



Mýliť sa je ľudské

# Cieľ prednášky

- chyby v softvéri
- vyhľadávanie a odstraňovanie chýb
- overovanie správnej funkcie softvéru
  
- príklad: projekt plánovací diár

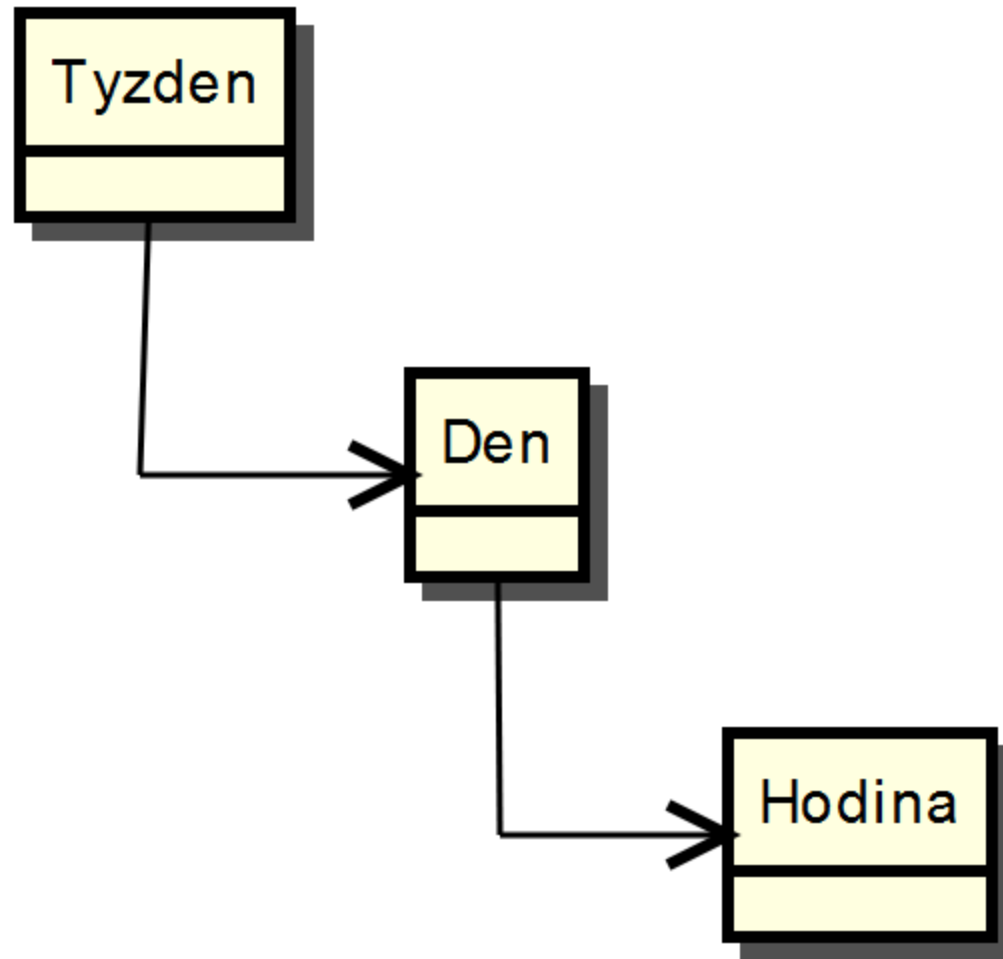
# Projekt planovacíDiar

- diár v prvom semestri – zápis poznámok
- nový diár – plánovanie úloh v rámci týždňa

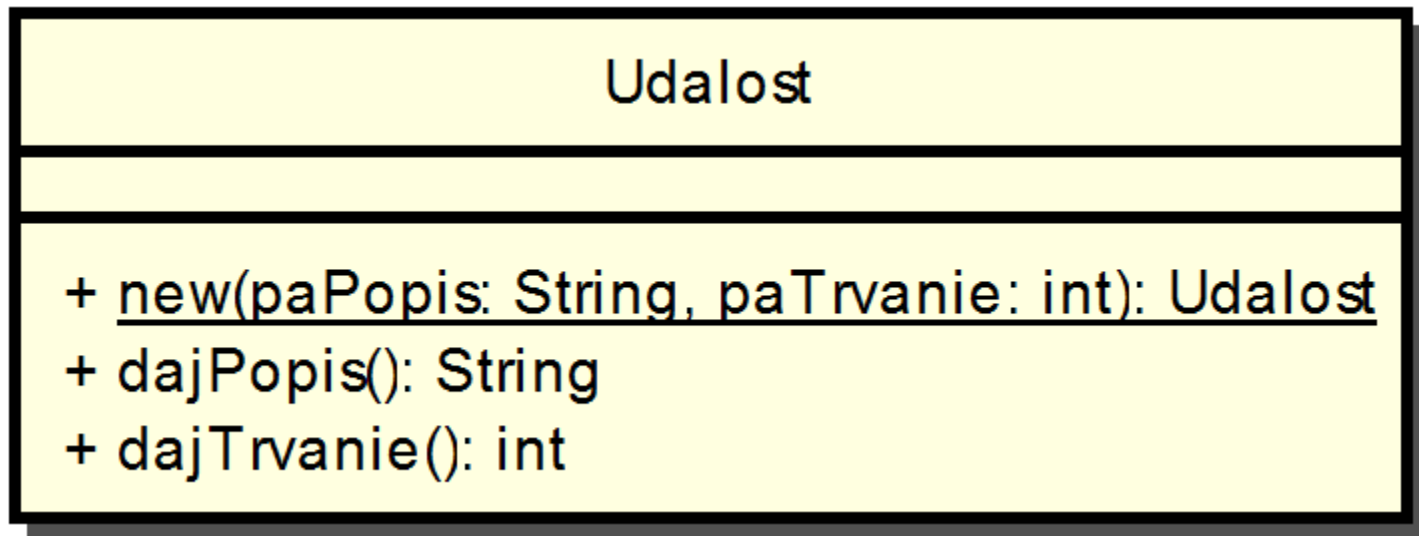
# Projekt planovacíDiar – zadanie

- udalosti – iba na pracovné dni
- pracovný čas od 9. hodiny do 17. hodiny
- začiatok udalosti – celá hodina
- trvanie udalosti – celé hodiny

# Projekt planovacíDiar – model



# Projekt planovaciDiar – trieda Udalost



# Projekt planovaciDiar – trieda Den

## Den

- + new(paCisloDna: int): Den
- + najdiPriestor(paUdalost: Udalost): int
- + vlozUdalost(paHodina: int, paUdalost: Udalost): boolean
- + dajUdalost(paHodina: int): Udalost
- + vypisUdalost(): void
- + dajCisloDna(): int
- + jePripustnaHodina(paHodina: int): boolean

# Projekt planovaciDiar – trieda Tyzden

## Tyzden

```
+ new(paCisloTyzdna: int): Tyzden  
+ vypisUdalosti(): void  
+ dajDen(paDenTyzdna: int): Den  
+ dajCisloTyzdna(): int
```



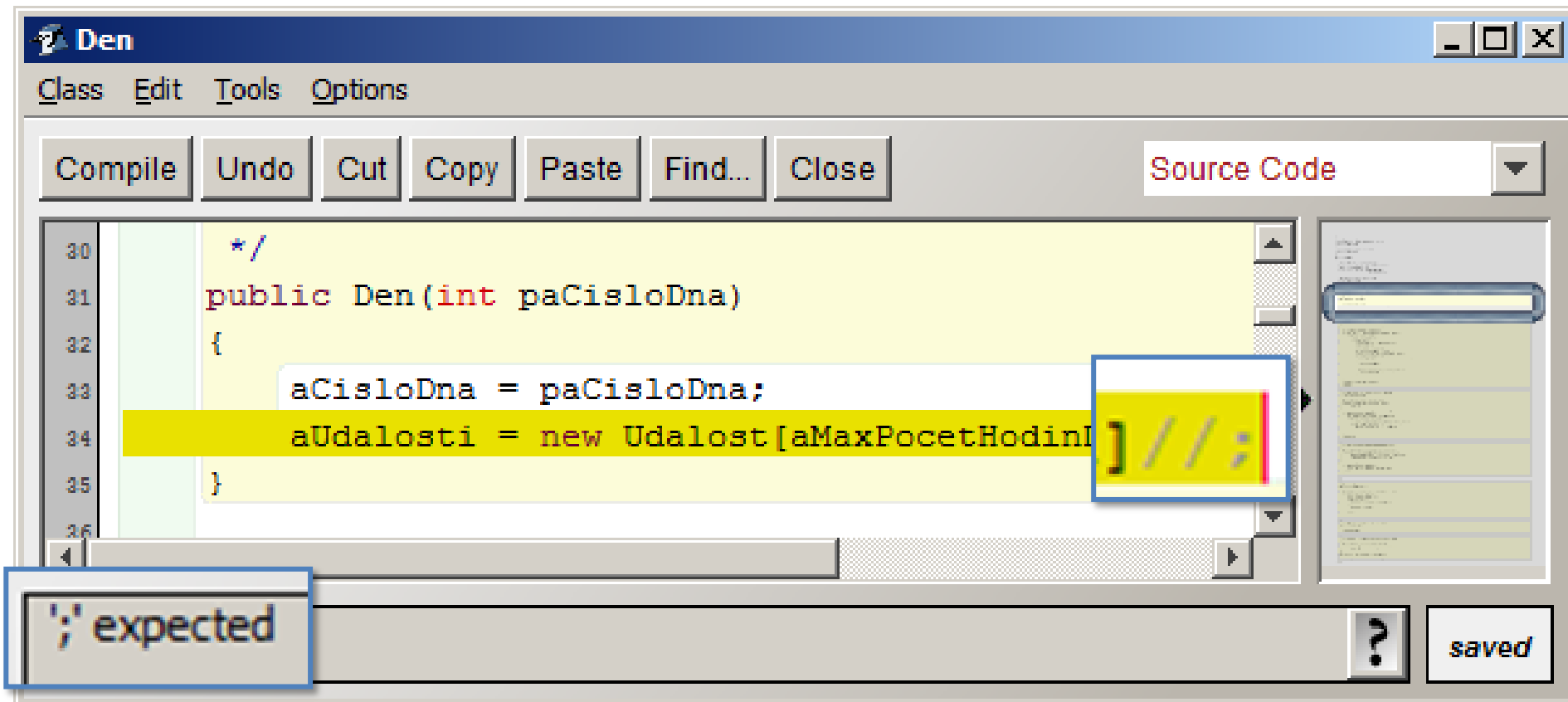
# Typy chýb

- 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<sub>(1)</sub>



# Syntaktické chyby – príklad<sub>(2)</sub>

The screenshot shows an IDE window titled "Den" with a menu bar (Class, Edit, Tools, Options) and a toolbar (Compile, Undo, Cut, Copy, Paste, Find..., Close). The source code is as follows:

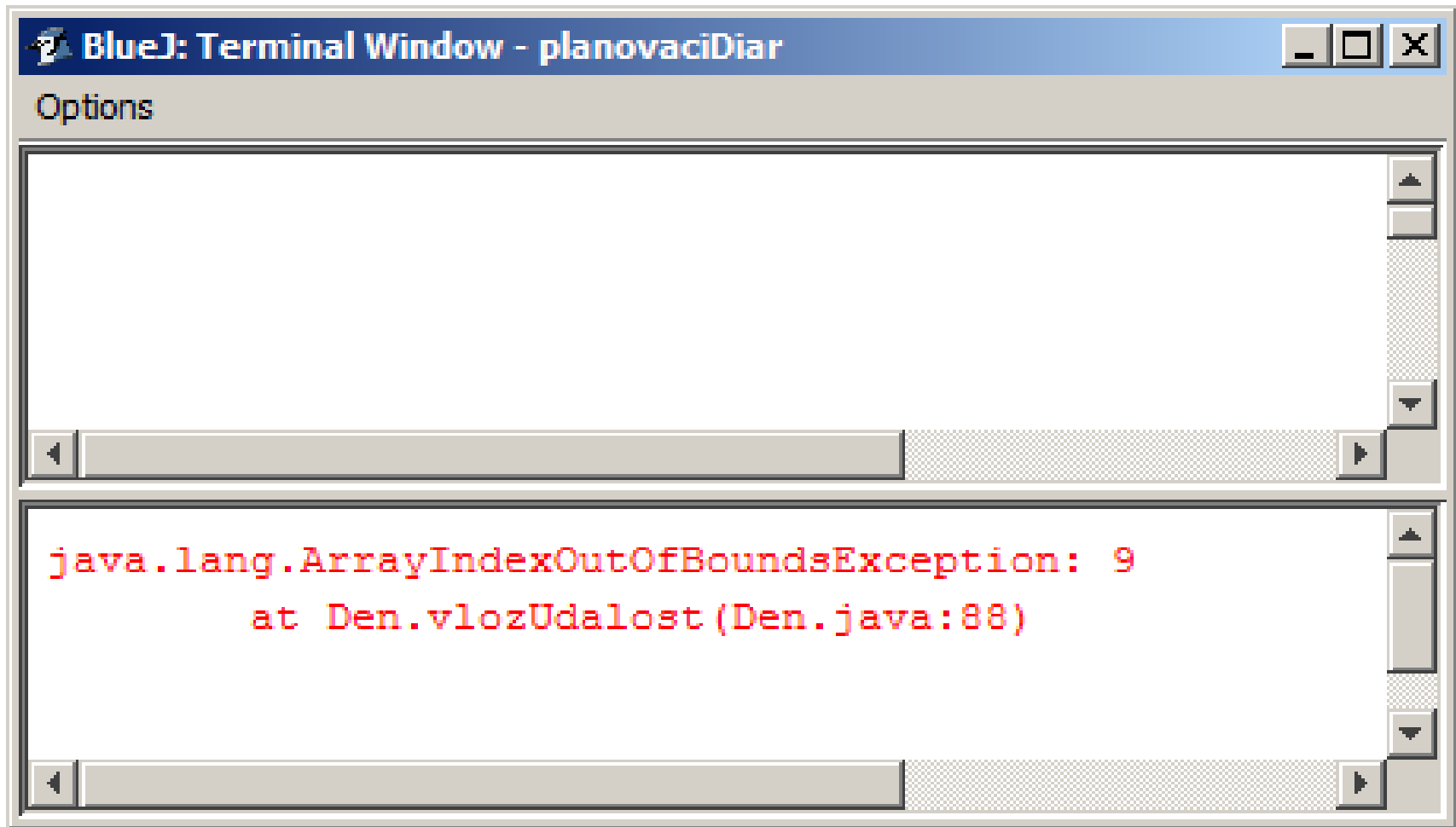
```
34 // }
35
36
37 /**
38  * Pokusi sa najst volny cas pre udalost.
39  *
40  * @param paUdalost udalost, ktora ma byt naplanov
41  * @return prva mozna hodina pre zaciatok udalosti
42  *         ak neexistuje volny cas pre udalost, vr
43  */
44 public int najdiPriestor(Udalost paUdalost)
45 {
46     int trvanie = paUdalost.dajTrvanie();
```

A blue box highlights the closing curly brace of the method on line 34. A tooltip at the bottom left of the IDE displays the error message "illegal start of expression".

# 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<sub>(1)</sub>



# Behové chyby – príklad<sub>(2)</sub>

The screenshot shows an IDE window titled "Den" with a menu bar (Class, Edit, Tools, Options) and a toolbar (Compile, Undo, Cut, Copy, Paste, Find..., Close). The main editor displays Java code for a method `vlozUdalost`. The code is as follows:

```
83  */
84  public boolean vlozUdalost(int paHodina, Udalost paUdalost) {
85  {
86      if (jePripustnaHodina(paHodina)) {
87          int indexZaciatku = paHodina - HodinaDobry;
88          if (aUdalosti[indexZaciatku] != null) {
89              // ...
90              // udalost je vlozena do kazdej hodiny,
91              for(int i = 0; i < trvanie; i++) {
```

The error message in the bottom-left corner is:

```
java.lang.ArrayIndexOutOfBoundsException:
9
```

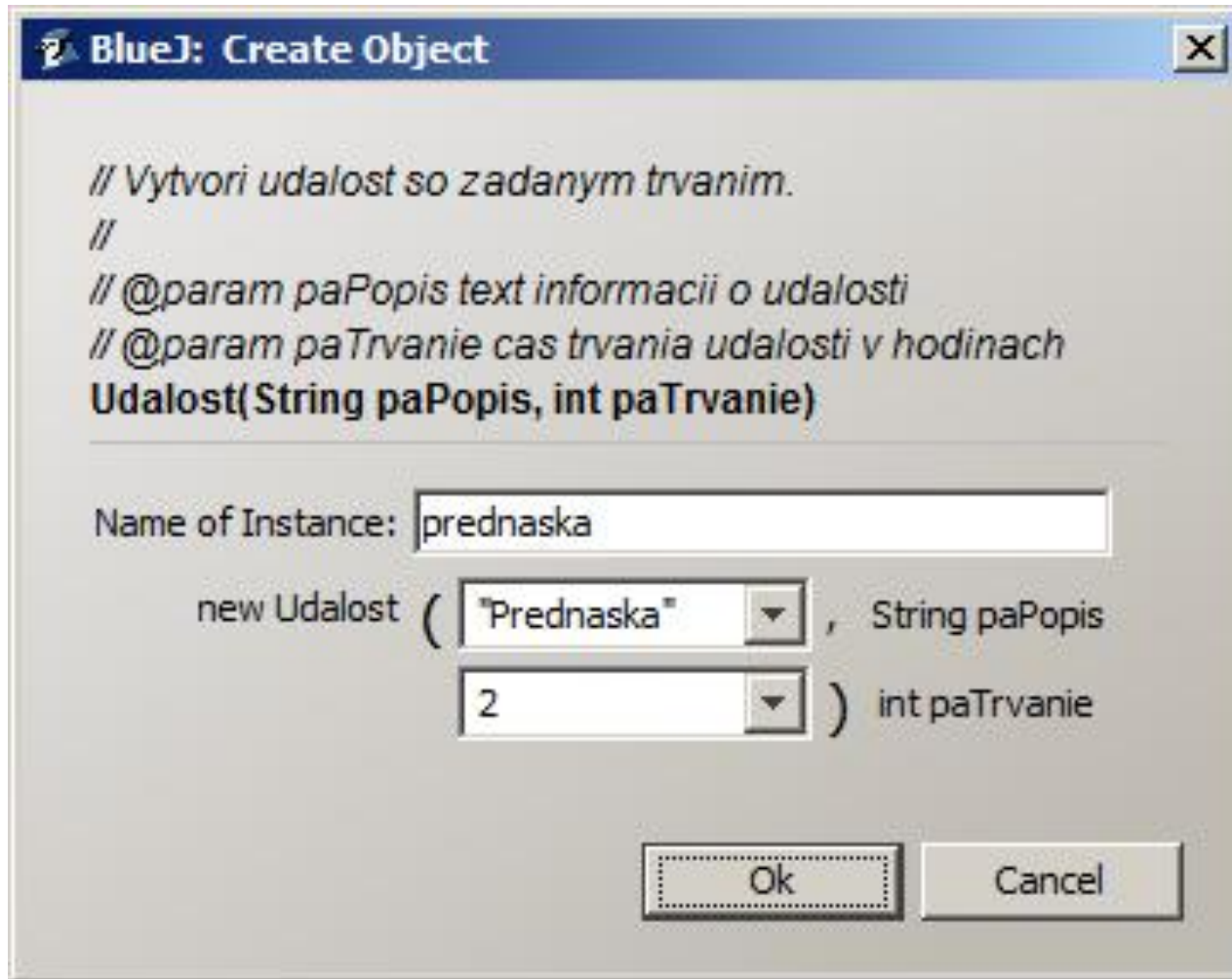
The IDE also shows a "Source Code" dropdown menu and a "saved" button in the bottom-right corner.

# Logické chyby

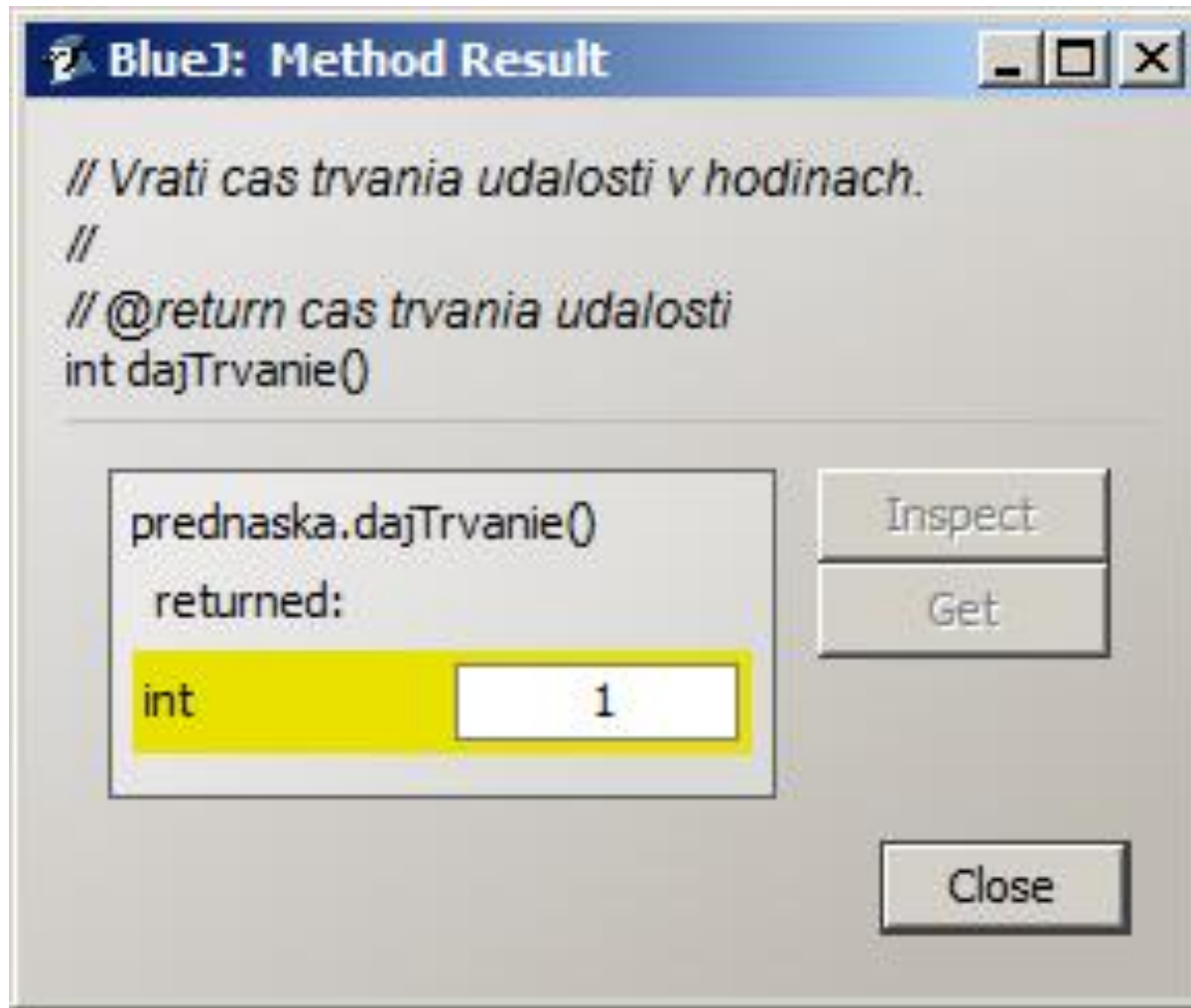
- môže zistiť a „hlási“ používateľ programu
- program pracuje, ale jeho výsledky sú nesprávne
- najzákernejšie chyby



# Logické chyby – príklad<sub>(1)</sub>



# Logické chyby – príklad<sub>(2)</sub>



# Logické chyby – príklad<sub>(3)</sub>

```
17
18 /**
19  * Vytvori udalost so zadanym trvanim.
20  *
21  * @param paPopis text informacii o udalosti
22  * @param paTrvanie cas trvania udalosti v hodinach
23  */
24 public Udalost(String paPopis, int paTrvanie)
25 {
26     this.aTrvanie = 1;
27 }
28
```

saved

# Techniky boja s chybami

- testovanie (testing)
- ladenie (debuging)
- písanie udržovateľného kódu  
(maintainable code)

# Testovanie

- proces overovania správneho fungovania programu
- testovanie fungovania celej aplikácie – aplikačné testovanie (application testing)
- testovanie fungovania časti aplikácie – testovanie jednotiek (unit testing)
  - „jednotka“ – skupina tried, trieda, metóda, skupina metód

# 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
- automatické



# Manuálne testovanie jednotiek<sub>(1)</sub>

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

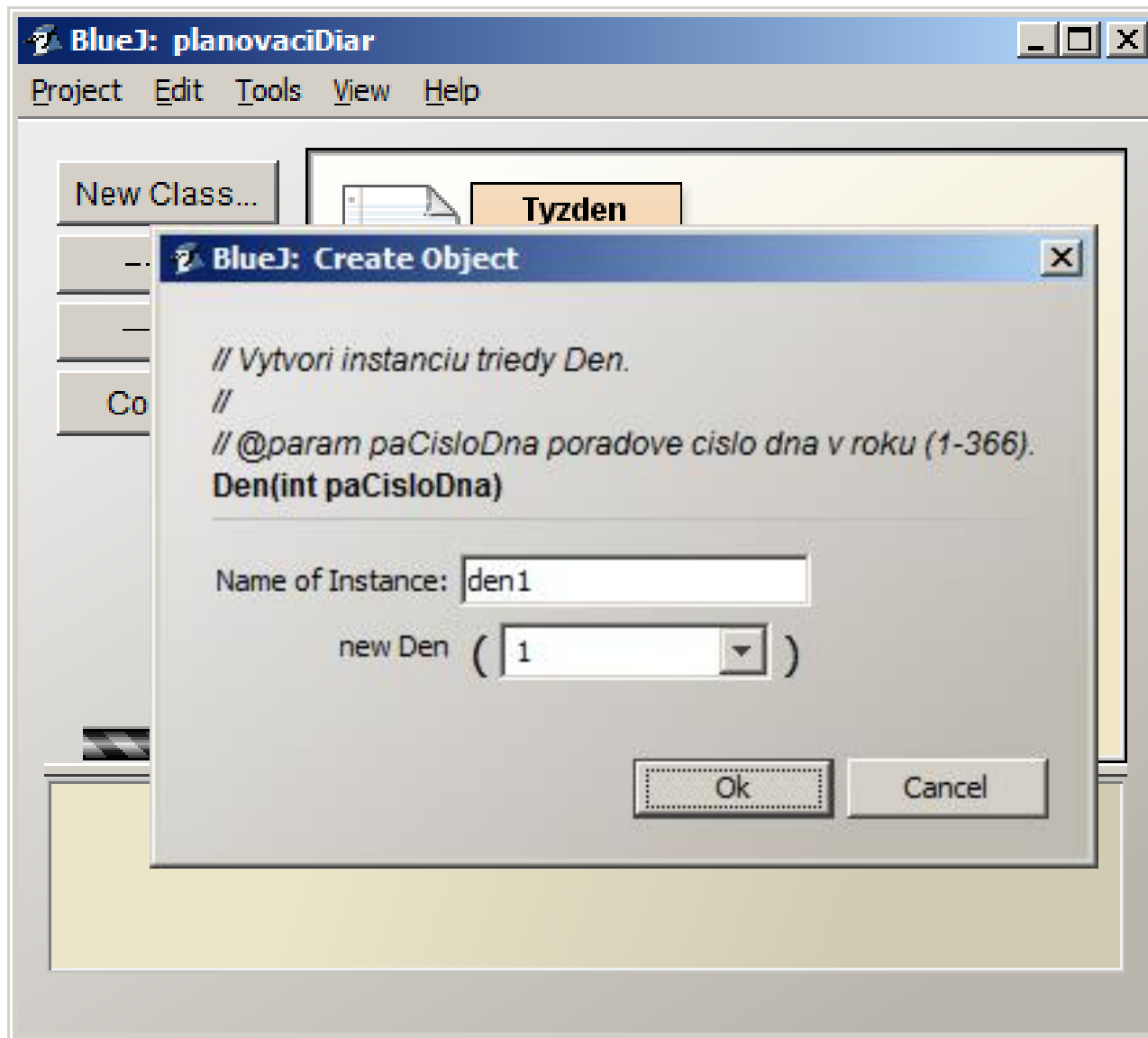
# Manuálne testovanie jednotiek<sub>(2)</sub>

- 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

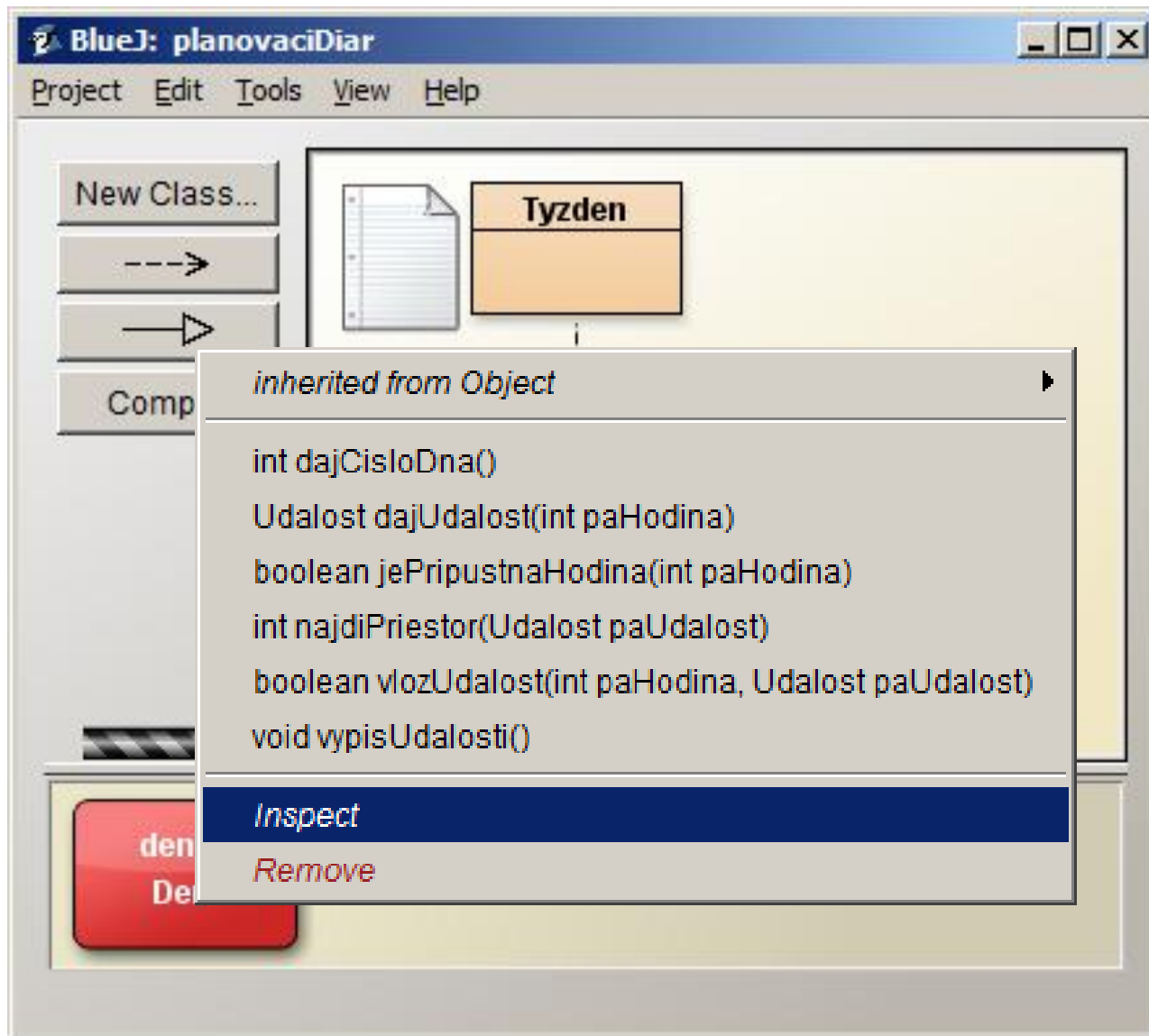
# Využitie funkcie objekt inspector

- sledovanie reakcie objektu na správu
- object inspector ostáva otvorený
- kontrola stavu atribútov
  - trieda
  - inštancia

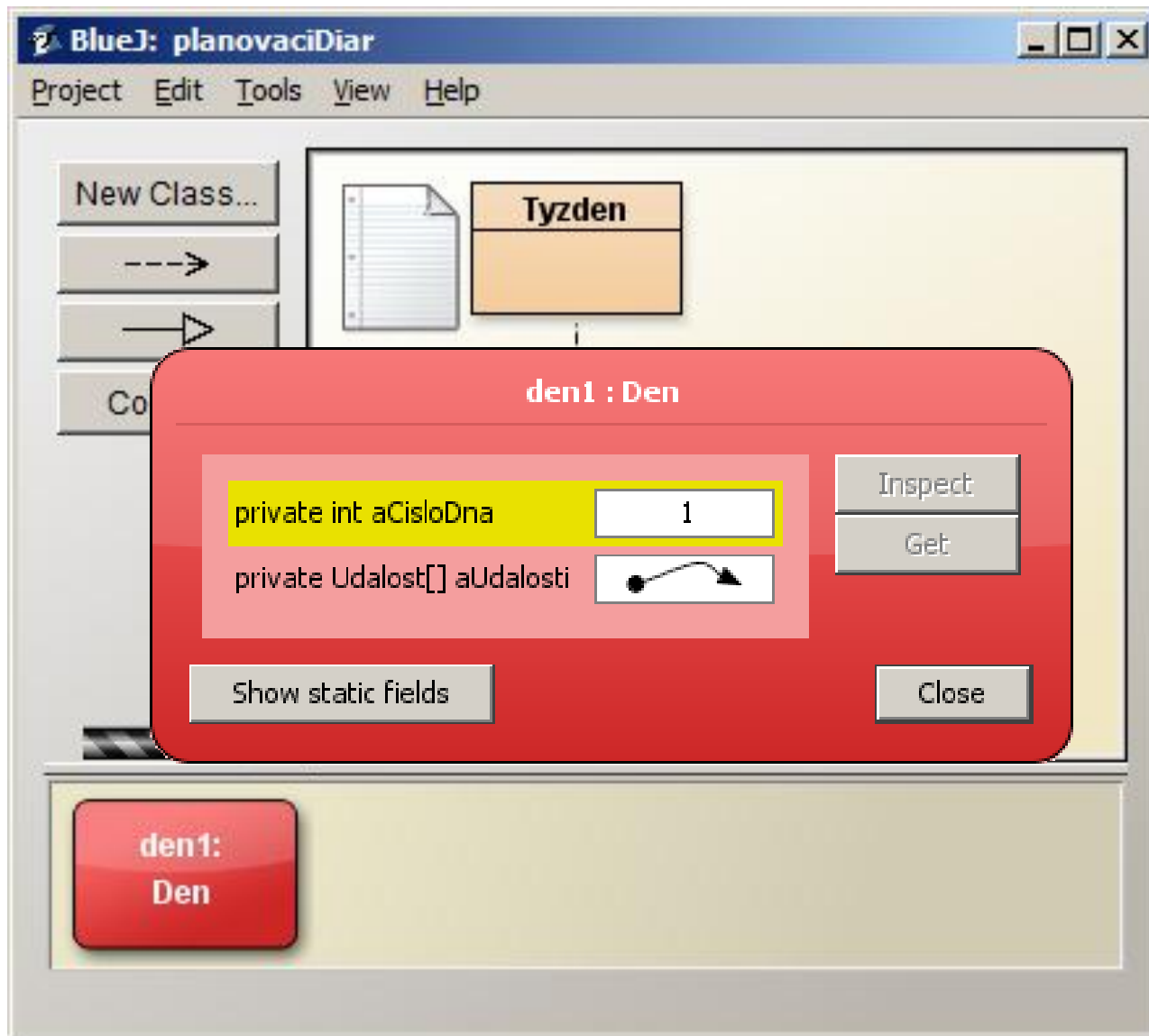
# Object inspector – príklad<sub>(1)</sub>



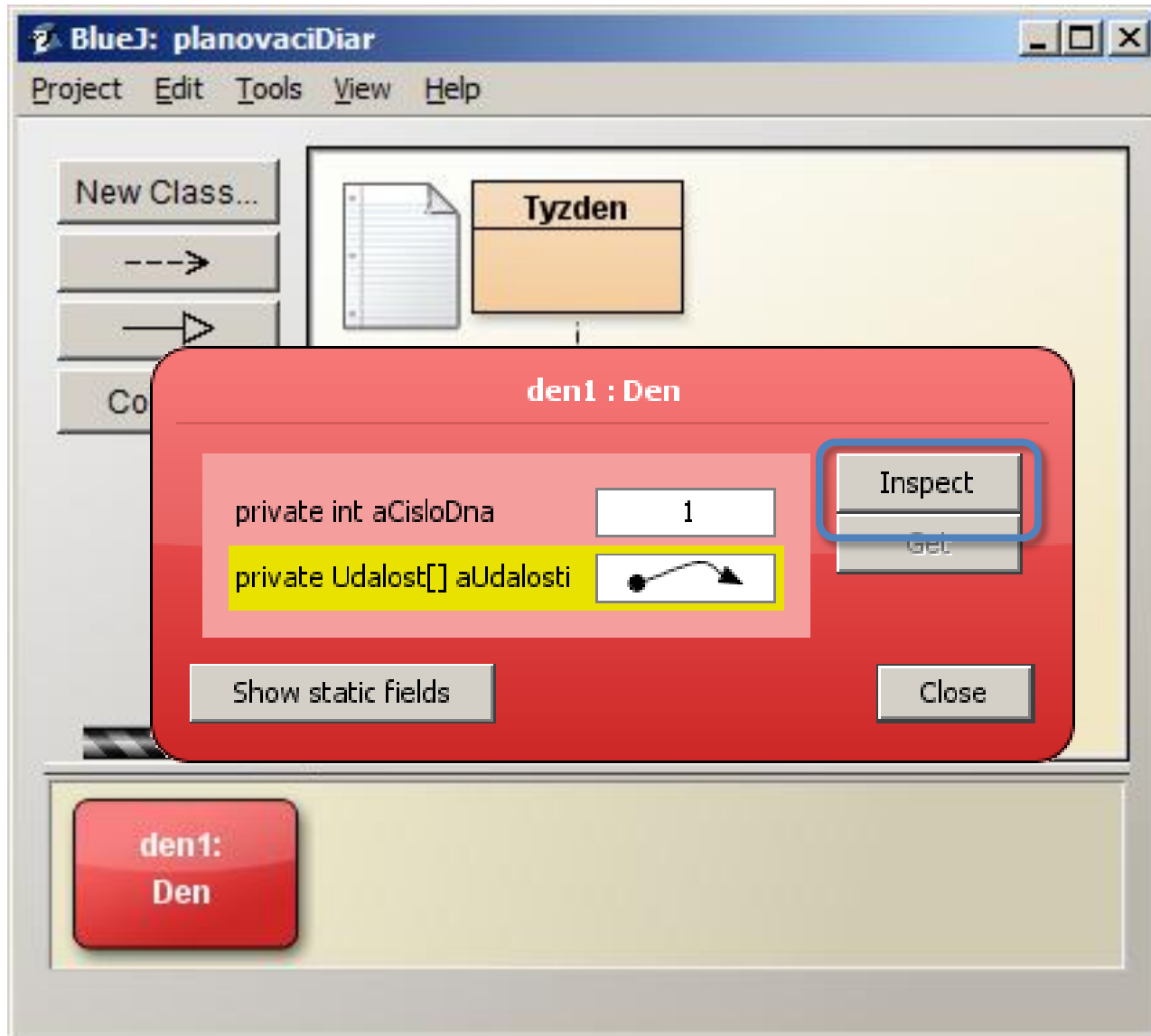
# Object inspector – príklad<sub>(2)</sub>



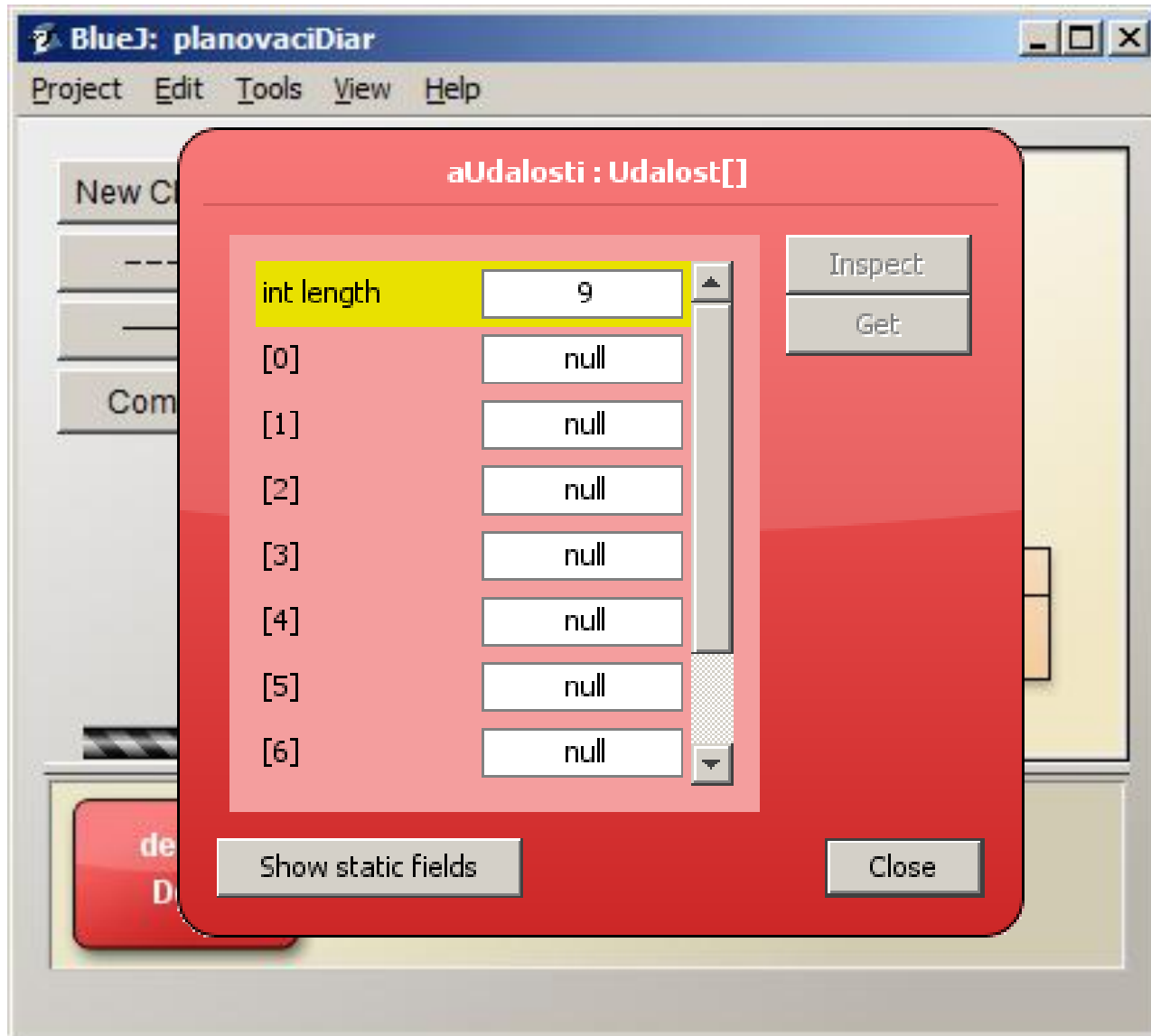
# Object inspector – príklad<sub>(3)</sub>



# Object inspector – príklad<sub>(4)</sub>



# Object inspector – príklad<sub>(5)</sub>





# Automatické testovanie

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

# Dôvody automatického testovania

- testy sa vykonávajú opakovane
- manuálne testy
  - zdĺhavé – náročné na čas
  - náchylné na chyby – ľudský činiteľ
- automatické 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

# TDD – Test driven development

- vývoj založený na testoch
- testy sa napíšu skôr ako sa začne s vývojom programu
- v každej fáze vývoja sa dá jednoducho skontrolovať funkčnosť programu
- Kent Beck: Programování řízené testy, Grada, ISBN 80-247-0901-5

# Testovacie triedy

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

# Testovacia trieda v JUnit

- jedna trieda = niekoľko testov jednej jednotky
  - špeciálne klauzule v hlavičke – preberieme neskôr
- jedna metóda = jeden test
  - verejná metóda
  - bez parametrov a návratovej hodnoty
  - ! názov musí začínať slovom test

# Príklad testu v JUnit<sub>(1)</sub>

```
import static org.junit.Assert.*;  
import org.junit.Before;  
import org.junit.Test;
```

```
public class TestDiara  
{  
    @Before  
    public void setUp()  
    {  
    }  
    ...  
}
```

# Príklad testu v JUnit<sub>(2)</sub>

```
@Test
```

```
public void testVytvorTriUdalosti()
```

```
{
```

```
    Den den1 = new Den(1);
```

```
    Udalost vymysliet = new Udalost("Vymysliet", 1);
```

```
    Udalost vykonat = new Udalost("Vykonat", 1);
```

```
    Udalost zabudnut = new Udalost("Zabudnut", 1);
```

```
    assertTrue(den1.vlozUdalost(9, vymysliet));
```

```
    assertTrue(den1.vlozUdalost(10, vykonat));
```

```
    assertTrue(den1.vlozUdalost(11, zabudnut));
```

```
}
```



# Príklad testu v JUnit<sub>(3)</sub>

```
@Test
```

```
public void testOtestujUdalost()
```

```
{
```

```
    Udalost vymysliet = new Udalost("Vymysliet", 1);
```

```
    Udalost vykonat = new Udalost("Vykonat", 2);
```

```
    assertEquals(1, vymysliet.dajTrvanie());
```

```
    assertEquals(2, vykonat.dajTrvanie());
```

```
    assertEquals("Vymysliet", vymysliet.dajPopis());
```

```
    assertEquals("Vykonat", vykonat.dajPopis());
```

```
}
```

# Správa assertEquals

```
this.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

# Správa assertTrue

```
this.assertTrue(pravdivostnyVyras);
```

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

# 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<sub>(1)</sub>

```
private Den den1;  
private Udalost vymysliet;  
private Udalost vykonat;  
private Udalost zabudnut;
```

@Before

```
public void setUp()  
{  
    den1 = new Den(1);  
    vymysliet = new Udalost("Vymysliet", 1);  
    vykonat = new Udalost("Vykonat", 1);  
    zabudnut = new Udalost("Zabudnut", 1);  
}
```

# Príklad testu v Junit, Fixtures<sub>(2)</sub>

```
@Test
```

```
public void testVytvorTriUdalosti()
```

```
{
```

```
    assertTrue(den1.vlozUdalost(9, vymysliet));
```

```
    assertTrue(den1.vlozUdalost(10, vykonat));
```

```
    assertTrue(den1.vlozUdalost(11, zabudnut));
```

```
}
```

# Príklad testu v Junit, Fixtures<sub>(3)</sub>

@Test

```
public void testOtestujUdalost()
```

```
{
```

```
    assertEquals(1, vymysliet.dajTrvanie());
```

```
    assertEquals(2, vykonat.dajTrvanie());
```

```
    assertEquals("Vymysliet", vymysliet.dajPopis());
```

```
    assertEquals("Vykonat", vykonat.dajPopis());
```

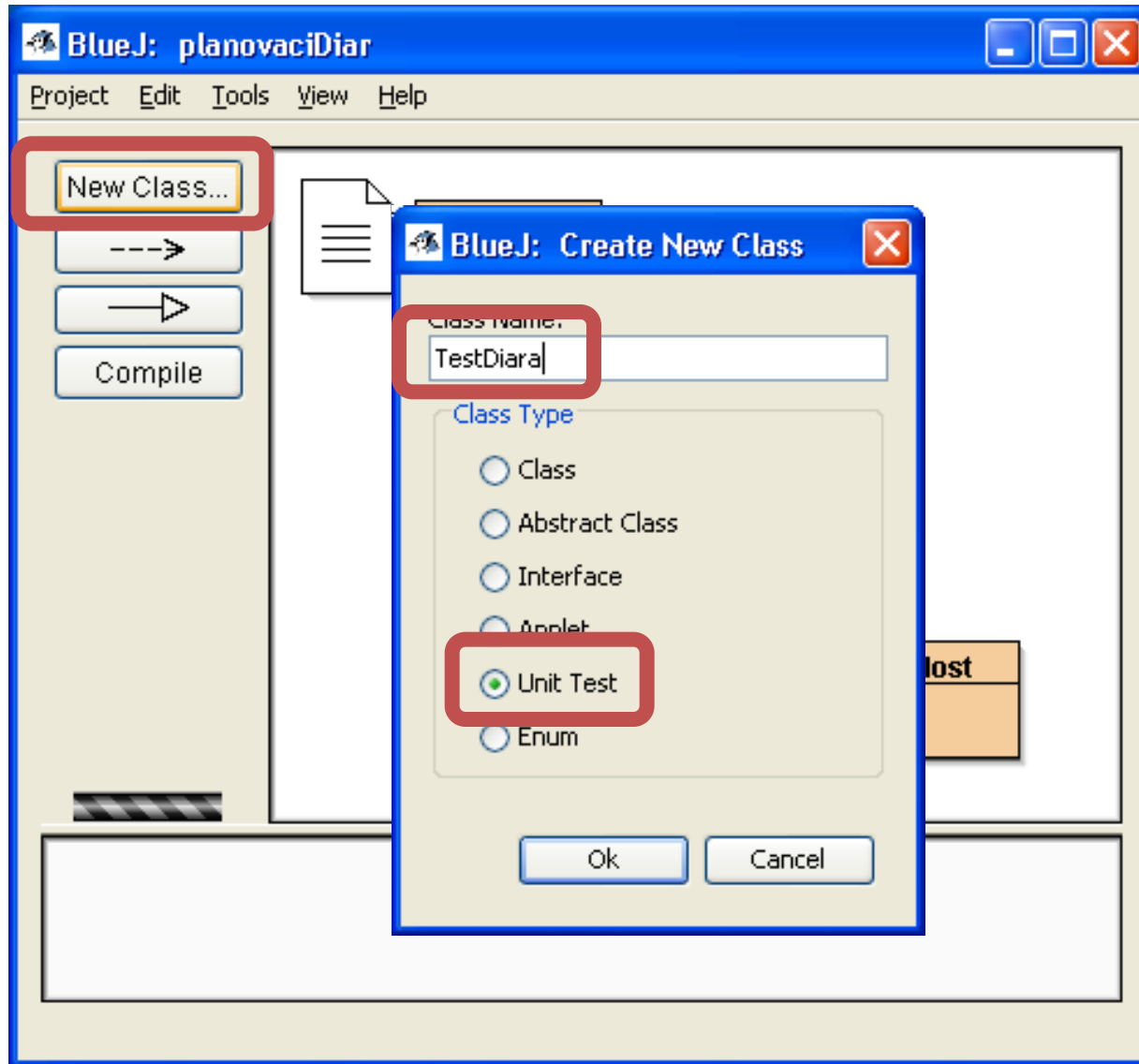
```
}
```

# Unit testy v prostředí BlueJ

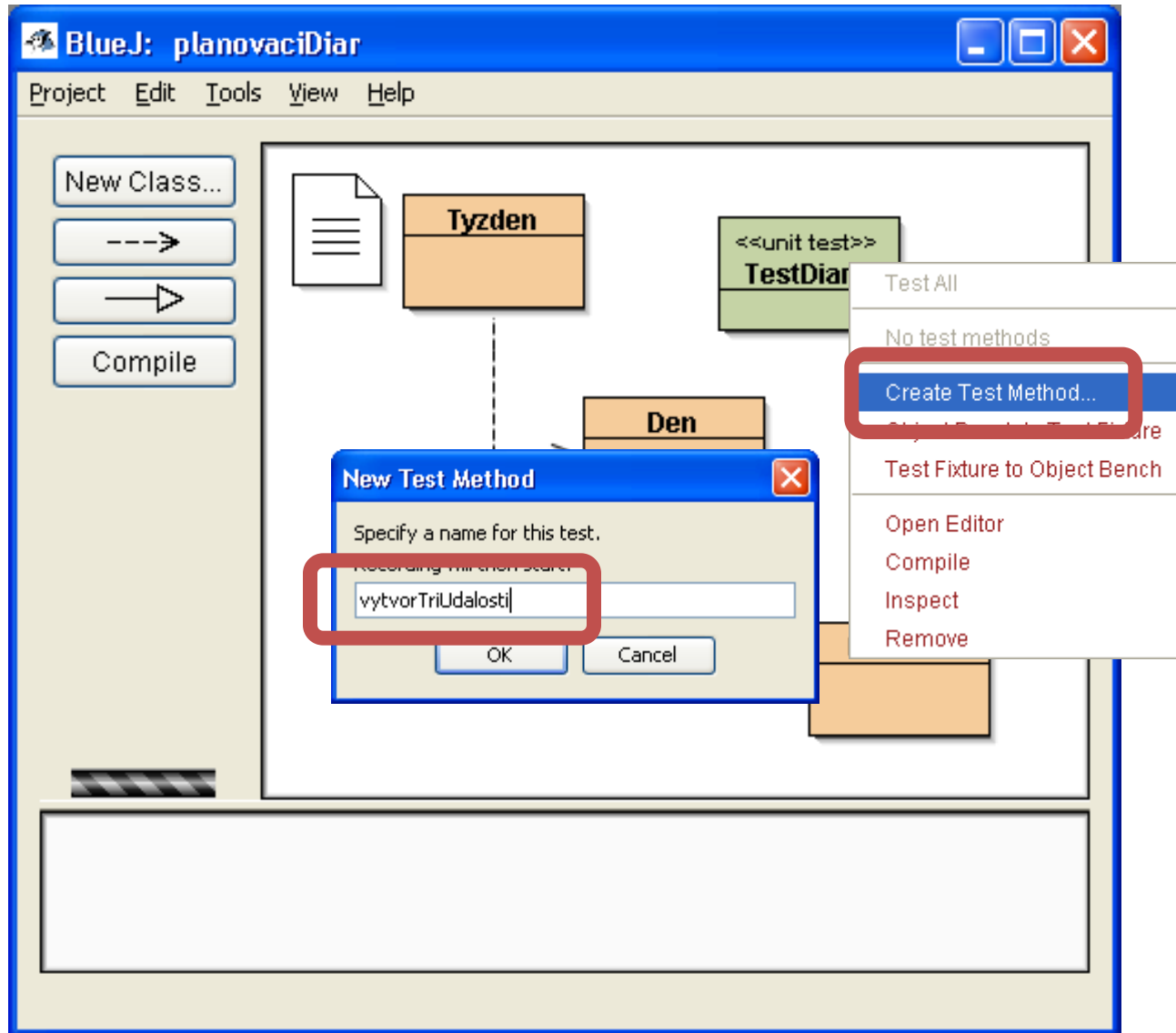
- využíva knižnicu JUnit
- „klikacie“ vytváranie testov
- zaznamenávanie činnosti testera
- dopĺňanie očakávaného parametra assertEquals
- záznam – telo testovacej metódy
- možnosť ukladania aktuálnych objektov v prostredí BlueJ ako Fixtures



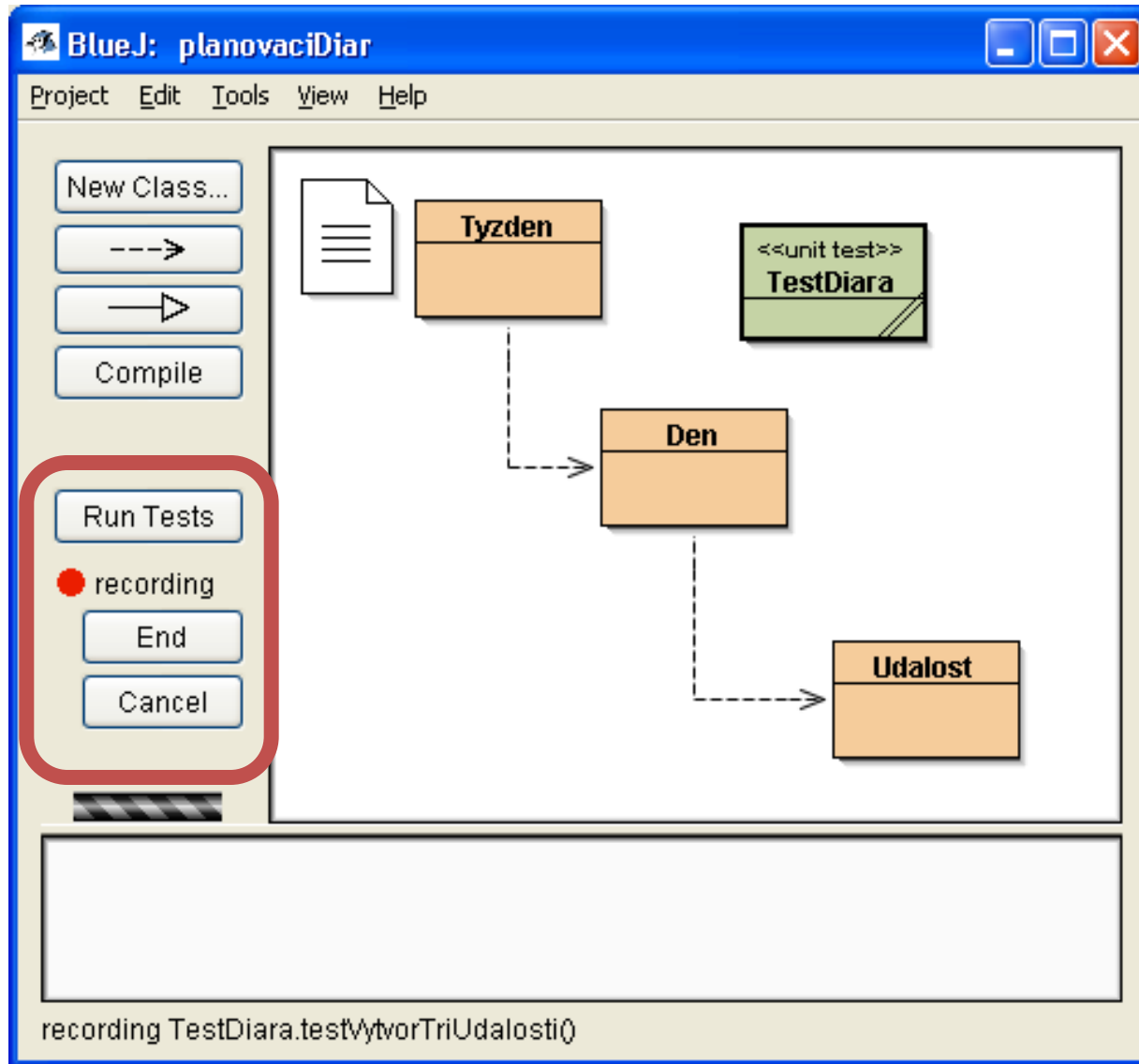
# Unit testy v prostředí BlueJ<sub>(1)</sub>



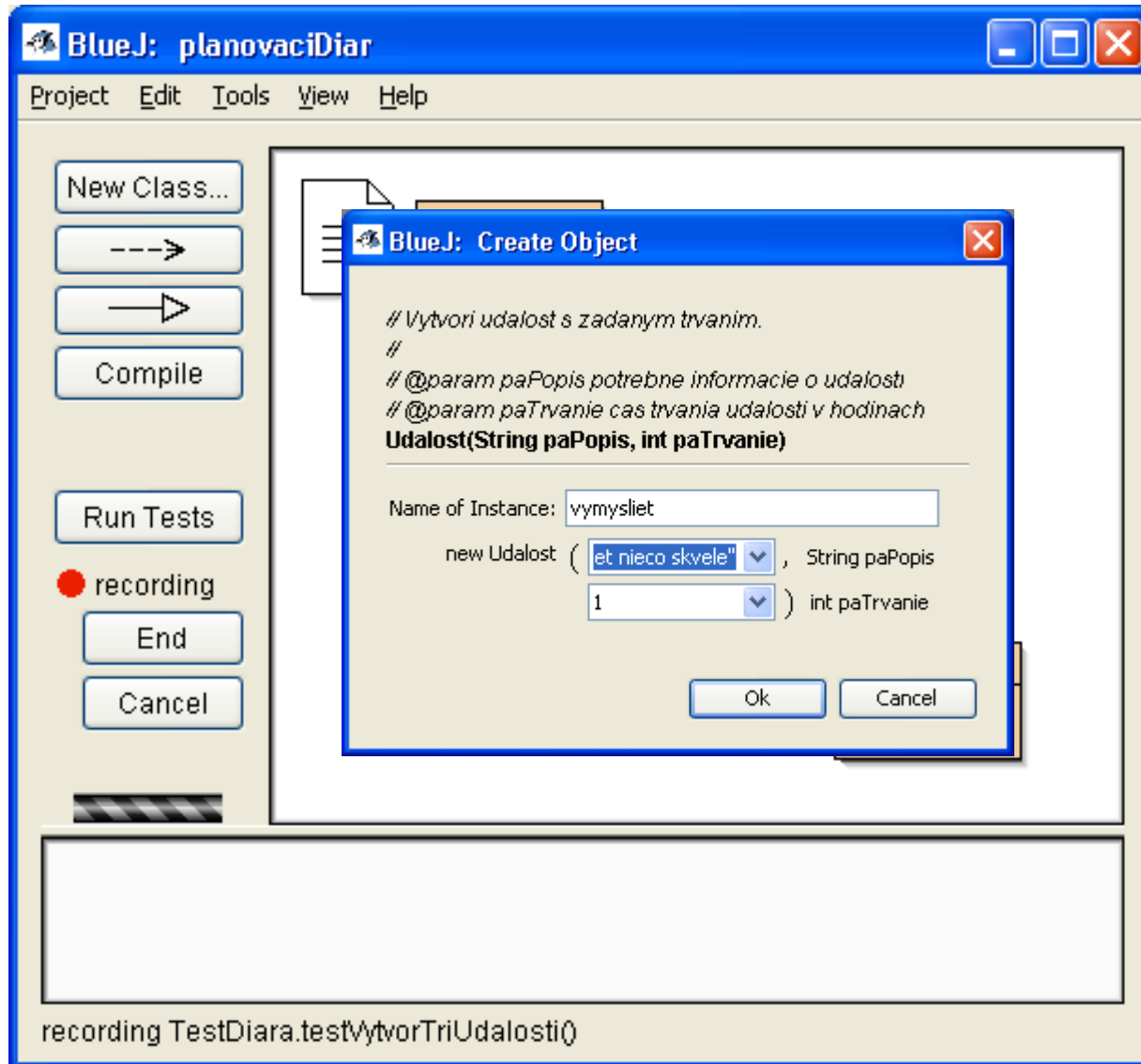
# Unit testy v prostředí BlueJ<sub>(2)</sub>



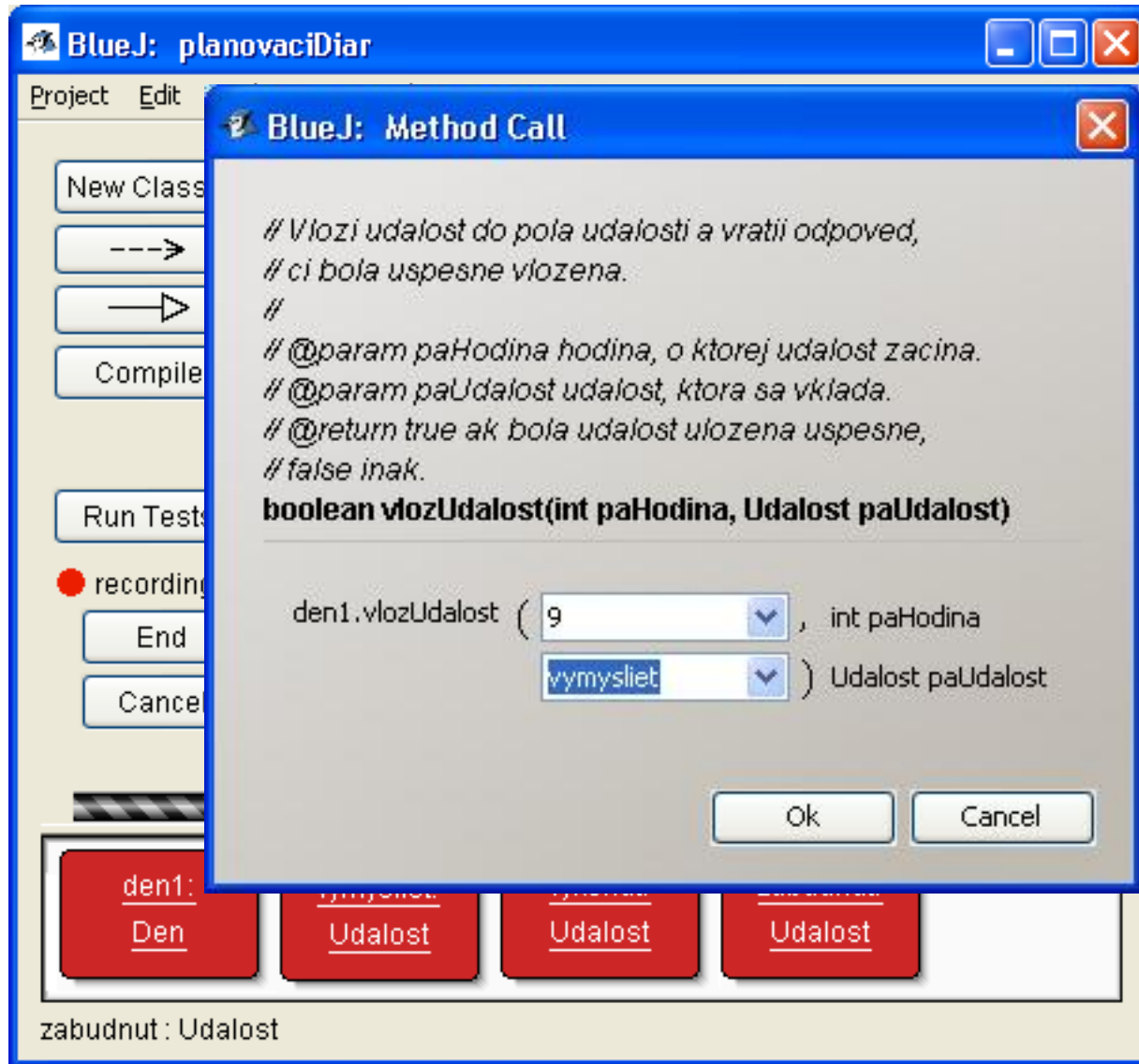
# Unit testy v prostředí BlueJ<sub>(3)</sub>



# Unit testy v prostredí BlueJ<sub>(4)</sub>



# Unit testy v prostredí BlueJ<sub>(5)</sub>



# Unit testy v prostředí BlueJ<sub>(6)</sub>

BlueJ: planovaciDiar

Project Edit Tools View Help

New Class...  
--->  
->  
Compile

Run Tests

● recording  
End  
Cancel

BlueJ: Method Result

```
// Make an appointment.  
// @param time The hour at which the appointment starts.  
// @param appointment The appointment to be made.  
// @return true if the appointment was successful,  
// false otherwise.  
boolean vlozUdalost(int paHodina, Udalost paUdalost)
```

den1.vlozUdalost(9, vykonaj) Inspect  
returned: Get  
boolean true

Assert that:  
result is equal to true

Close

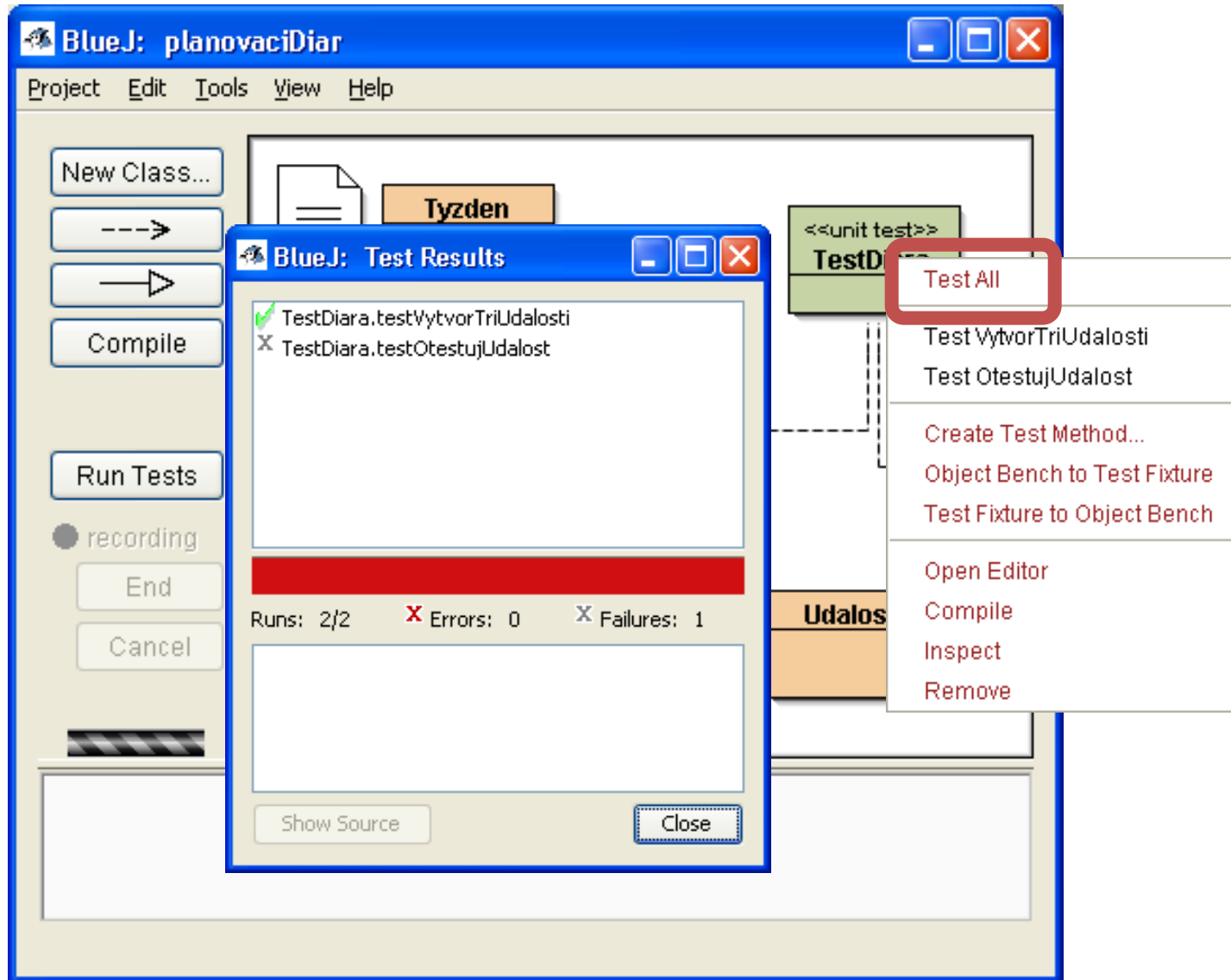
den1: Den  
vymysliet: Udalost  
vykonat: Udalost  
zabudnut: Udalost

zabudnut: Udalost

# Unit testy v prostředí BlueJ<sub>(7)</sub>

The screenshot shows the BlueJ IDE window titled "BlueJ: planovaciDiar". The interface includes a menu bar (Project, Edit, Tools, View, Help) and a toolbar with buttons for "New Class...", "Compile", "Run Tests", "End", and "Cancel". A "recording" indicator is visible above the "End" button, which is highlighted with a red rectangle. The main workspace displays a class diagram with three classes: "Tyzden", "Den", and "Udalost", all represented as orange boxes. A green box labeled "<<unit test>> TestDiara" is also present. Dashed arrows indicate dependencies: "Tyzden" depends on "Den", "Den" depends on "Udalost", and "TestDiara" depends on both "Den" and "Udalost". At the bottom, there is a red button bar with four buttons: "den1: Den", "vymysliet: Udalost", "vykonat: Udalost", and "zabudnut: Udalost". The text "zabudnut: Udalost" is also displayed below the button bar.

# Unit testy v prostředí BlueJ<sub>(8)</sub>





# 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
- 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

Compile

Undo

Cut

Copy

Paste

Find...

Find Next

Close

Source Code



```
20
21     * @param paPopis potrebne informacie o udalosti
22     * @param paTrvanie cas trvania udalosti v hodinach
23     */
24     public Udalost(String paPopis, int paTrvanie)
25     {
26         this.aPopis = paPopis;
27         this.aTrvanie = 1;
28     }
29
30     /**
31     * @return informacie o udalosti
32     */
33     public String doPopisu()
```

STOP

*saved*

New Class



Compile

## BlueJ: Create Object

```
// Vytvori udalost s zadanim trvanim.
```

```
//
```

```
// @param paPopis potrebne informacie o udalosti
```

```
// @param paTrvanie cas trvania udalosti v hodinach
```

```
Udalost(String paPopis, int paTrvanie)
```

Name of Instance: new Udalost (  , String paPopis  
 ) int paTrvanie

Ok

Cancel

den1:

Den



```
Udalost
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
20
21 * @param paPopis potrebne informacie o udalosti
22 * @param paTrvanie cas trvania udalosti v hodinach
23 */
24 public Udalost(String paPopis, int paTrvanie)
25 {
26     this.aPopis = paPopis;
27     this.aTrvanie = 1;
28 }
29 /**
30 * @return informacie o udalosti
31 */
32 public String doPopisu()
33 {
34     return aPopis;
35 }
36 }
Thread "main" stopped at breakpoint.
```

BlueJ: Debugger

Options

Threads  
● main (at breakpoint)

Call Sequence  
Udalost, <init>

class attributes

Instance variables  
String aPopis = null  
int aTrvanie = 0

Local variables  
String paPopis = "cvicenie"  
int paTrvanie = 2

Halt Step Step Into Continue Terminate

```
Udalost
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
20
21 * @param paPopis potrebne informacie o udalosti
22 * @param paTrvanie cas trvania udalosti v hodinach
23 */
24 public Udalost(String paPopis, int paTrvanie)
25 {
26     this.aPopis = paPopis;
27     this.aTrvanie = 1;
28 }
29
30 /**
31 * @return informacie o udalosti
32 */
33 public String doPopisu()
```

BlueJ: Debugger

Options

Threads  
● main (stopped)

Call Sequence	class attributes
Udalost, <init>	

Instance variables  
String aPopis = "cvicenie"  
int aTrvanie = 0

Local variables  
String paPopis = "cvicenie"  
int paTrvanie = 2

Halt Step Step Into Continue Terminate

```
Udalost
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
20
21 * @param paPopis potrebne informacie o udalosti
22 * @param paTrvanie cas trvania udalosti v hodinach
23 */
24 public Udalost(String paPopis, int paTrvanie)
25 {
26     this.aPopis = paPopis;
27     this.aTrvanie = 1;
28 }
29
30 /**
31 * @return informacie o udalosti
32 */
33 public String doDeta...()
```

BlueJ: Debugger

Options

Threads  
● main (stopped)

Call Sequence	class attributes
Udalost, <init>	

Instance variables  
String aPopis = "cvicenie"  
int aTrvanie = 1

Local variables  
String paPopis = "cvicenie"  
int paTrvanie = 2

Halt Step Step Into Continue Terminate

chyba

```
Udalost
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
20
21 * @param paPopis potrebne informacie o udalosti
22 * @param paTrvanie cas trvania udalosti v hodinach
23 */
24 public Udalost(String paPopis, int paTrvanie)
25 {
26     this.aPopis = paPopis;
27     this.aTrvanie = 1;
28 }
29
30 /**
31 * @return informacie o udalosti
32 */
33 public String doPopisu()
```

BlueJ: Debugger

Options

Threads

- main (finished)

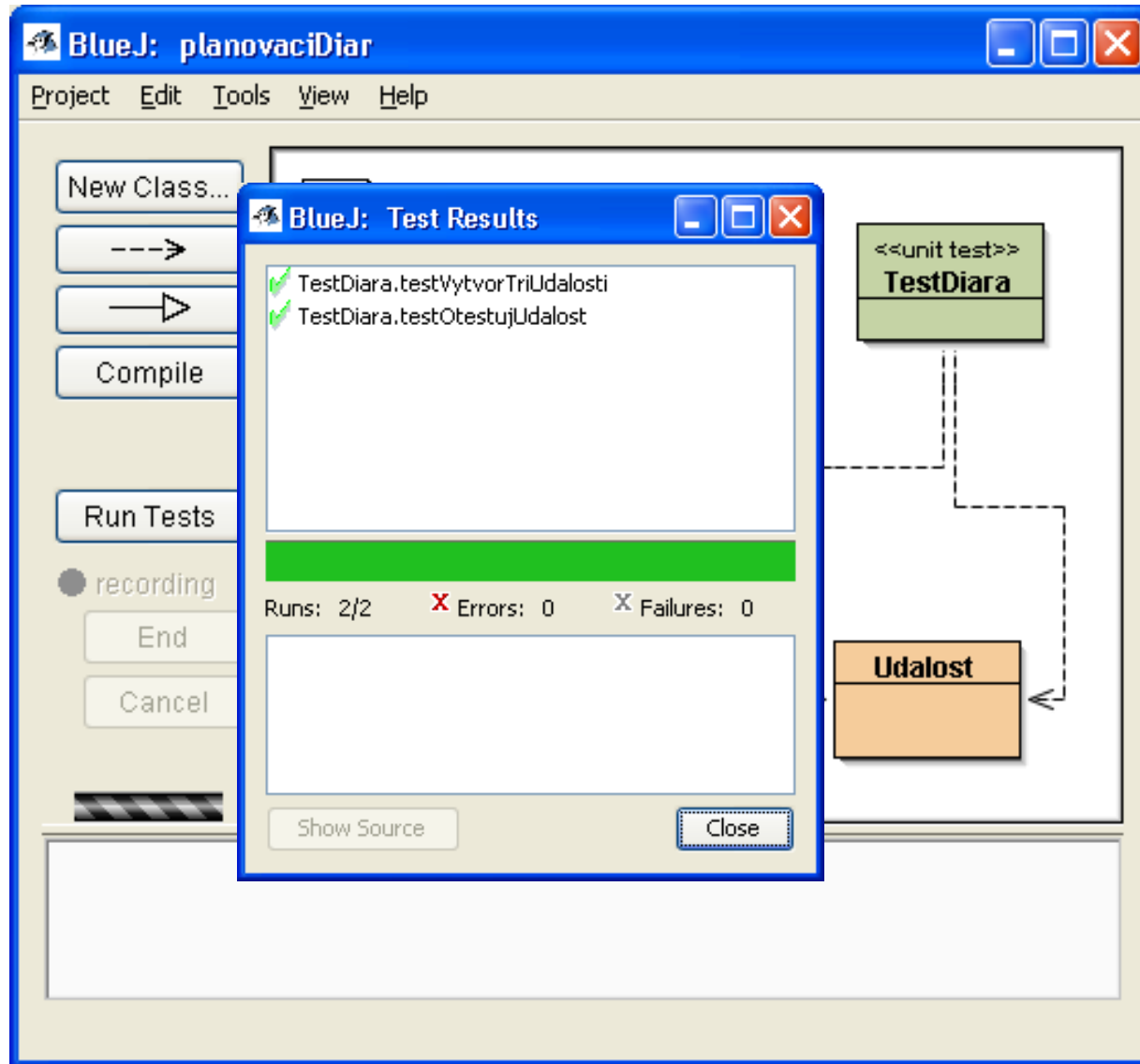
Thread is running.  
Threads must be stopped to view details.

Halt Step Step Into Continue Terminate

# Oprava chyby

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

# Výsledek po oprave



# Metóda inštancie triedy Programator ☺

```
public Program vytvorProgram(Zadanie paZadanie)
{
    Program program = this.napisProgram(paZadanie);
    Test test = this.napisTestPre(program);
    Chyba chyba = test.dajChybu(program);
    while (chyba != null) {
        InfoOChybe info = this.lad(program, chyba);
        this.odstranChybu(program, info);
        chyba = test.dajChybu(program);
    }
    return program;
}
```

# Písanie udržovateľného kódu

- čitateľnosť kódu
- konvencie
- dokumentačné komentáre
- komentáre v zložitejších miestach algoritmu
- samopopisné identifikátory
- súdržnosť (cohesion) – max.
- implementačná závislosť (coupling) – min.



# Vďaka za pozornosť