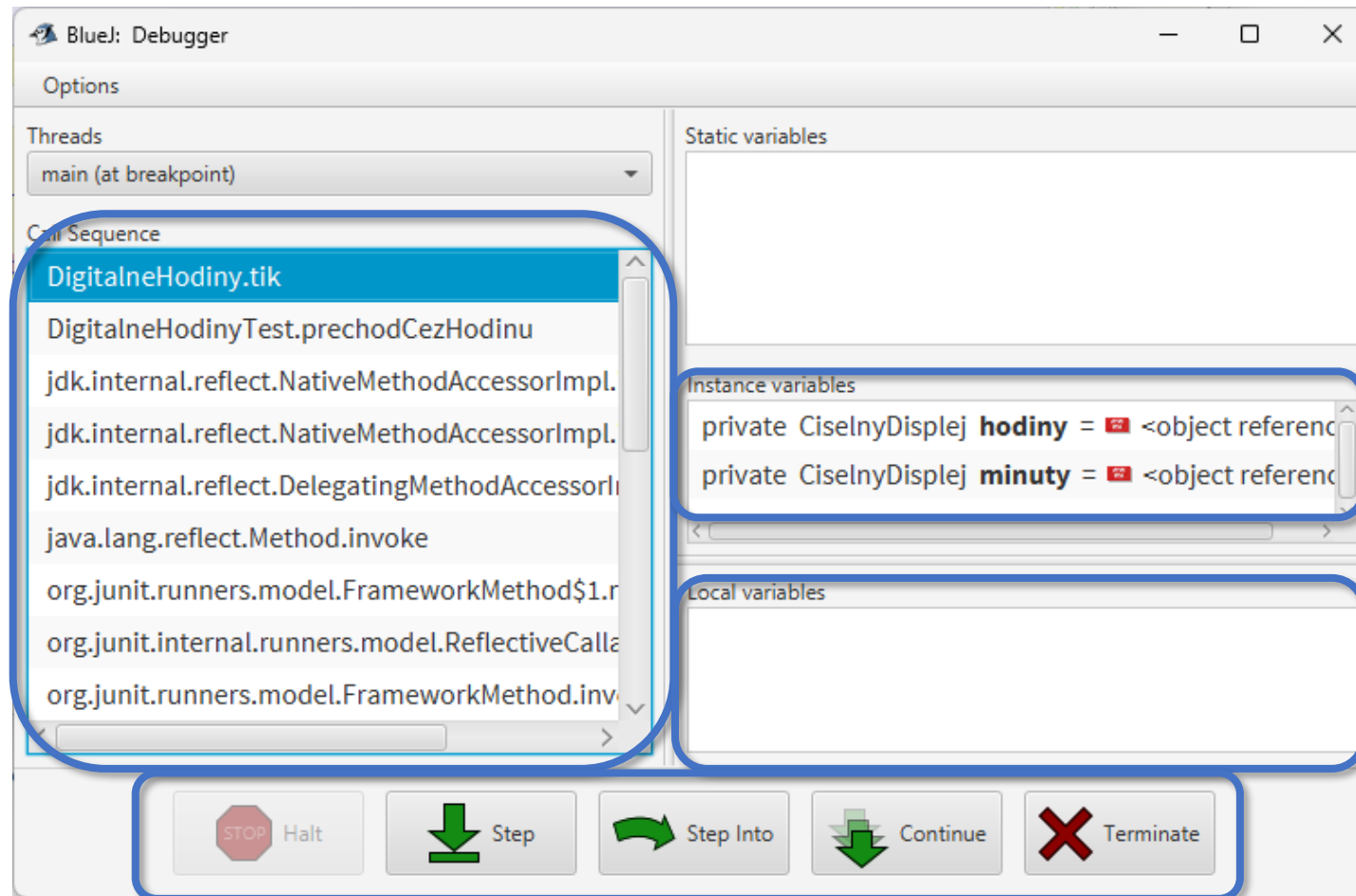
```
/**
 * Vykonava posunutie virtualneho casu o jednu minutu.
 */
public void tik() {
    this.minuty.krok();

    if (this.minuty.getHodnota() == 0) {
        this.hodiny.krok();
    }
}

/**
 * Nastavi virtualny cas na zadanu hodnotu.
 */
```


Debugger v BlueJ (2)

Zásobník



Atribúty prístupné
cez this

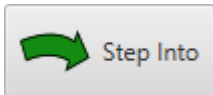
Lokálne
premenné

Ovládanie debuggera

Ovládanie debuggera



vykonať aktuálny príkaz a pokračovať ďalším



vykonať aktuálny príkaz a ak obsahuje poslanie správy, vstúpiť do vnoreného rámca



pokračovať vo vykonávaní po najbližšiu zarážku



ukončiť program

Nájdienie chyby (1)

- nájdeme chybné správanie krokovaním

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

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

- krok minúty zmení z 58 rovno na 0

Nájdenie chyby (2)

- manuálnym prechádzaním kódu, alebo ďalším krokováním hľadáme príčinu

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

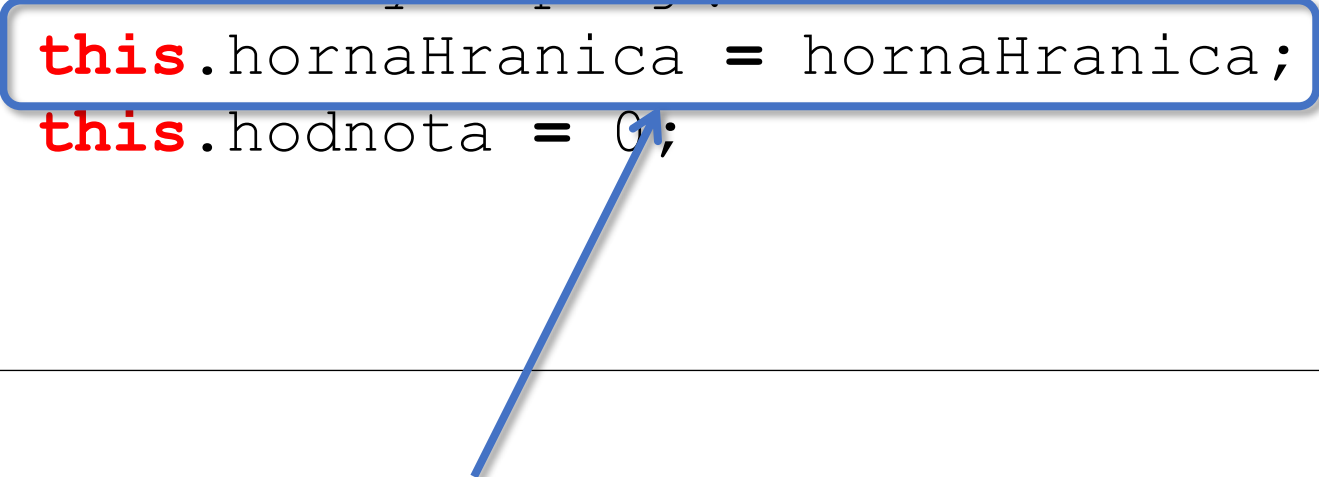


- algoritmus vyzerá byť v poriadku
- hornaHranica má byť 60
- krokováním overíme – nie je

Nájdenie chyby (3)

- manuálnym prechádzaním kódu, alebo ďalším krokováním hľadáme príčinu

```
public CiselnýDisplej(int hornaHranica) {  
    this.hornaHranica = hornaHranica;  
    this.hodnota = 0;  
}
```



- inicializácia je v poriadku, chybná je teda hodnota parametra

Nájdenie chyby (4)

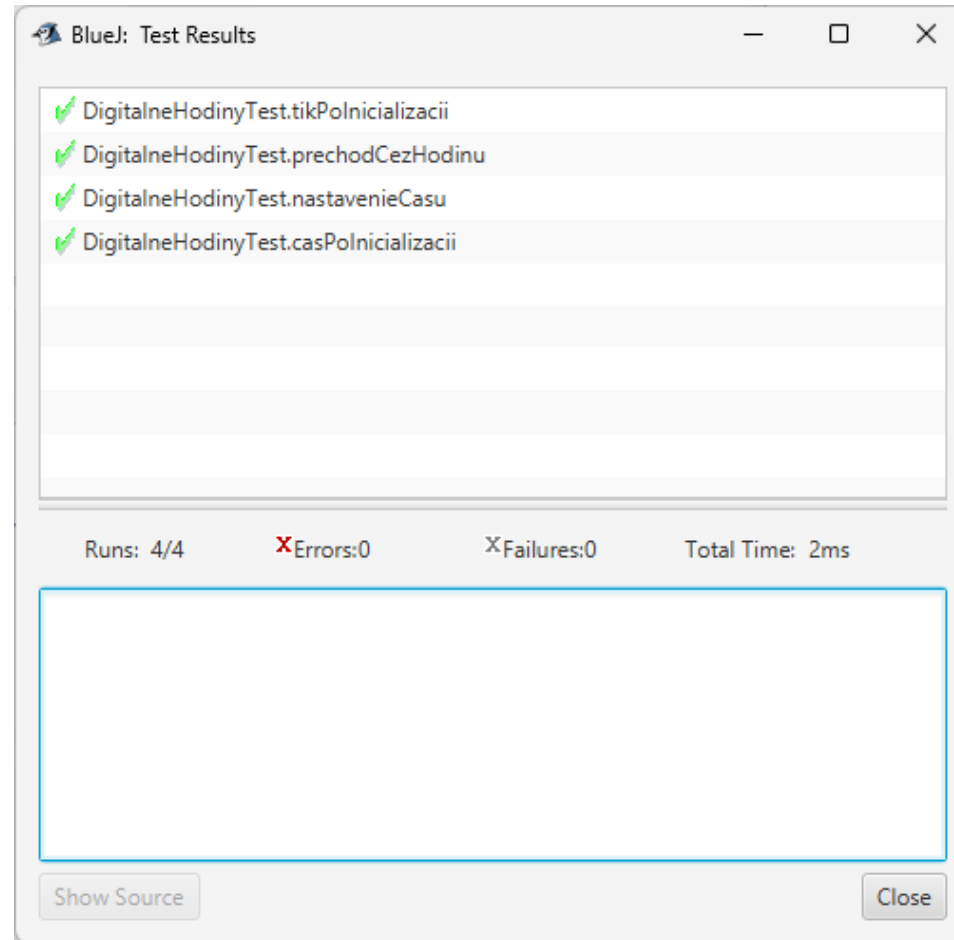
- manuálnym prechádzaním kódu, alebo ďalším krokováním hľadáme príčinu

```
public DigitalneHodiny() {  
    this.hodiny = new CiselnýDisplej(23);  
    this.minuty = new CiselnýDisplej(59);  
}
```

Oprava chyby

- oprava chyby – zmena zdrojového kódu
- => regresné testovanie

Výsledok po oprave



Písanie čitateľného kódu

- konvencie
- samopopisné identifikátory
- komentáre v zložitejších miestach algoritmu
- dokumentačné komentáre

Dokumentácia – forma rozhrania

Class CiselyDisplej

java.lang.Object
CiselyDisplej

```
public class CiselyDisplej  
extends Object
```

Trieda reprezentujúca jeden ciselny displej v digitálnych hodinách. Stará sa o zmenu hodnoty v zadanych hraniciach a o formátovanie čísla do tvaru dvojčífernej hodnoty.

Version:

2.0.0

Author:

David J. Barnes and Michael Kolling, Jan Janech

Constructor Summary

Constructors

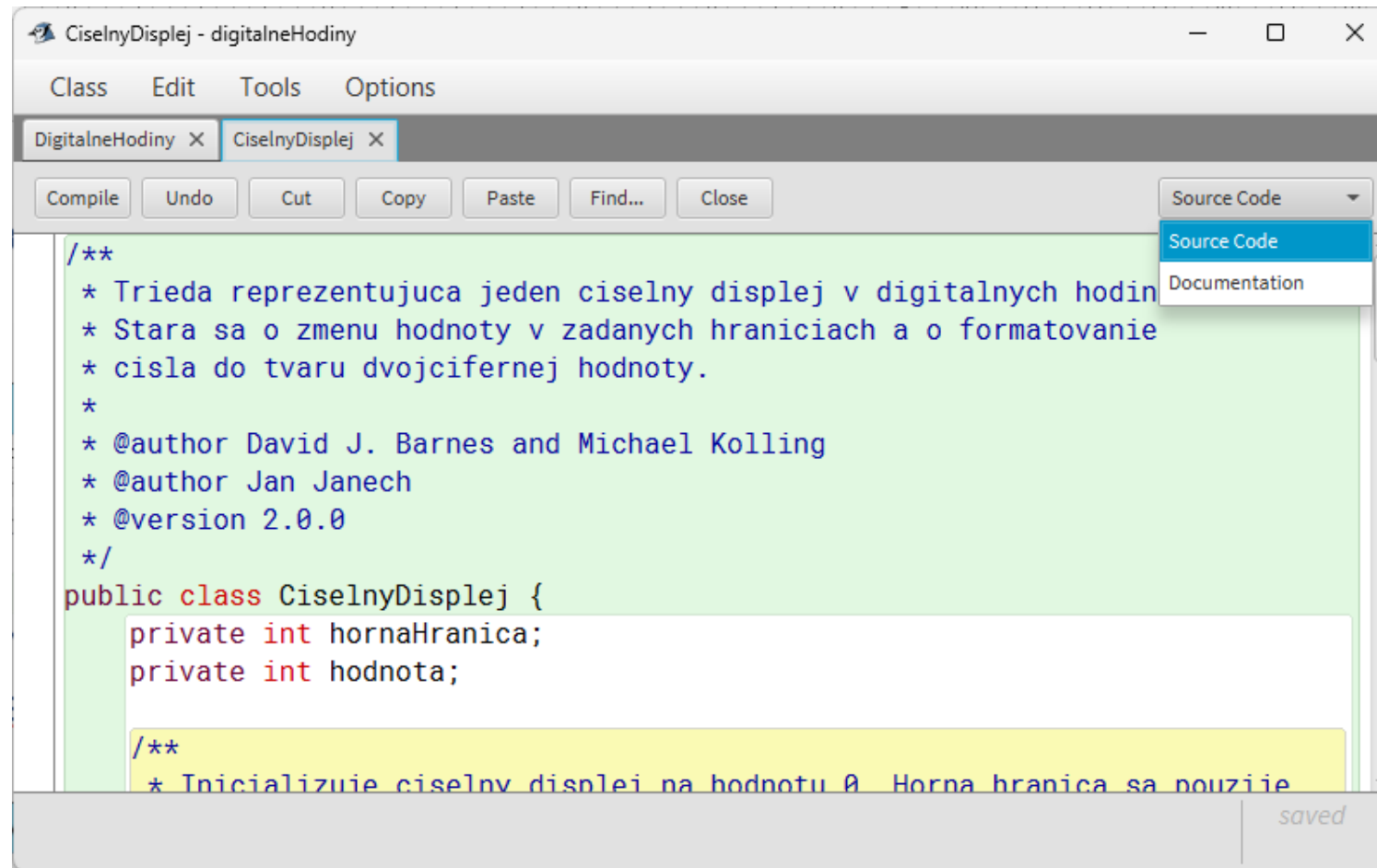
Constructor	Description
<code>CiselyDisplej(int hornaHranica)</code>	Inicializuje ciselny displej na hodnotu 0.

Method Summary

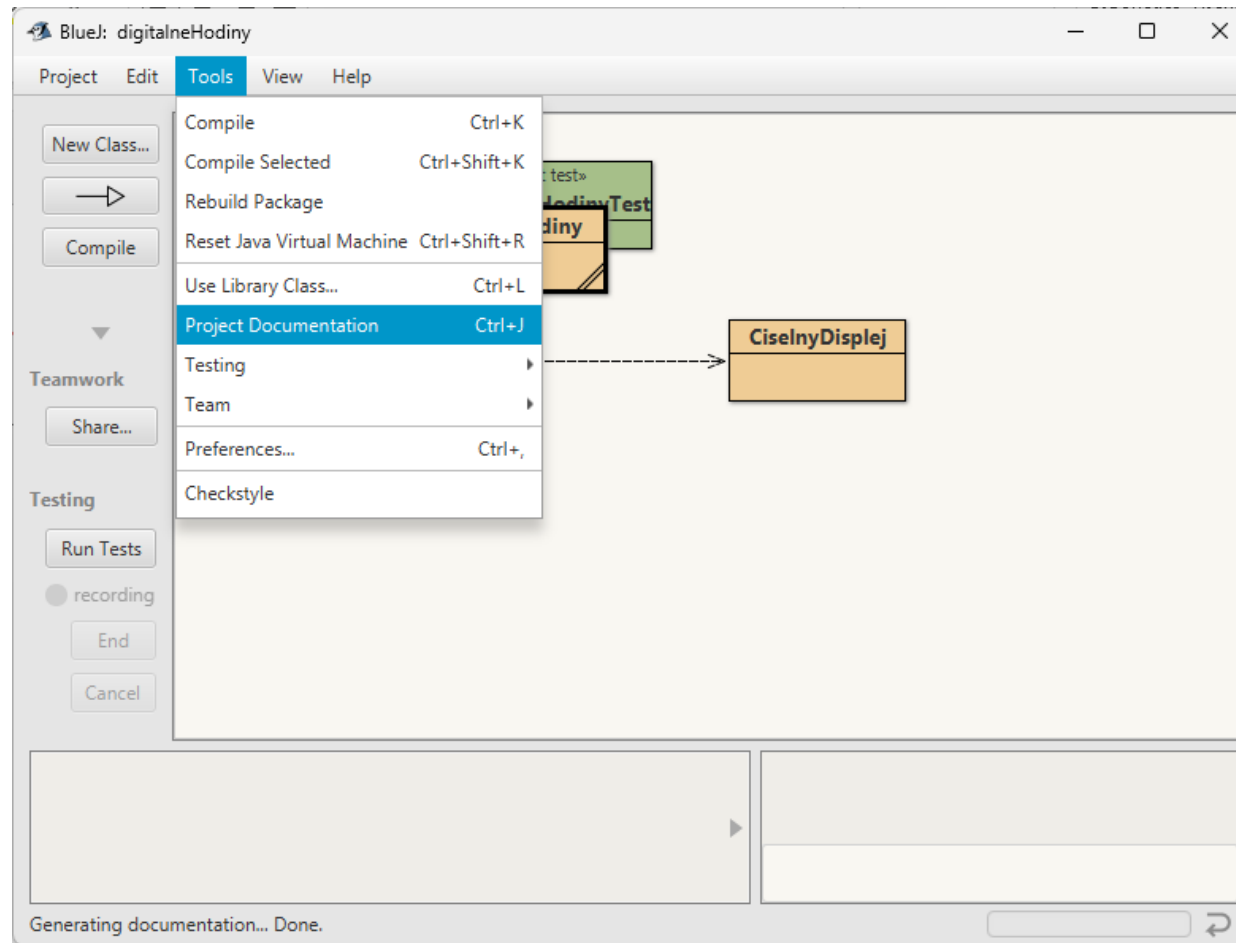
All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
int	<code>getHodnota()</code>	Vráti aktuálnu hodnotu ciselneho displeja vo forme celeho čísla typu int.
String	<code>getHodnotaAkoRetazec()</code>	Vráti hodnotu ciselneho displeja vo forme reťazca, pričom hodnota je vždy vo forme dvojčíferného čísla s prípadnou úvodnou nulou.
void	<code>krok()</code>	Zväčší hodnotu na ciselnom displeji o hodnotu jedna.
void	<code>setHodnota(int hodnota)</code>	Nastaví novú hodnotu ciselneho displeja vo forme celeho čísla typu int.

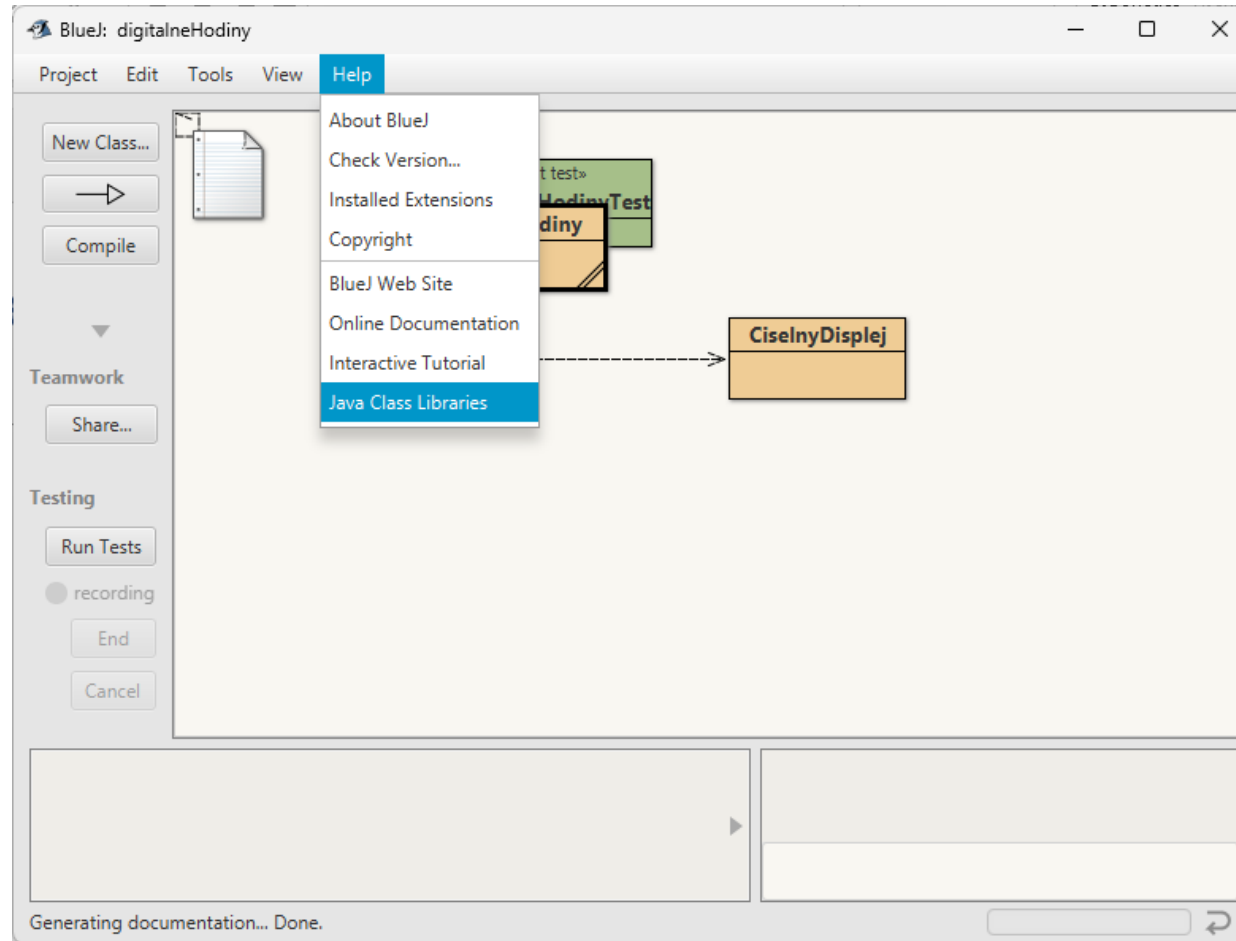
BlueJ – generovanie dokumentácie jednej triedy



BlueJ – generovanie dokumentácie celého projektu



BlueJ – dokumentácia štandardnej knižnice



Javadoc

- generátor dokumentácie objektu
- špeciálne komentáre – dokumentačné komentáre
- krátky popis nasledovaný tagmi

Javadoc – trieda – zdrojový kód

```
/**  
 * Jednoduché digitalne hodiny pracujuce s virtualnym casom.  
 * Pamataju si hodiny a minuty.  
 */  
*  
* @author David J. Barnes and Michael Kolling  
* @author Jan Janech  
* @version 2.0.0  
*/  
public class DigitalneHodiny {
```

Javadoc – trieda – dokumentácia

Class CislnyDisplej

java.lang.Object
CislnyDisplej

```
public class CislnyDisplej  
extends Object
```

Trieda reprezentujúca jeden ciselny displej v digitalných hodinách. Stará sa o zmenu hodnoty v zadanych hraniciach a o formatovanie čísla do tvaru dvojčífernej hodnoty.

Version:

2.0.0

Author:

David J. Barnes and Michael Kolling, Jan Janech

Javadoc – konštruktor – zdrojový kód

```
/**  
 * Inicializuje ciselny displej na hodnotu 0. Horna hranica sa pouzije  
 * ta, co zada pouzivatel v parametri.  
 *  
 * @param hornaHranica Predstavuje cislo, ktore hodnota ciselneho  
 * displeja nemoze dosiahnut.  
 */  
public CiselnyDisplej(int hornaHranica) {
```

Javadoc – konštruktor – dokumentácia

Constructor Summary

Constructors

Constructor	Description
<code>CiselnyDisplej(int hornaHranica)</code>	Inicializuje ciselny displej na hodnotu o.

Javadoc – konštruktor – detailná dokumentácia

Constructor Detail

CiselnyDisplej

```
public CiselnyDisplej(int hornaHranica)
```

Inicializuje ciselny displej na hodnotu o. Horna hranica sa pouzije ta, co zada pouzivatel v parametri.

Parameters:

hornaNanica - Predstavuje cislo, ktore hodnota ciselneho displeja nemoze dosiahnut.

Javadoc – metóda – zdrojový kód

```
/**  
 * Vráti hodnotu číselného displeja vo forme reťazca, pričom hodnota je  
 * vždy vo forme dvojčíferného čísla s prípadnou úvodnou nulou.  
 *  
 * @return Dvojčíferné číslo ako reťazec  
 */  
public String getHodnotaAkoRetazec() {
```

Javadoc – metóda – dokumentácia

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method	Description
int	<code>getHodnota()</code>	Vrati aktualnu hodnotu ciselneho displeja vo forme celeho cisla typu int.
String	<code>getHodnotaAkoRetazec()</code>	Vrati hodnotu ciselneho displeja vo forme retazca, pricom hodnota je vzdy vo forme dvojciferného čísla s prípadnou úvodnou nulou.
void	<code>krok()</code>	Zväčši hodnotu na číselnom displeji o hodnotu jedna.
void	<code>setHodnota(int hodnota)</code>	Nastavi novu hodnotu ciselneho displeja vo forme celeho cisla typu int.

Javadoc – metóda – detailná dokumentácia

getHodnotaAkoRetazec

```
public String getHodnotaAkoRetazec()
```

Vrati hodnotu ciselneho displeja vo forme retazca, pricom hodnota je vzdy vo forme dvojciferneho cisla s pripadnou uvodnou nulou.

Returns:

Dvojciferecne cislo ako retazec