1

Mýliť sa je ľudské

KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Cieľ prednášky

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

- príklad: projekt plánovací diár

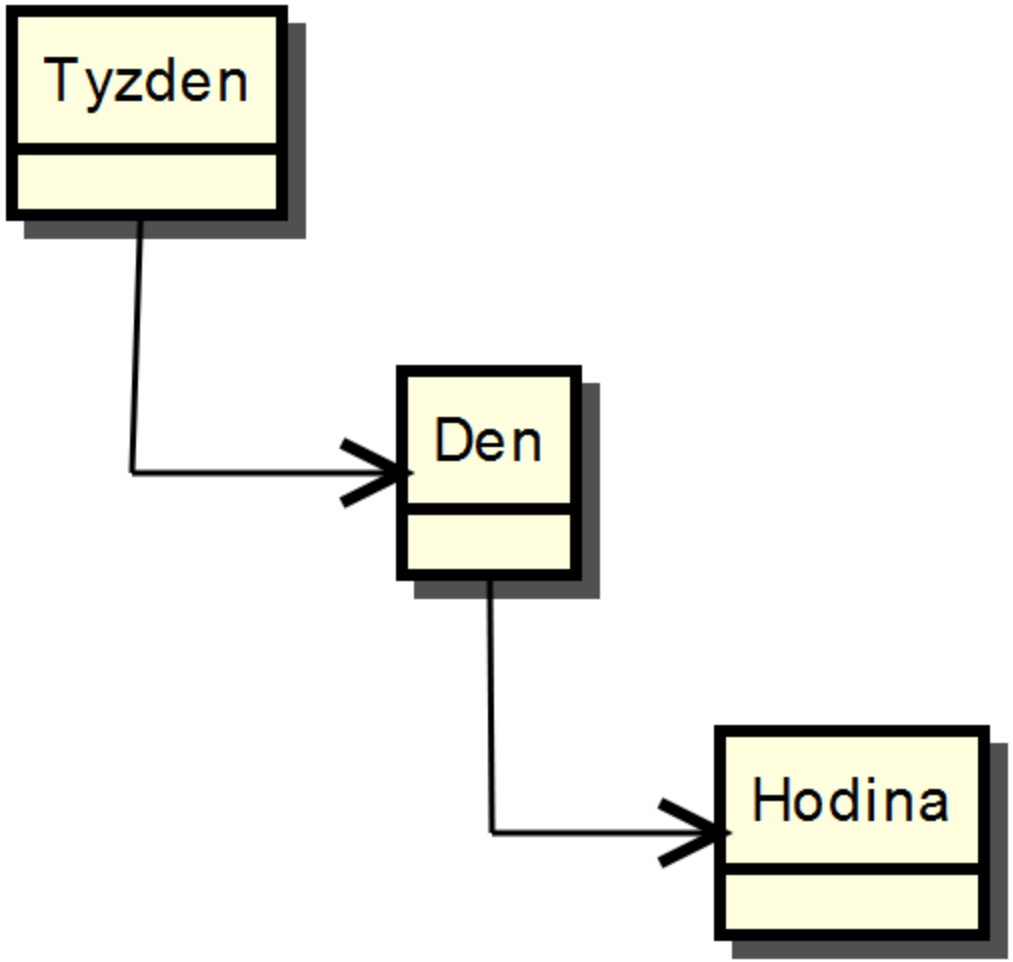
Projekt planovaciDiar

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

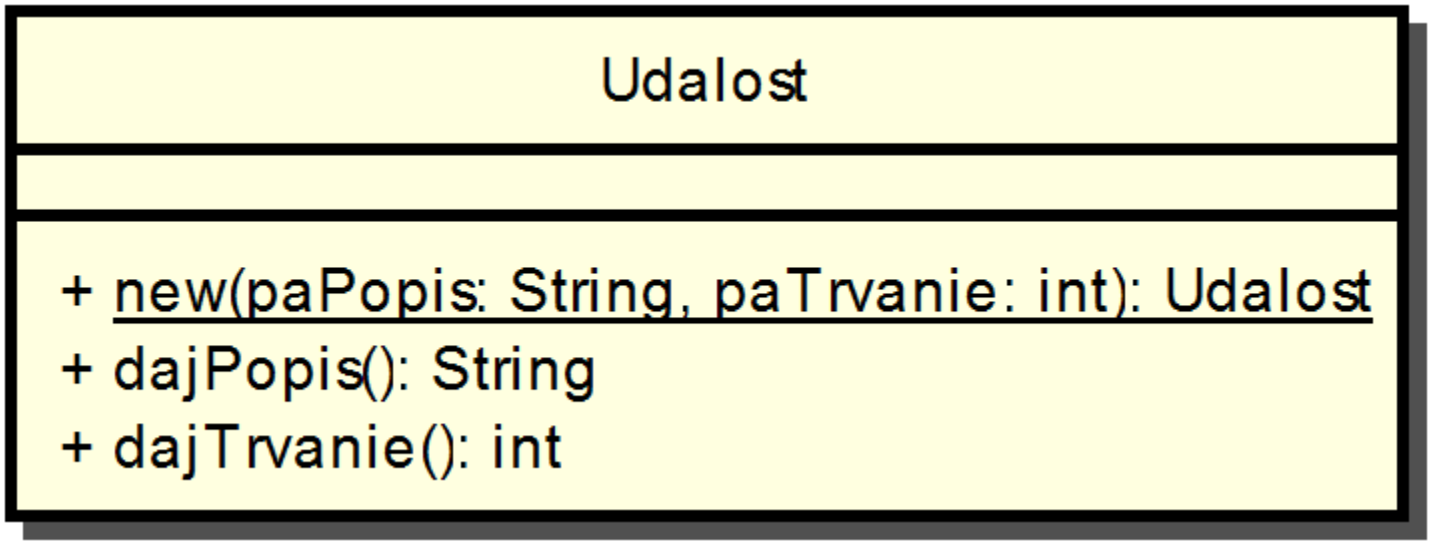
Projekt planovaciDiar – zadanie

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

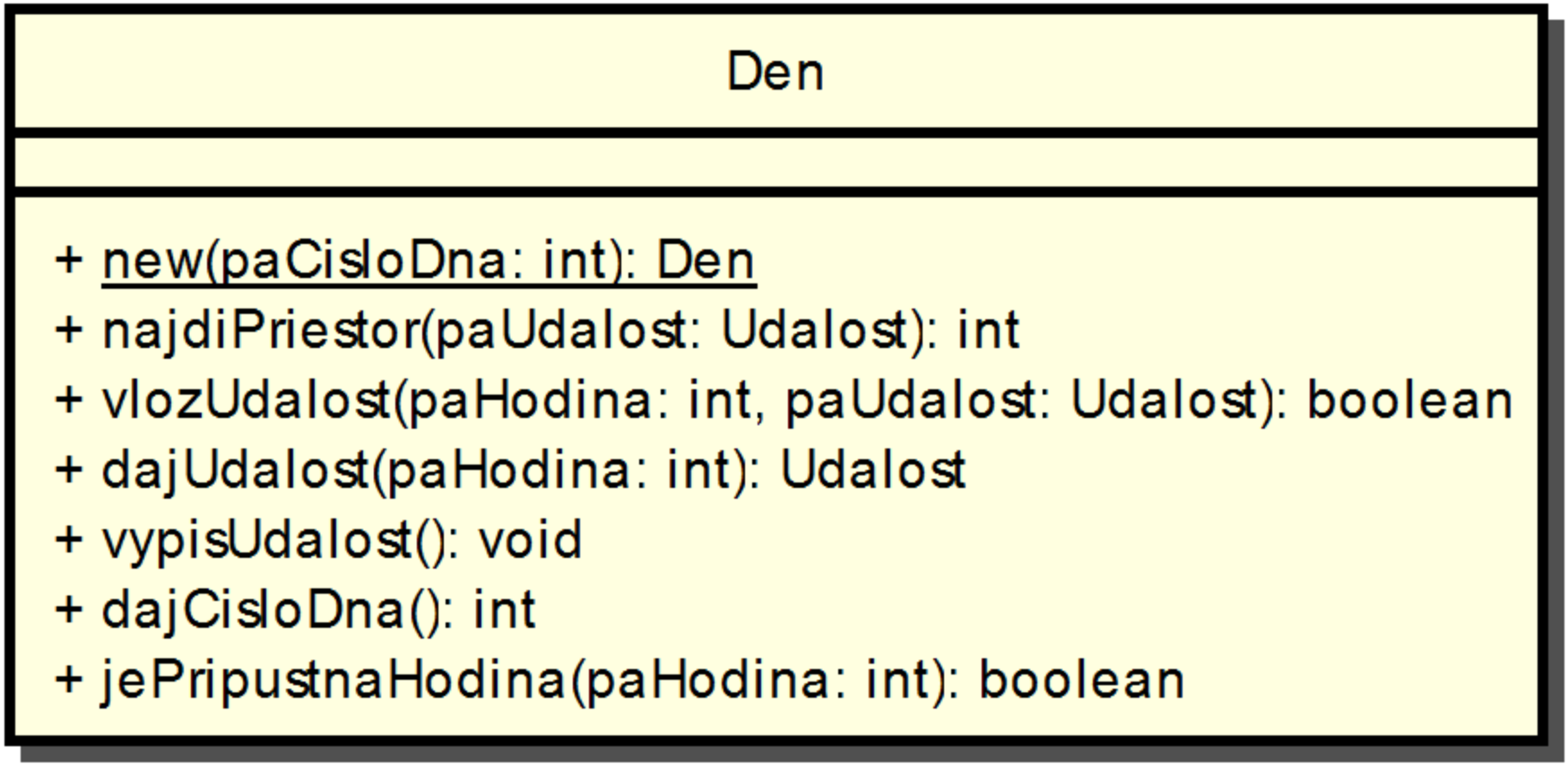
Projekt planovaciDiar – model



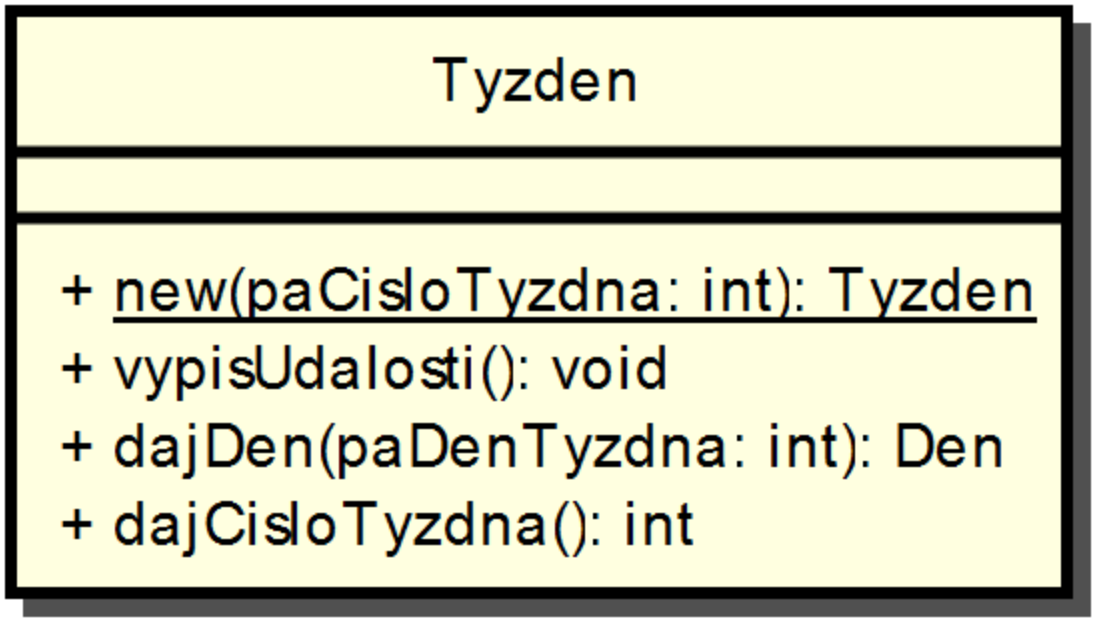
Projekt planovaciDiar – trieda Udalost



Projekt planovaciDiar – trieda Den



Projekt planovaciDiar – trieda Tyzden



Typy chýb

- syntaktické chyby
- behové chyby
- logické chyby

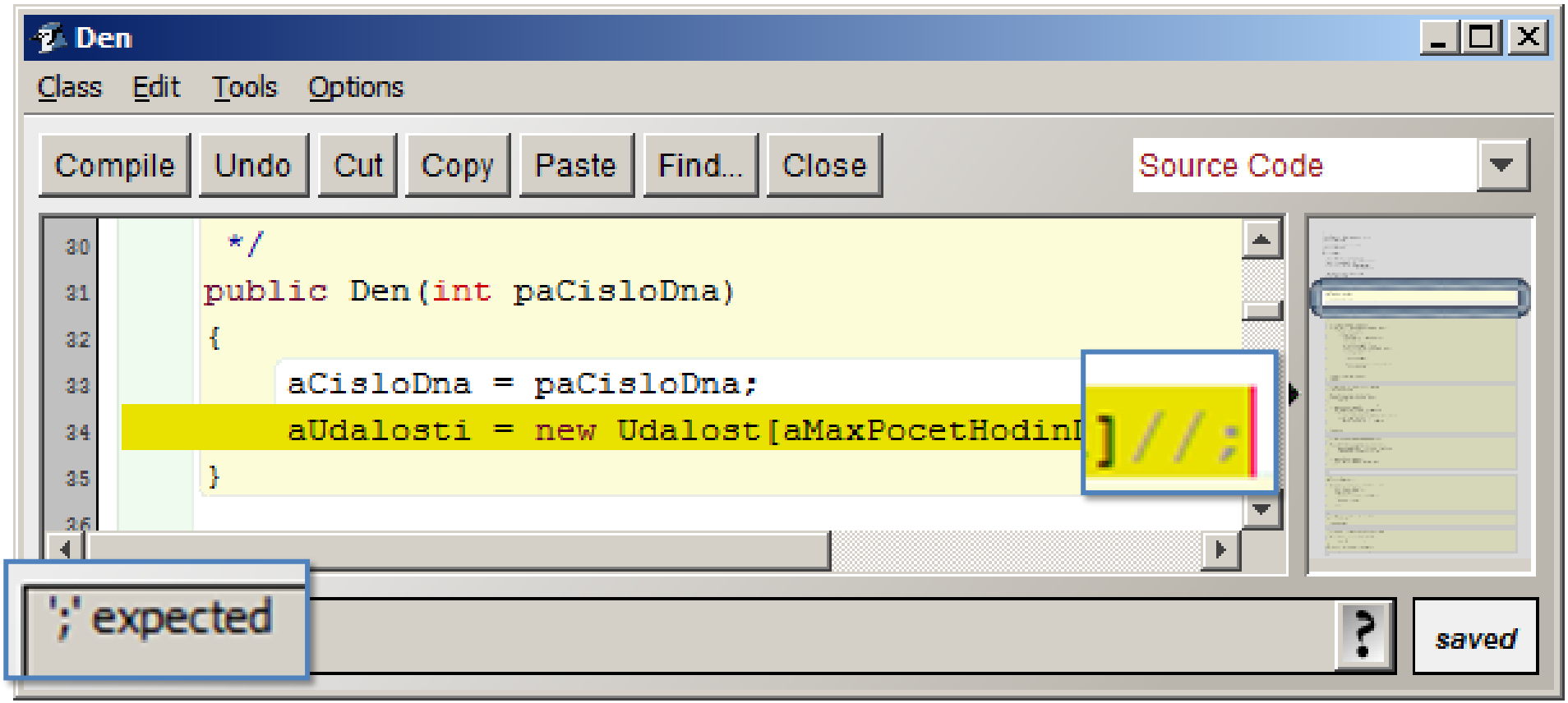
Syntaktické chyby

- 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

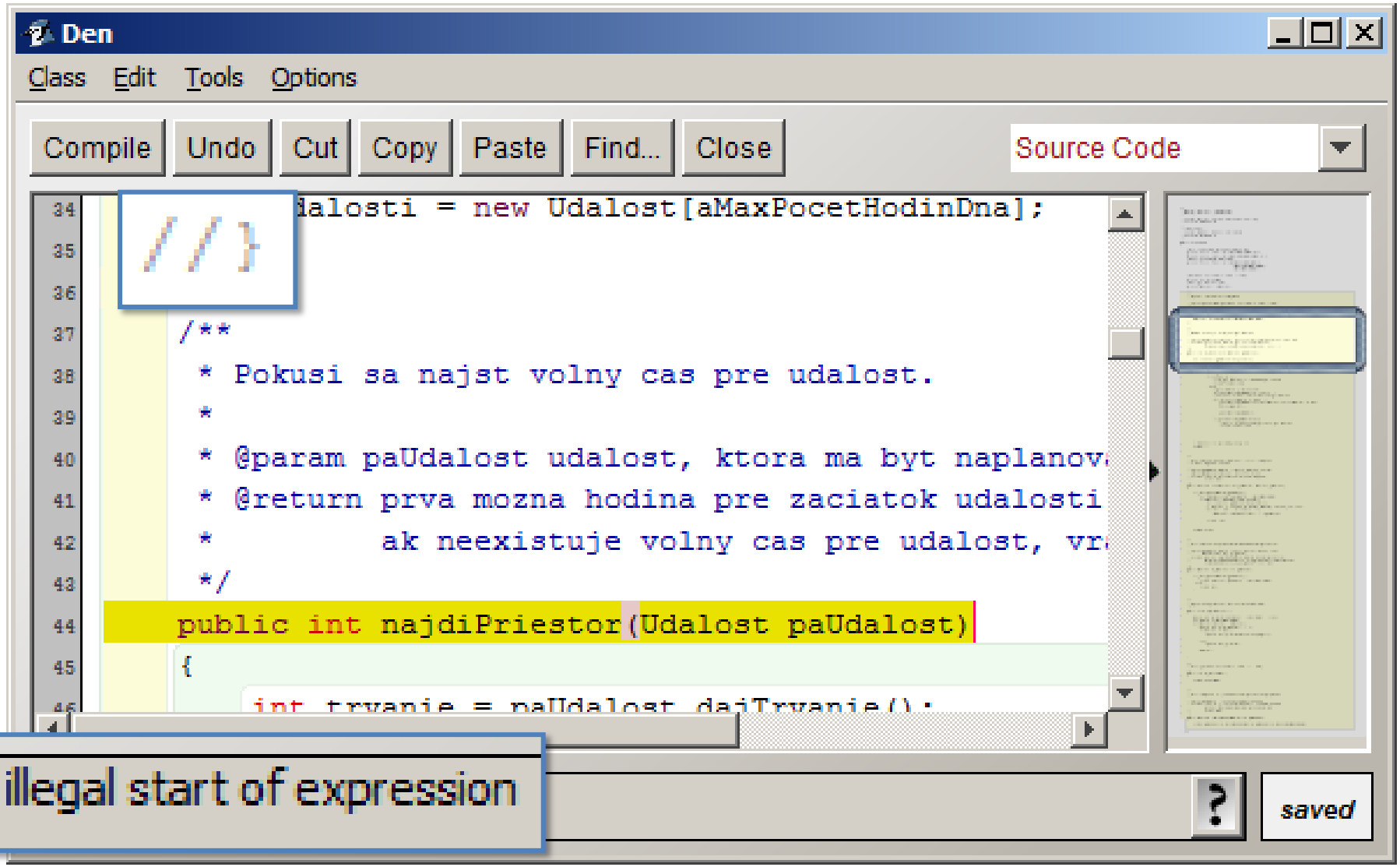
- !čítať texty chybových hlásení!

Syntaktické chyby – príklad(1)



KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

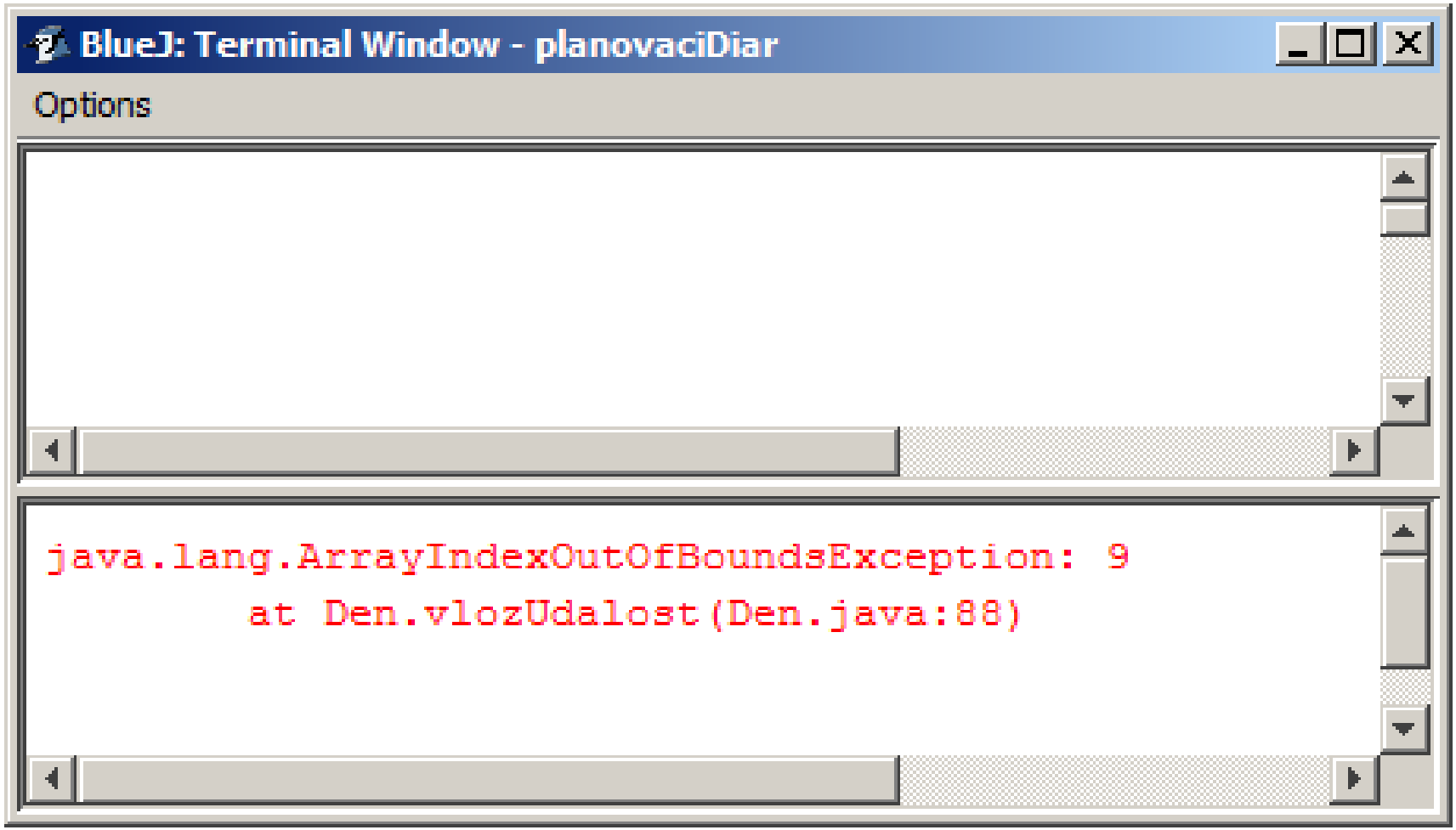
Syntaktické chyby – príklad(2)



Behové chyby

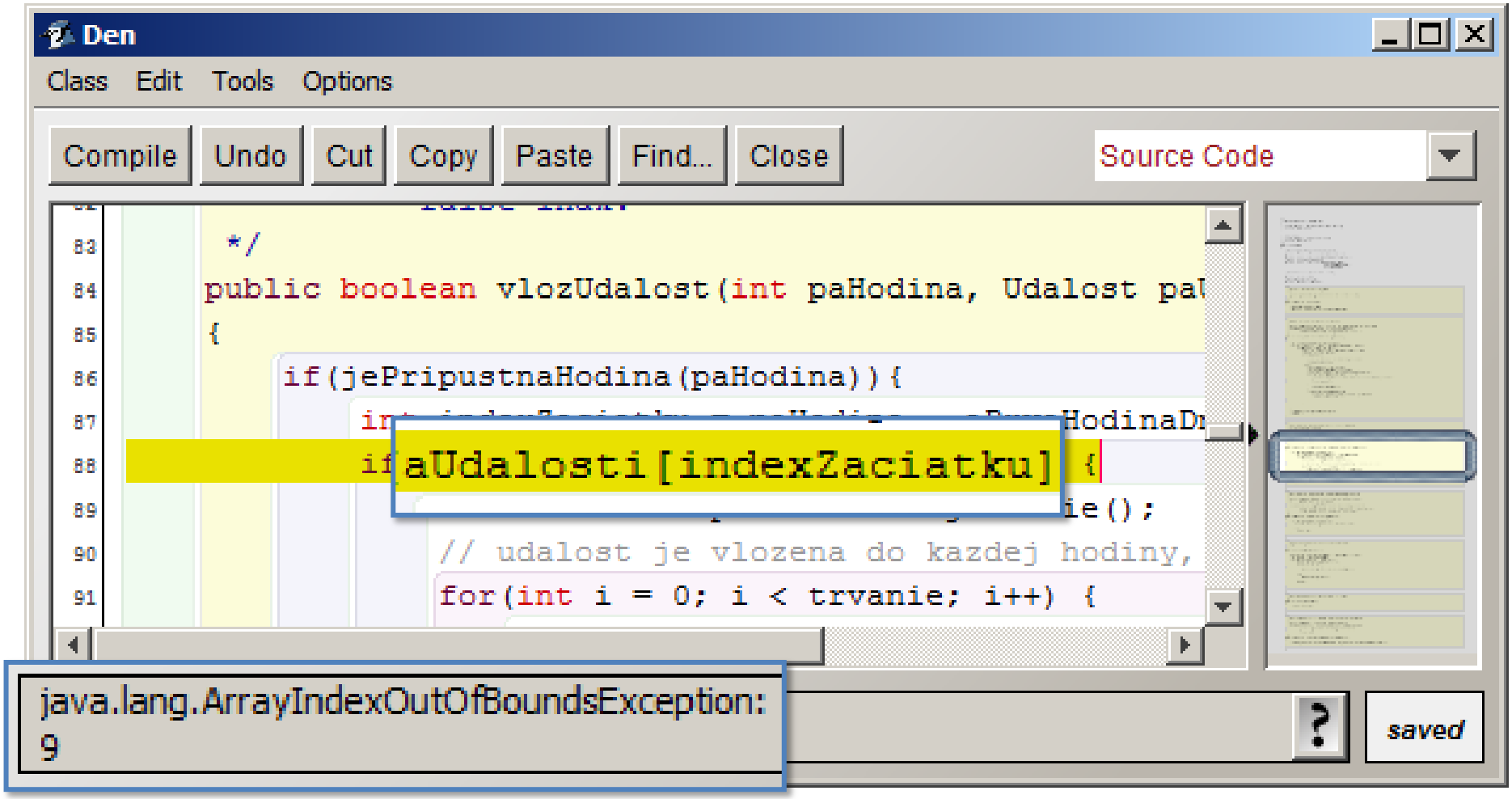
- zistí a „hlási“ procesor pri vykonávaní programu
- hlási = program „havaruje“
- procesor nemôže vykonať požadovaný príkaz
- delenie nulou, správa neexistujúcemu objektu...,
- zákernosť behových chýb
  - nemusia sa prejaviť pri každom spustení programu
  - „zavlečená“ – skutočná chyba je niekde skôr

Behové chyby – príklad(1)



KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

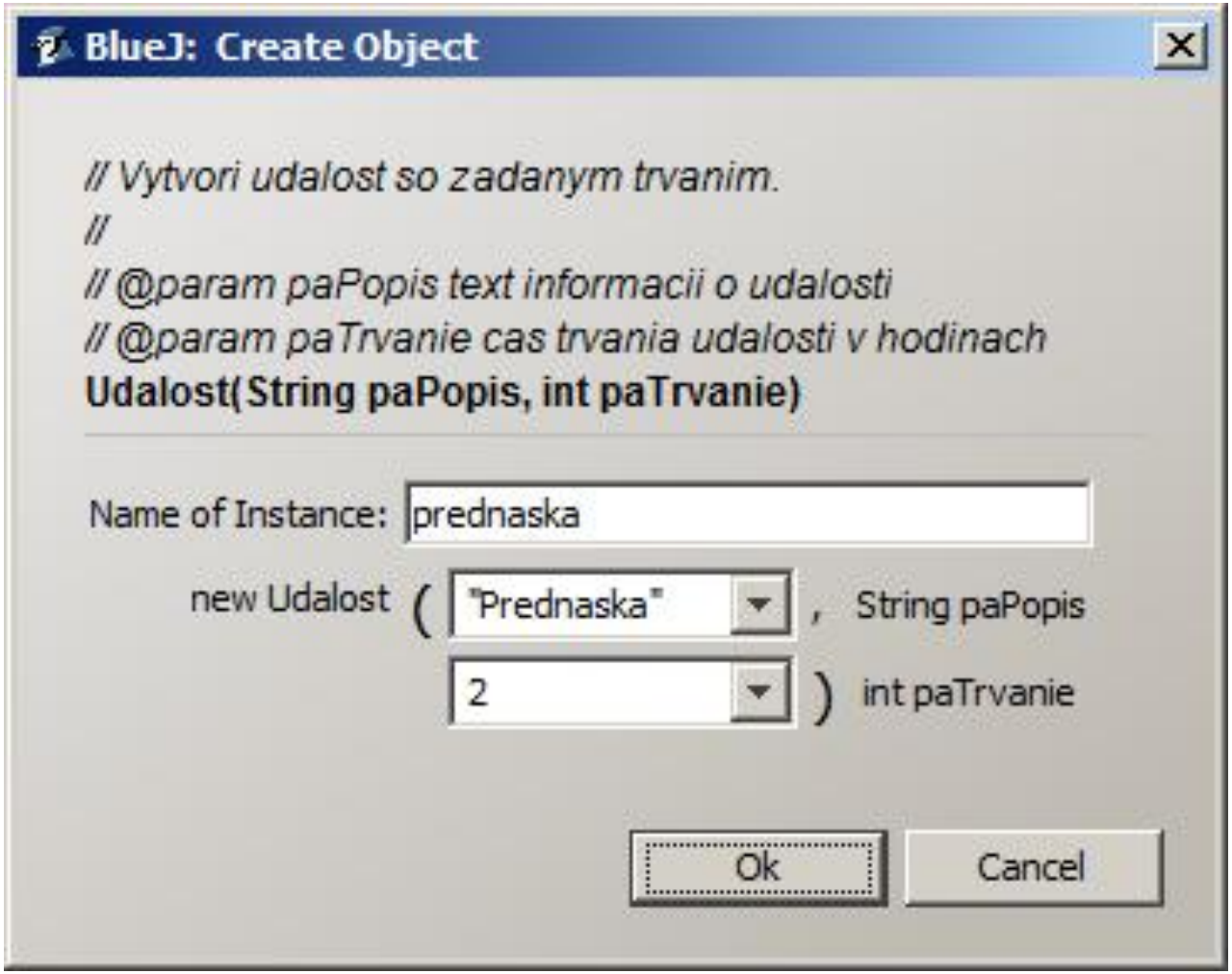
Behové chyby – príklad(2)



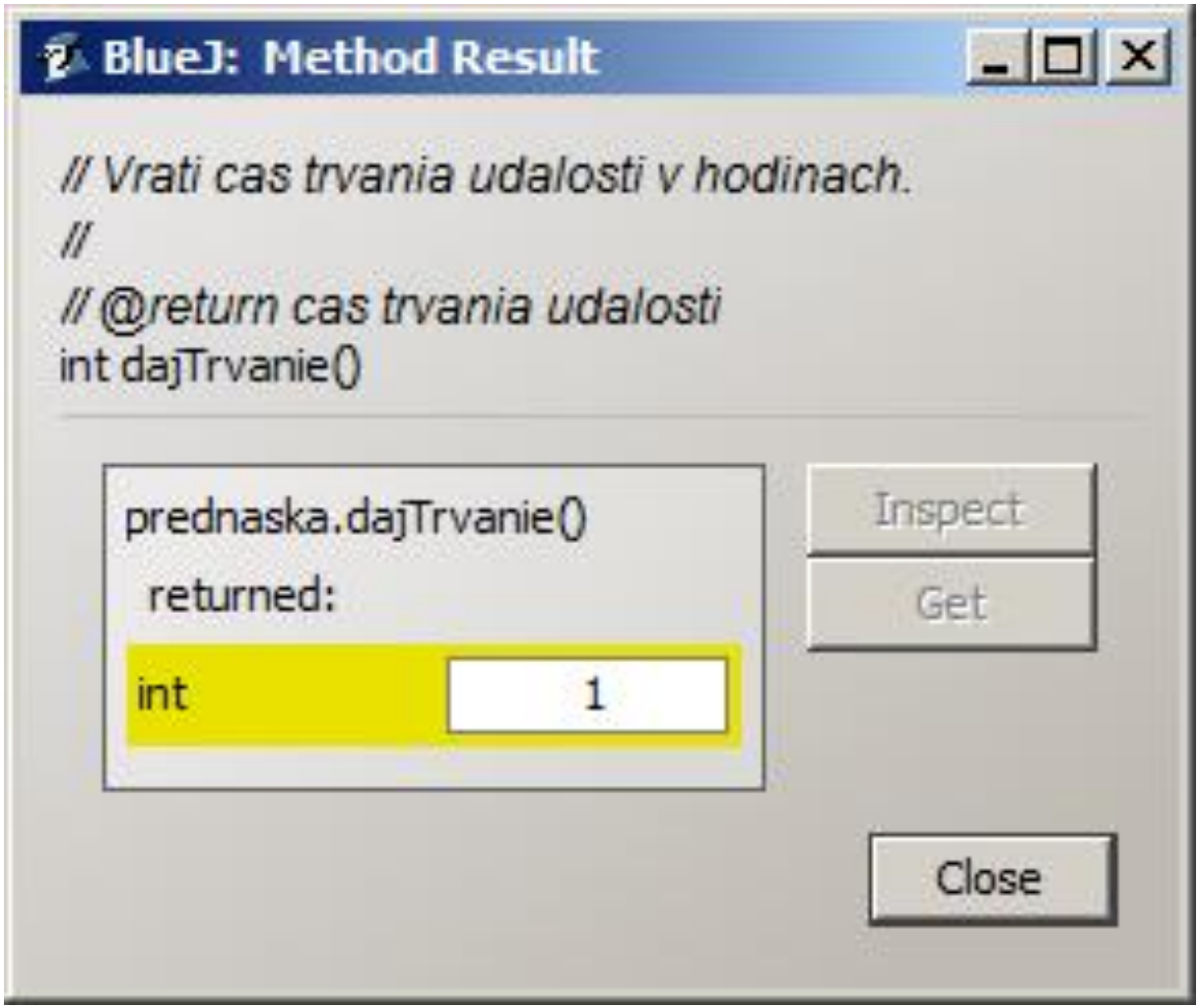
Logické chyby

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

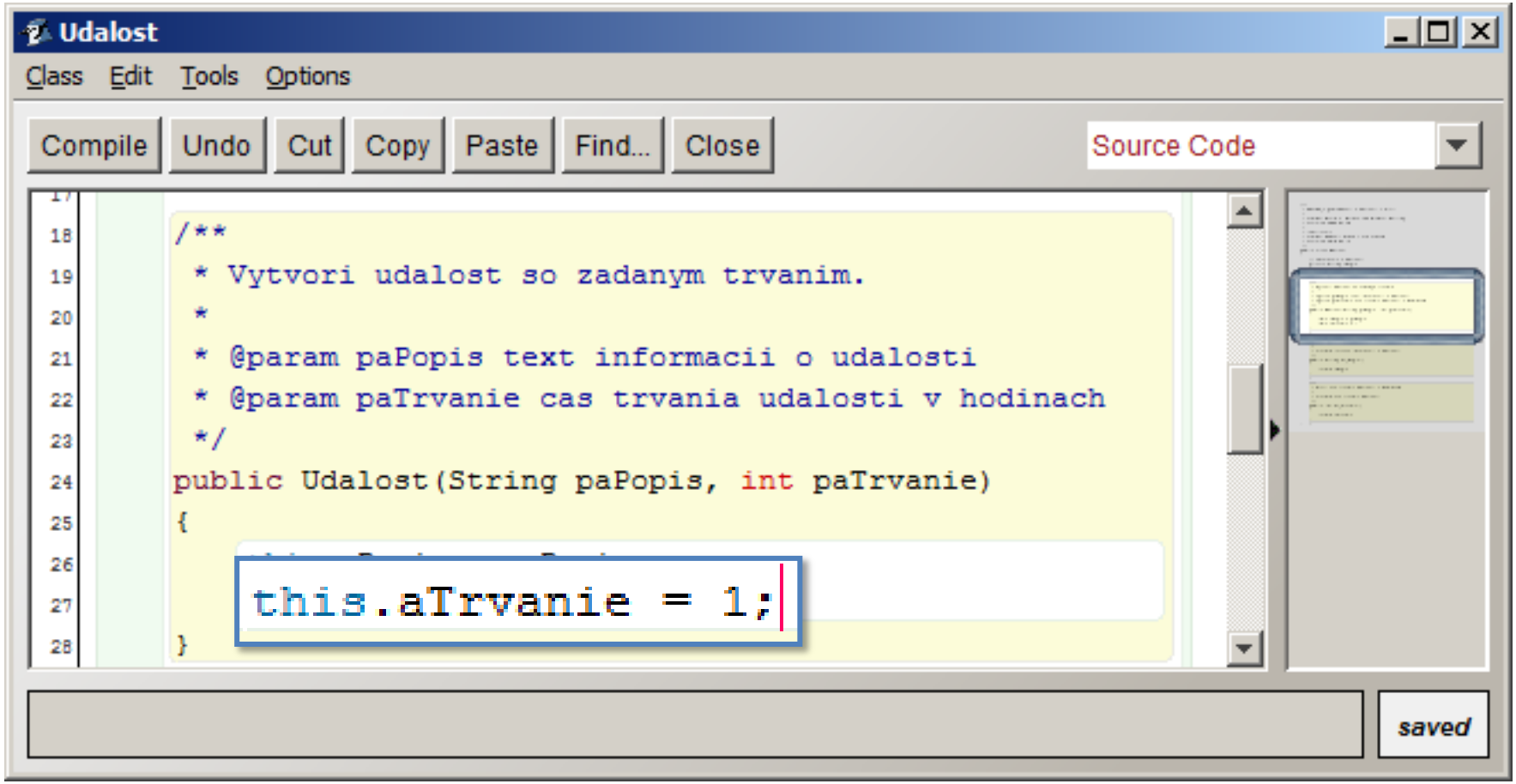
Logické chyby – príklad(1)



Logické chyby – príklad(2)



Logické chyby – príklad(3)



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(1)

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

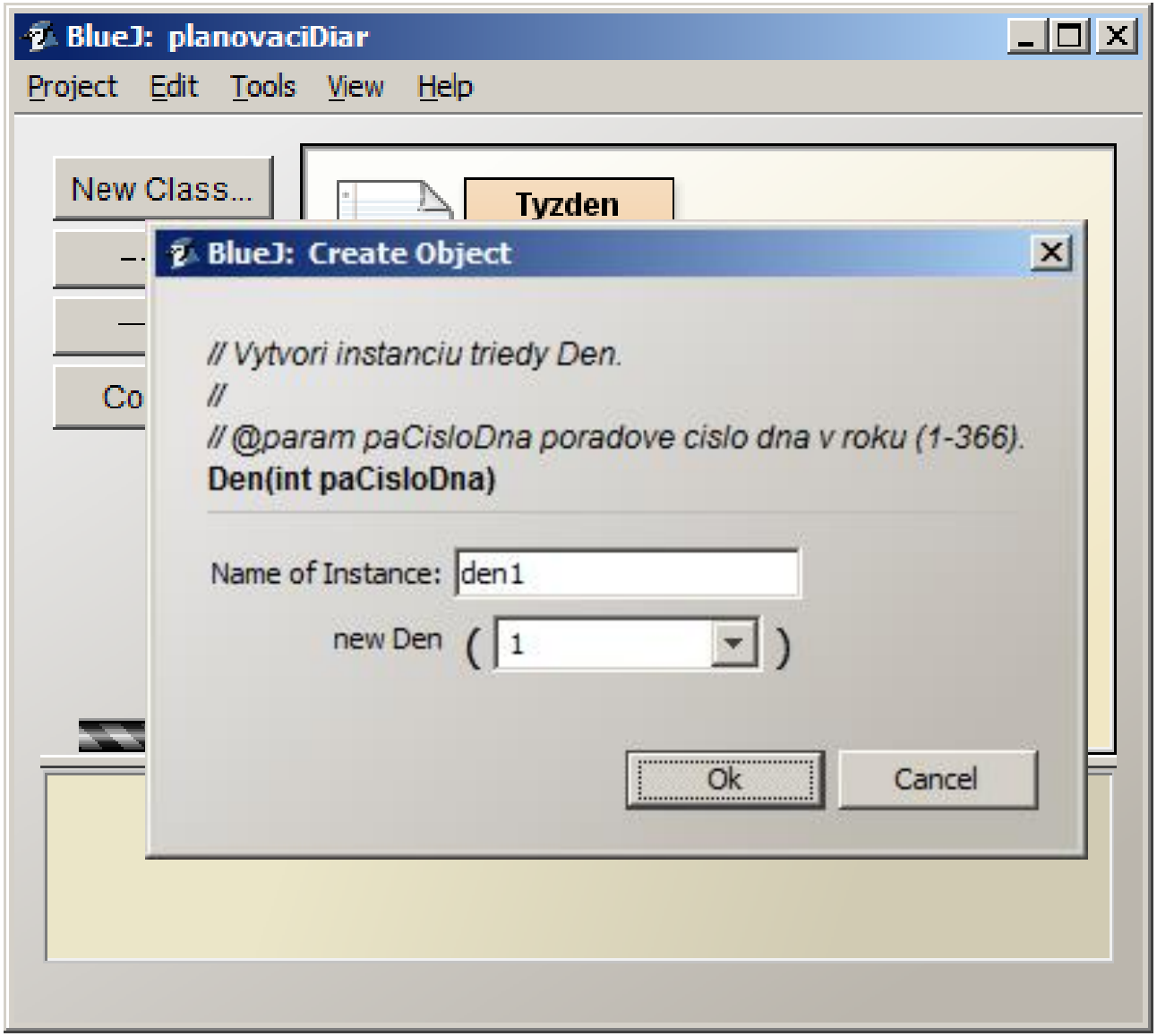
Manuálne testovanie jednotiek(2)

- 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

Využitie funkcie objekt inspector

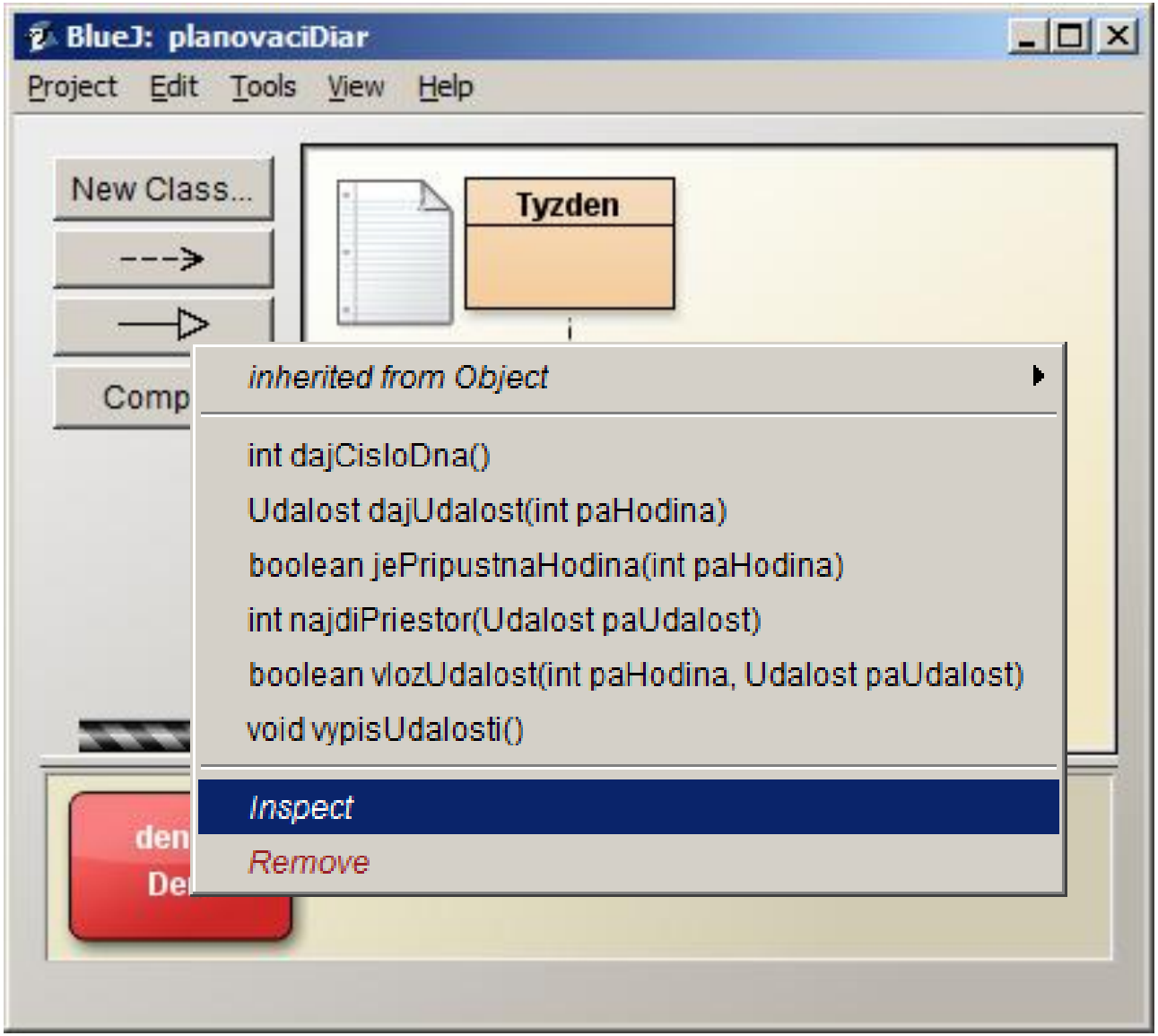
- sledovanie reakcie objektu na správu
- object inspector ostáva otvorený
- kontrola stavu atribútov
  - trieda
  - inštancia

Object inspector – príklad(1)



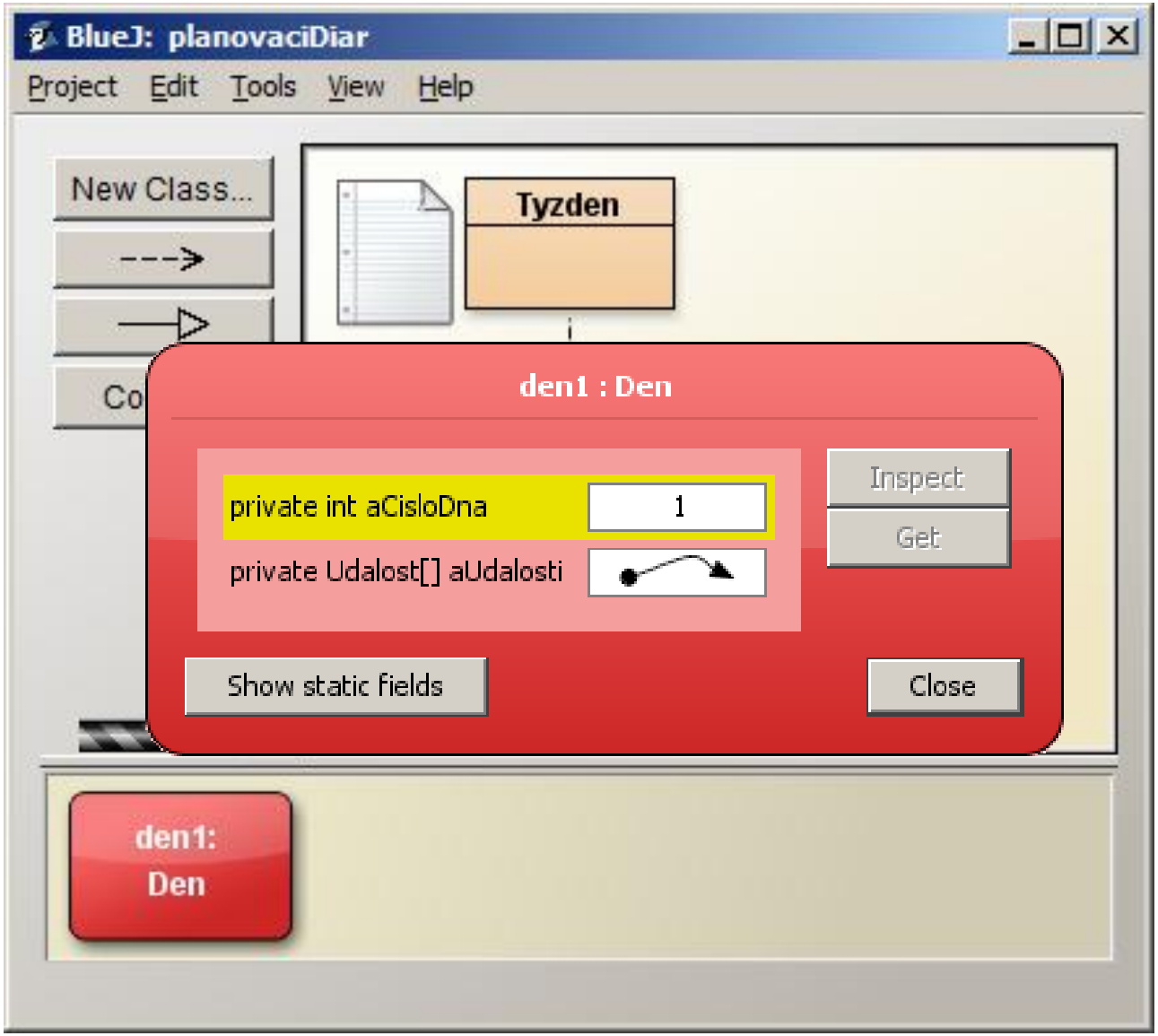
KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Object inspector – príklad(2)



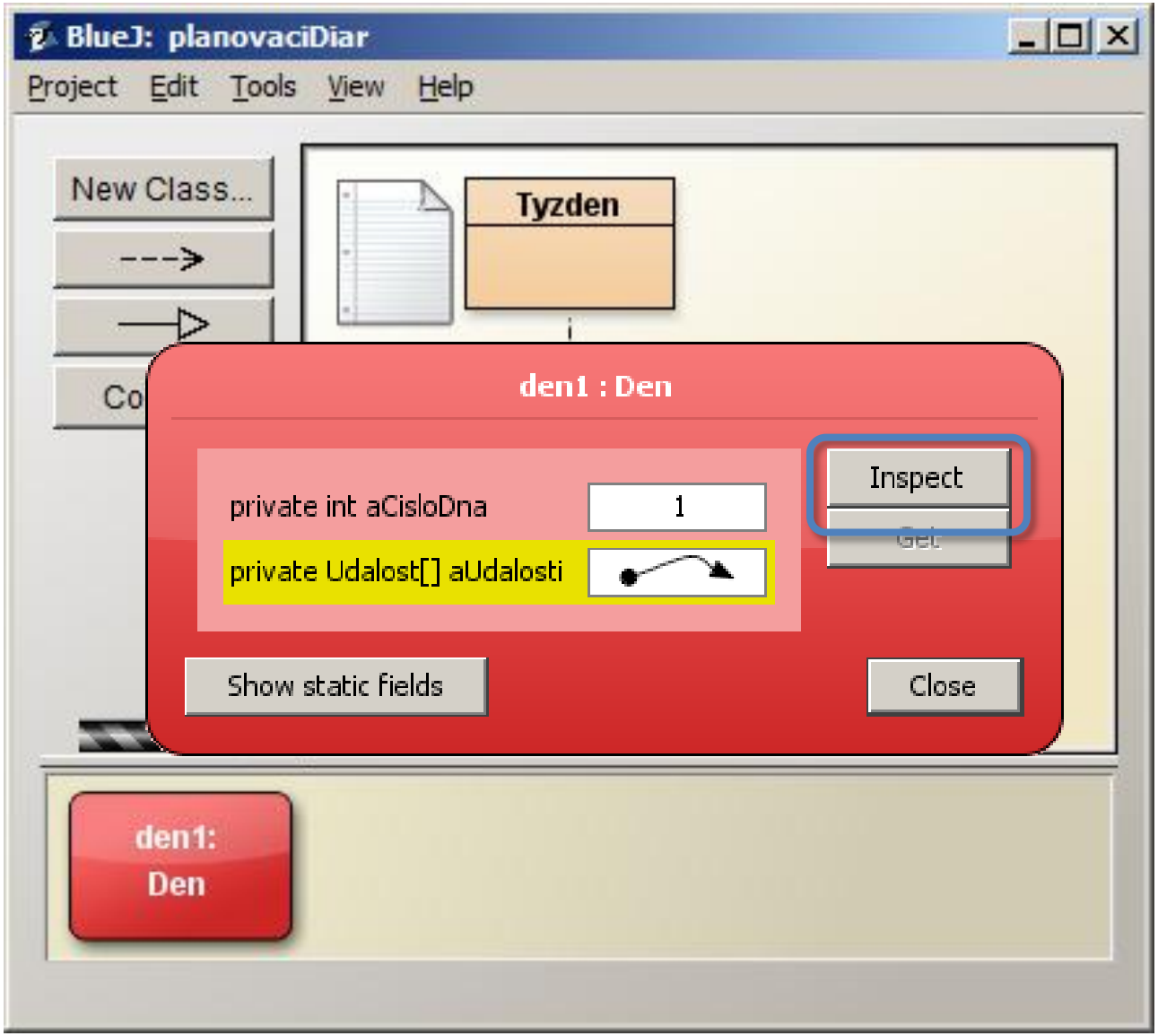
KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Object inspector – príklad(3)

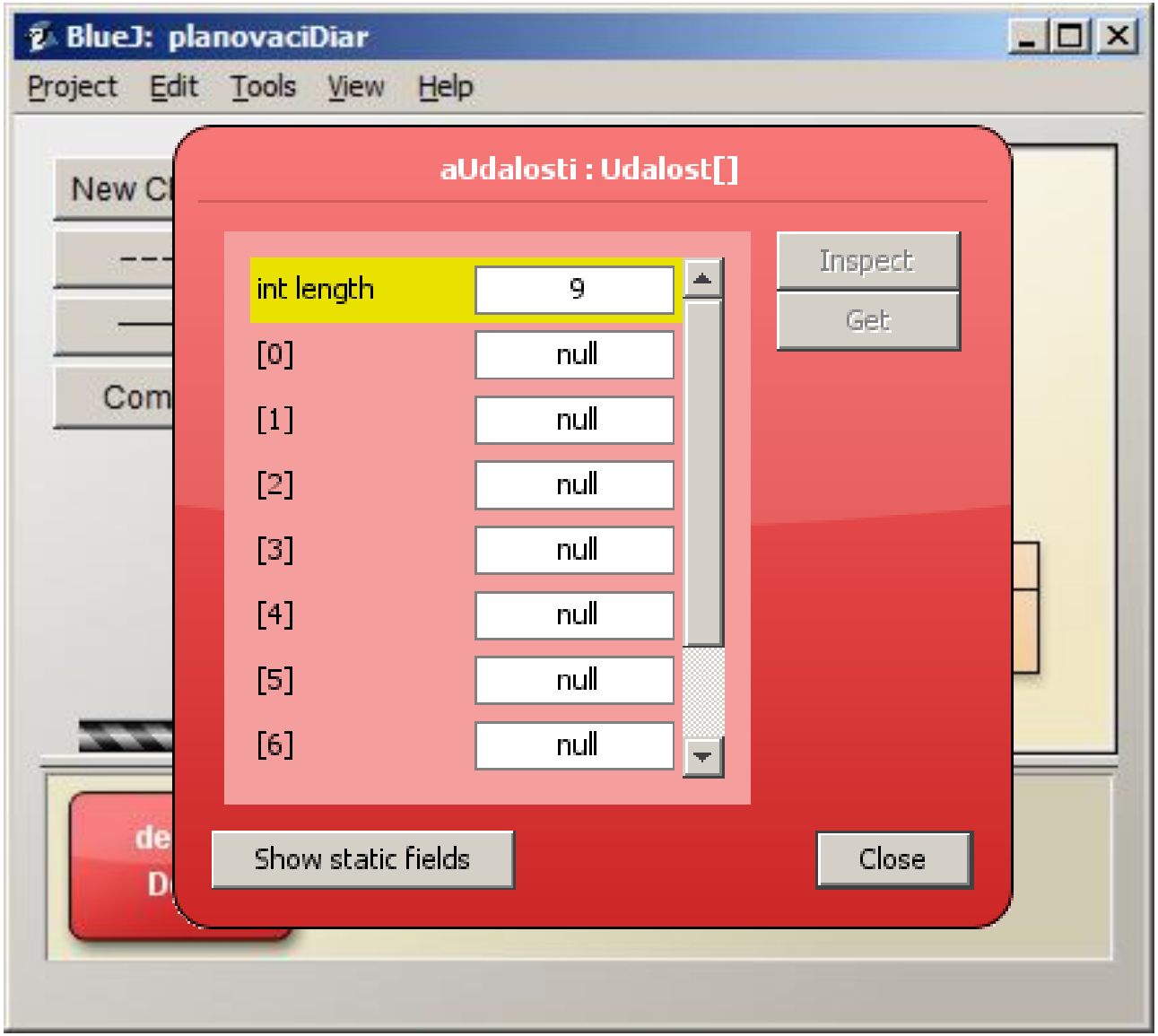


KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Object inspector – príklad(4)



Object inspector – príklad(5)



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 funkcionalita 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(1)

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

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

    ...
}
```


KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Príklad testu v JUnit(2)

```
@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));
}
```


KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Príklad testu v JUnit(3)

```
@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(pravdivostnyVyraz);
```

- 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(1)

```
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);
}
```


KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Príklad testu v Junit, Fixtures(2)

```
@Test
public void testVytvorTriUdalosti()
{
    assertTrue(den1.vlozUdalost(9, vymysliet));
    assertTrue(den1.vlozUdalost(10, vykonat));
    assertTrue(den1.vlozUdalost(11, zabudnut));
}
```

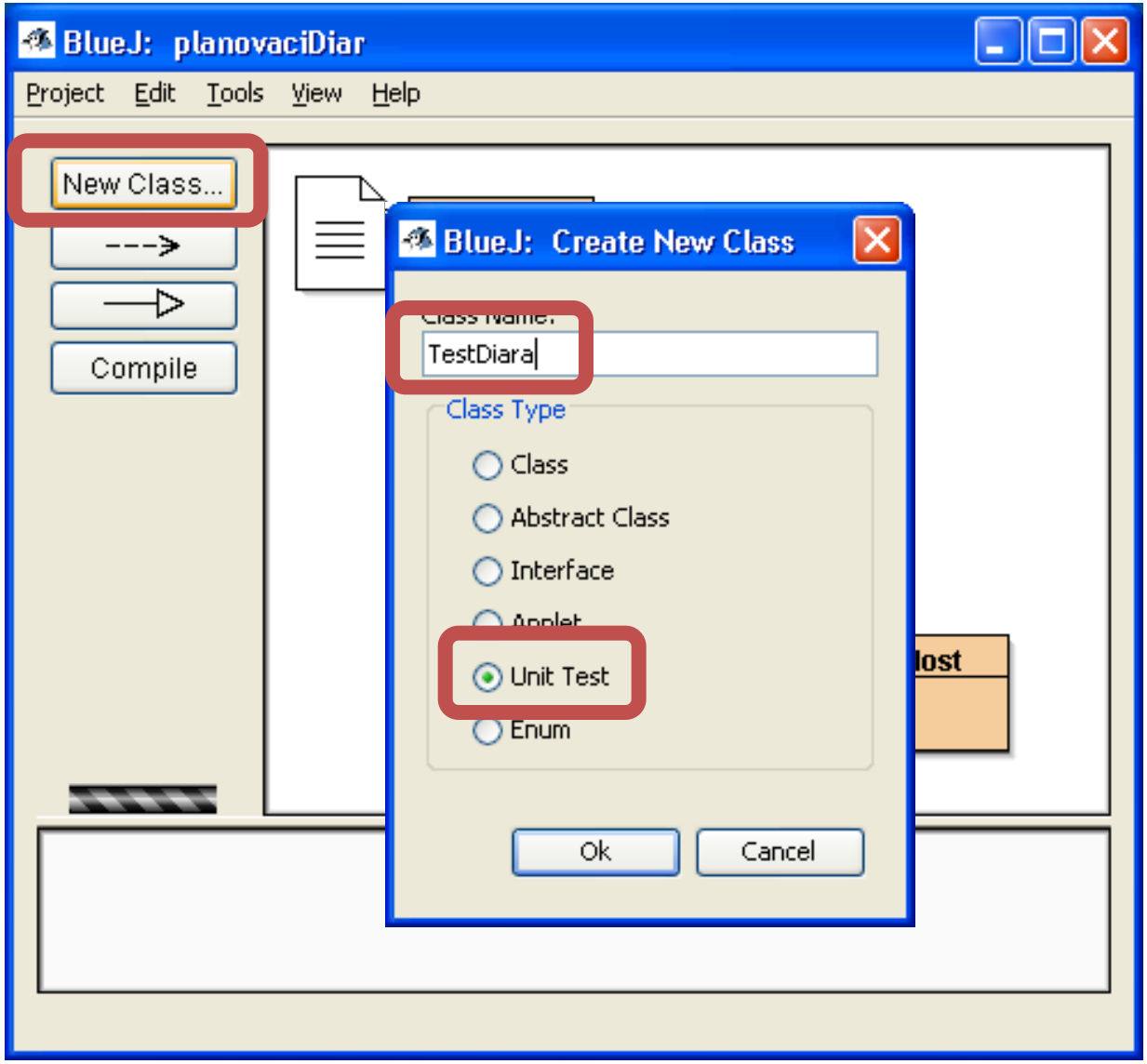
Príklad testu v Junit, Fixtures(3)

```
@Test
public void testOtestujUdalost()
{
    assertEquals(1, vymysliet.dajTrvanie());
    assertEquals(2, vykonat.dajTrvanie());
    assertEquals("Vymysliet", vymysliet.dajPopis());
    assertEquals("Vykonat", vykonat.dajPopis());
}
```

Unit testy v prostredí BlueJ

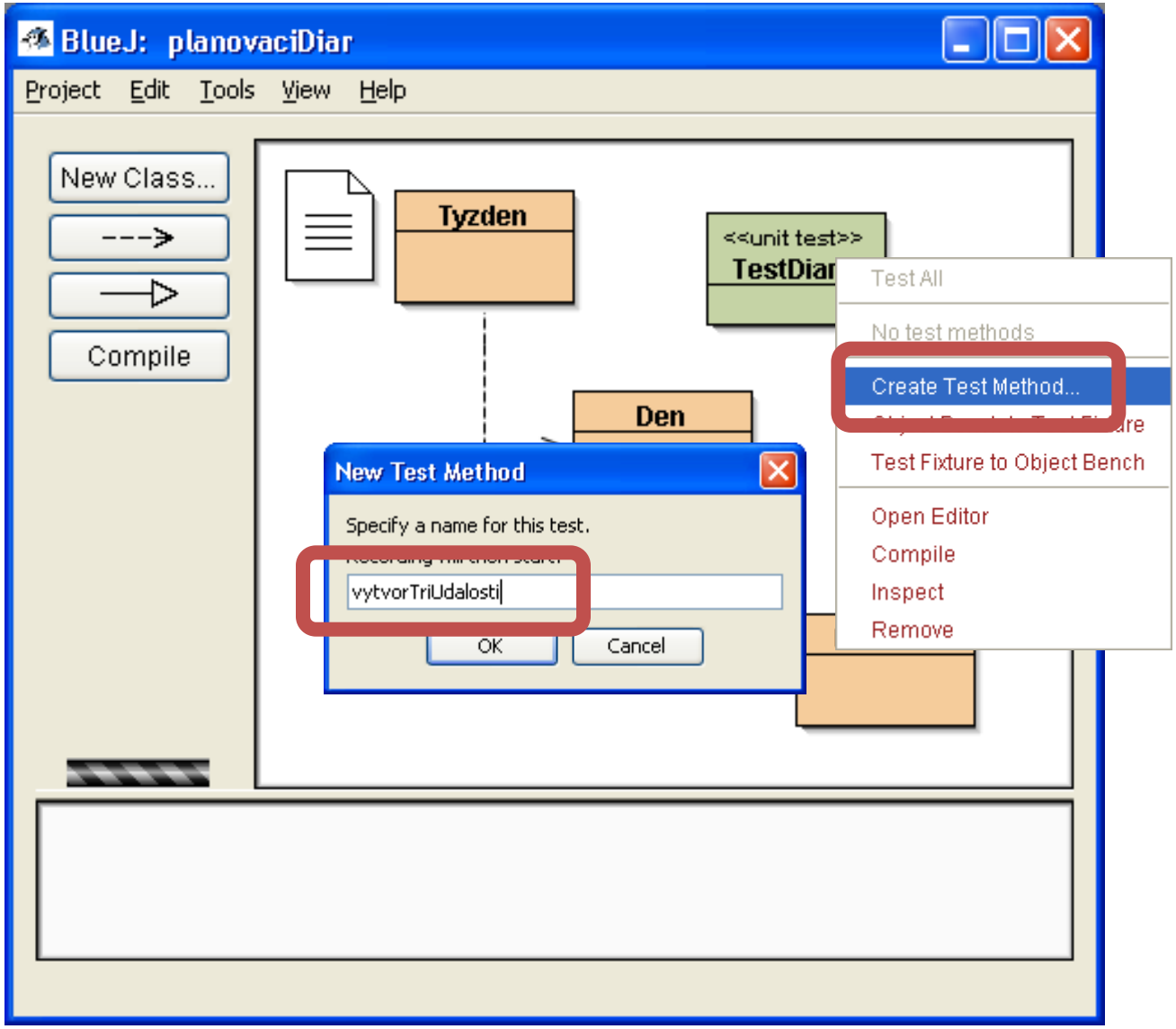
- využíva knižnicu JUnit
- „klikacie“ vytváranie testov
- zaznamenávanie činnosti testera
- dopĺňanie očakávaného parametra assertEquals
- záznam – telo testovacej metódy
- možnosť ukladania aktuálnych objektov v prostredí BlueJ ako Fixtures

Unit testy v prostredí BlueJ(1)



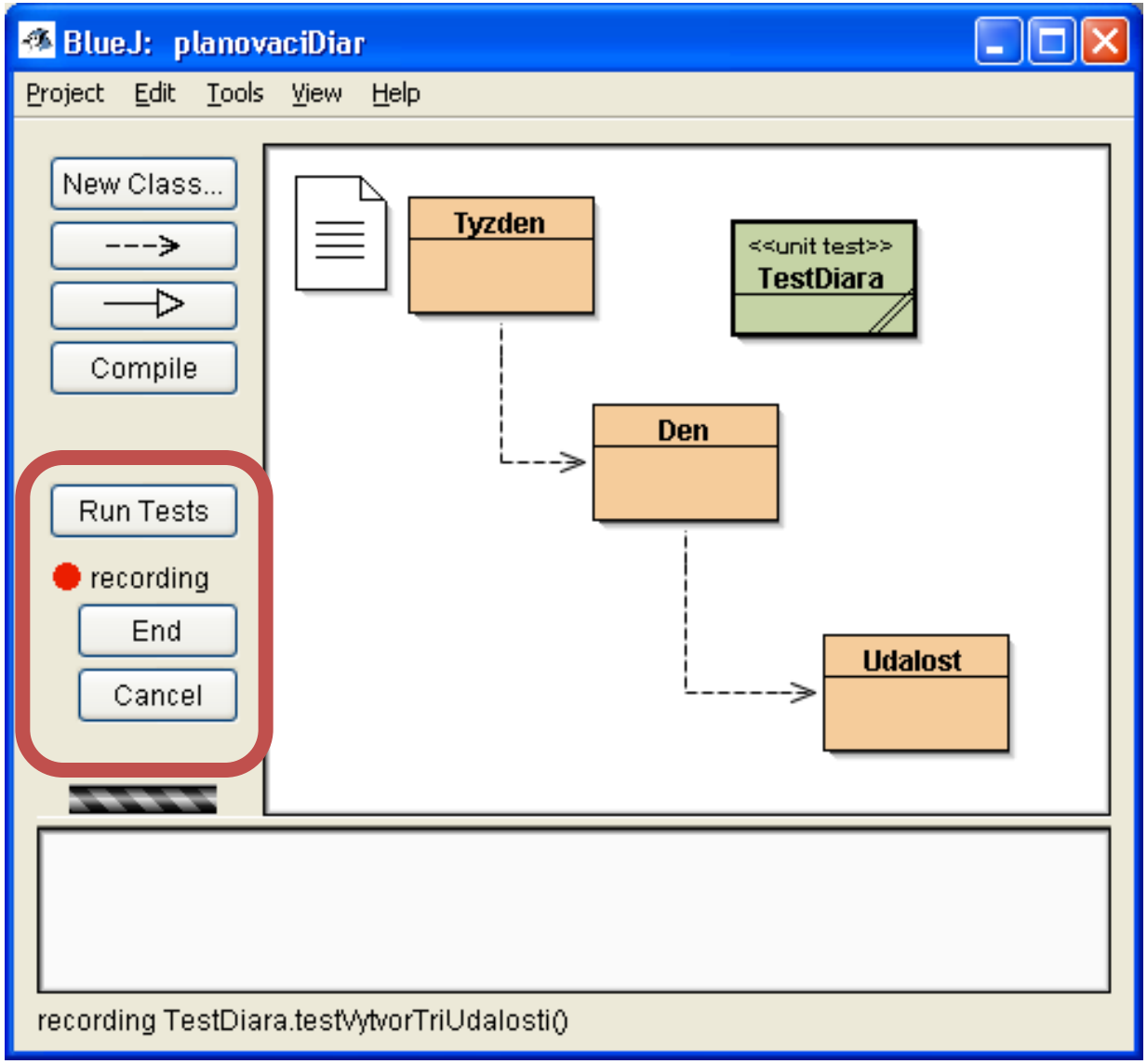
KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Unit testy v prostredí BlueJ(2)



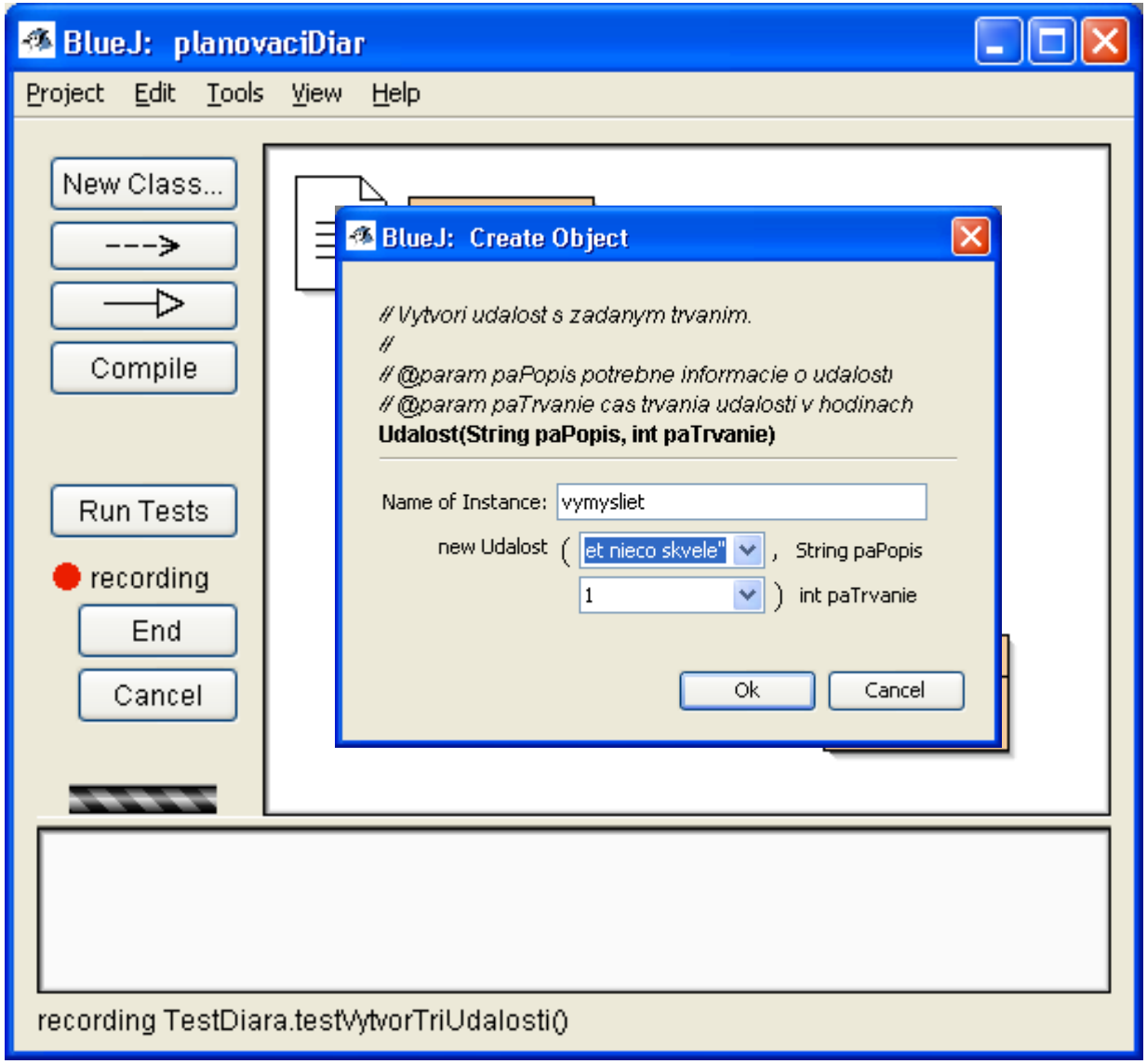
KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Unit testy v prostredí BlueJ(3)

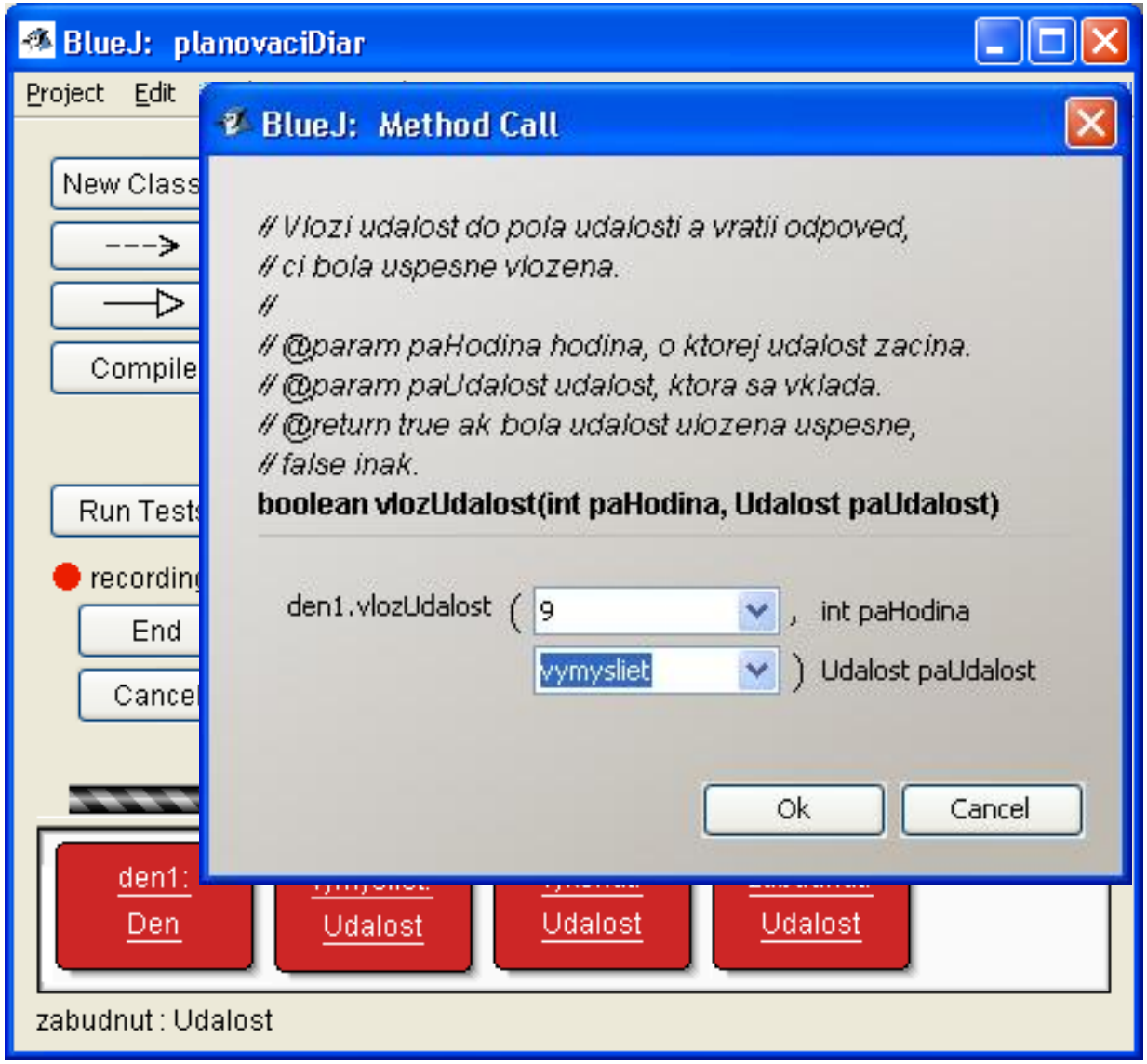


KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

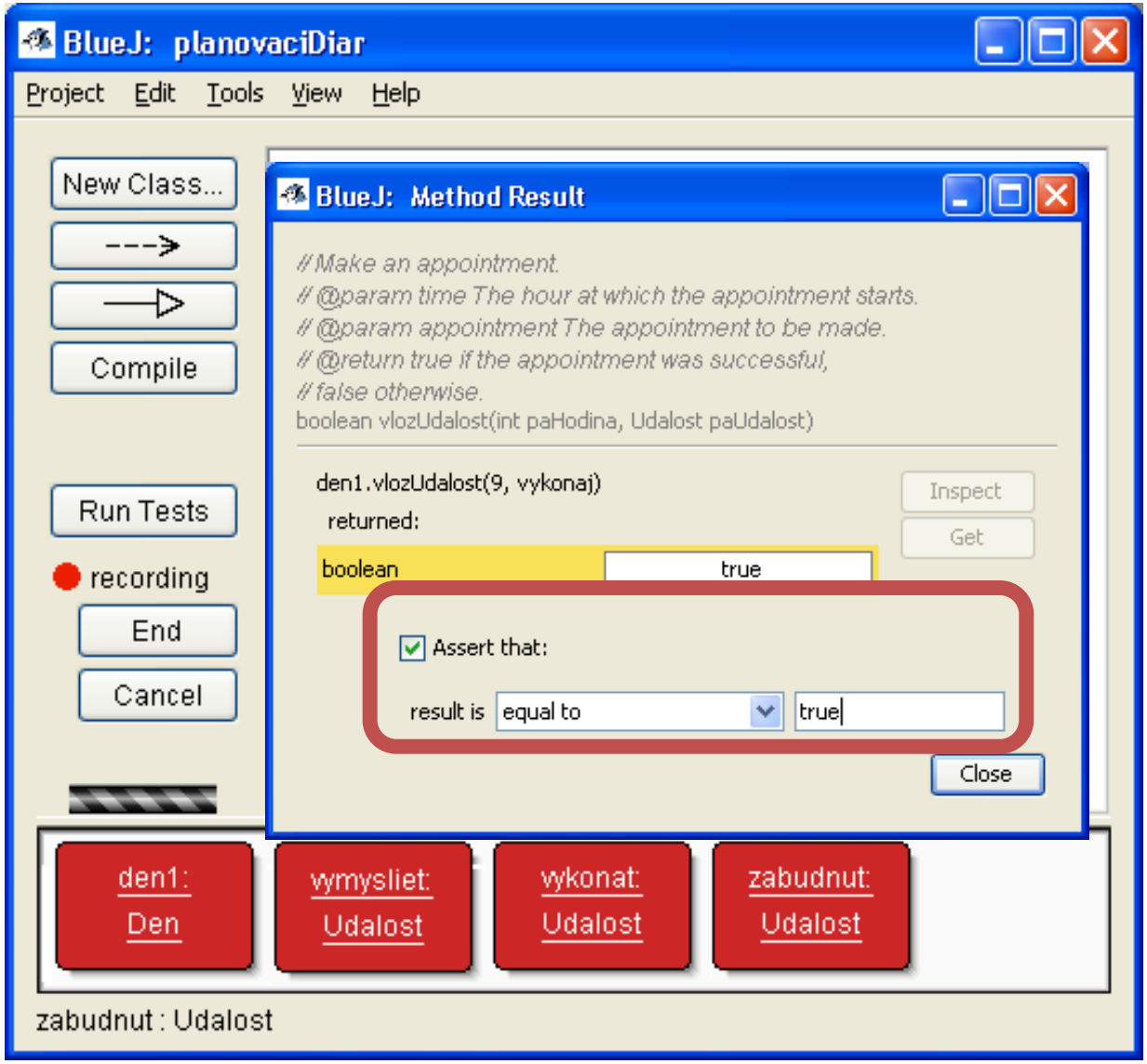
Unit testy v prostredí BlueJ(4)



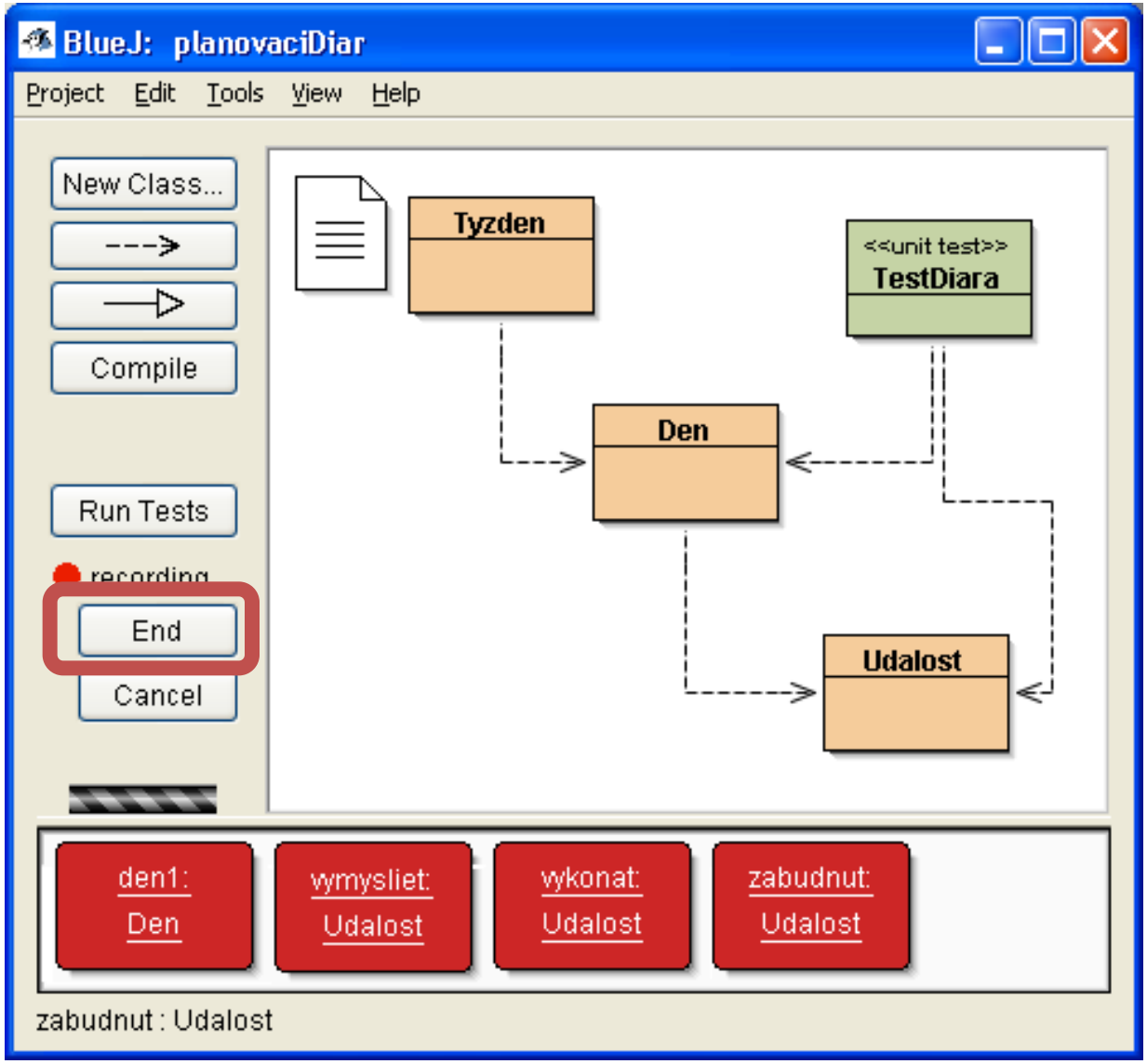
Unit testy v prostredí BlueJ(5)



Unit testy v prostredí BlueJ(6)

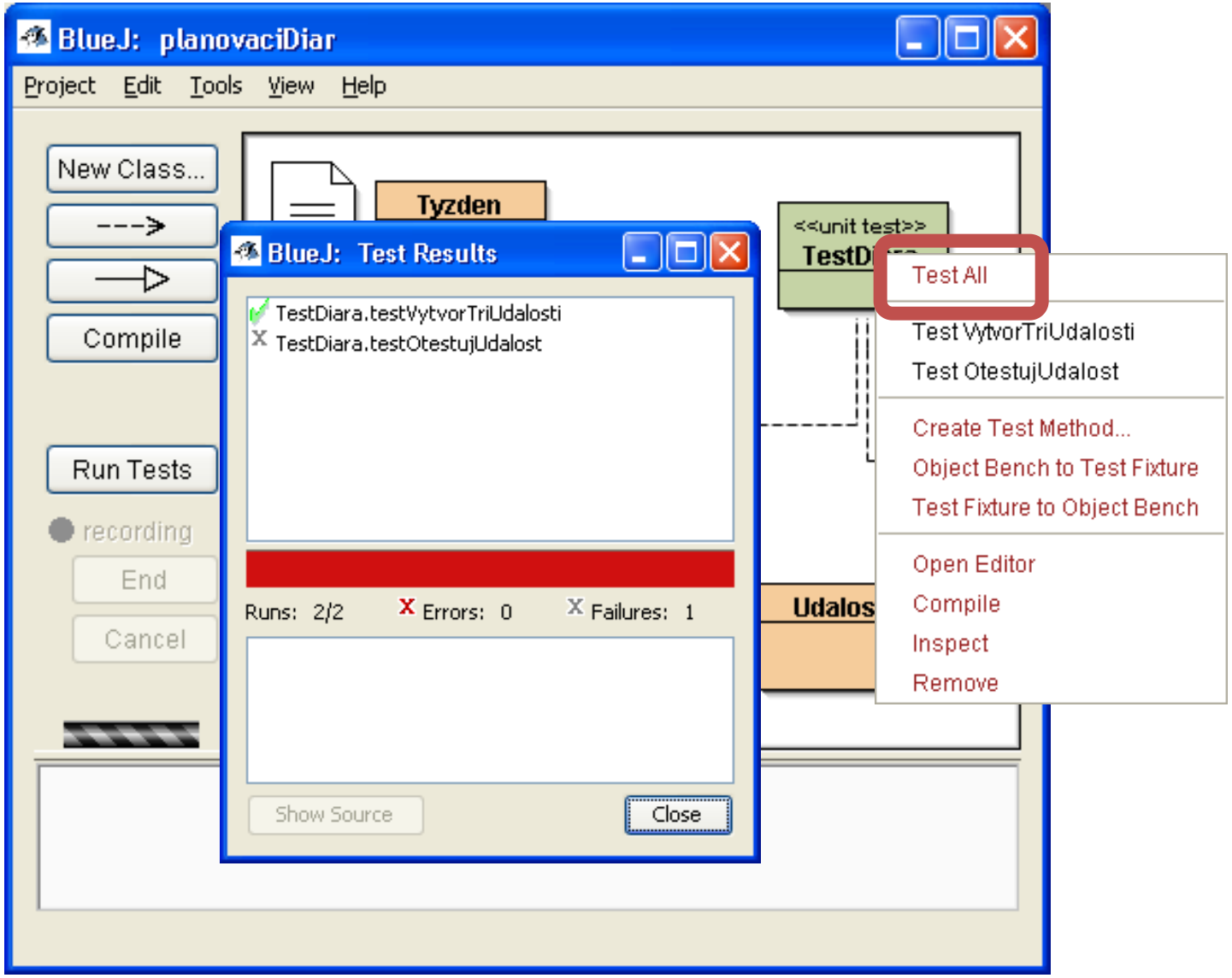


Unit testy v prostredí BlueJ(7)



KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Unit testy v prostredí BlueJ(8)



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

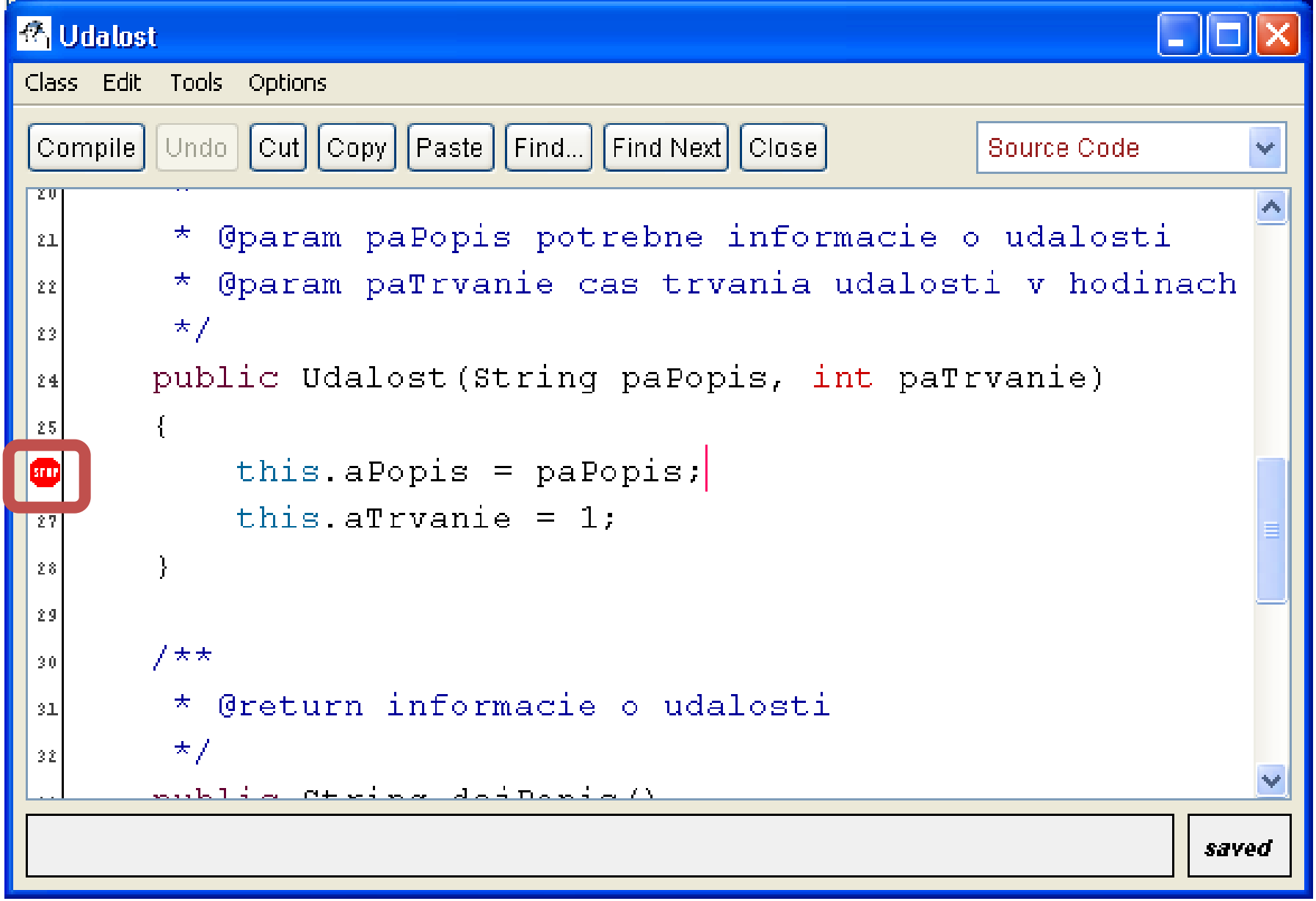
- jeden z najčastejších spôsobov ladenia

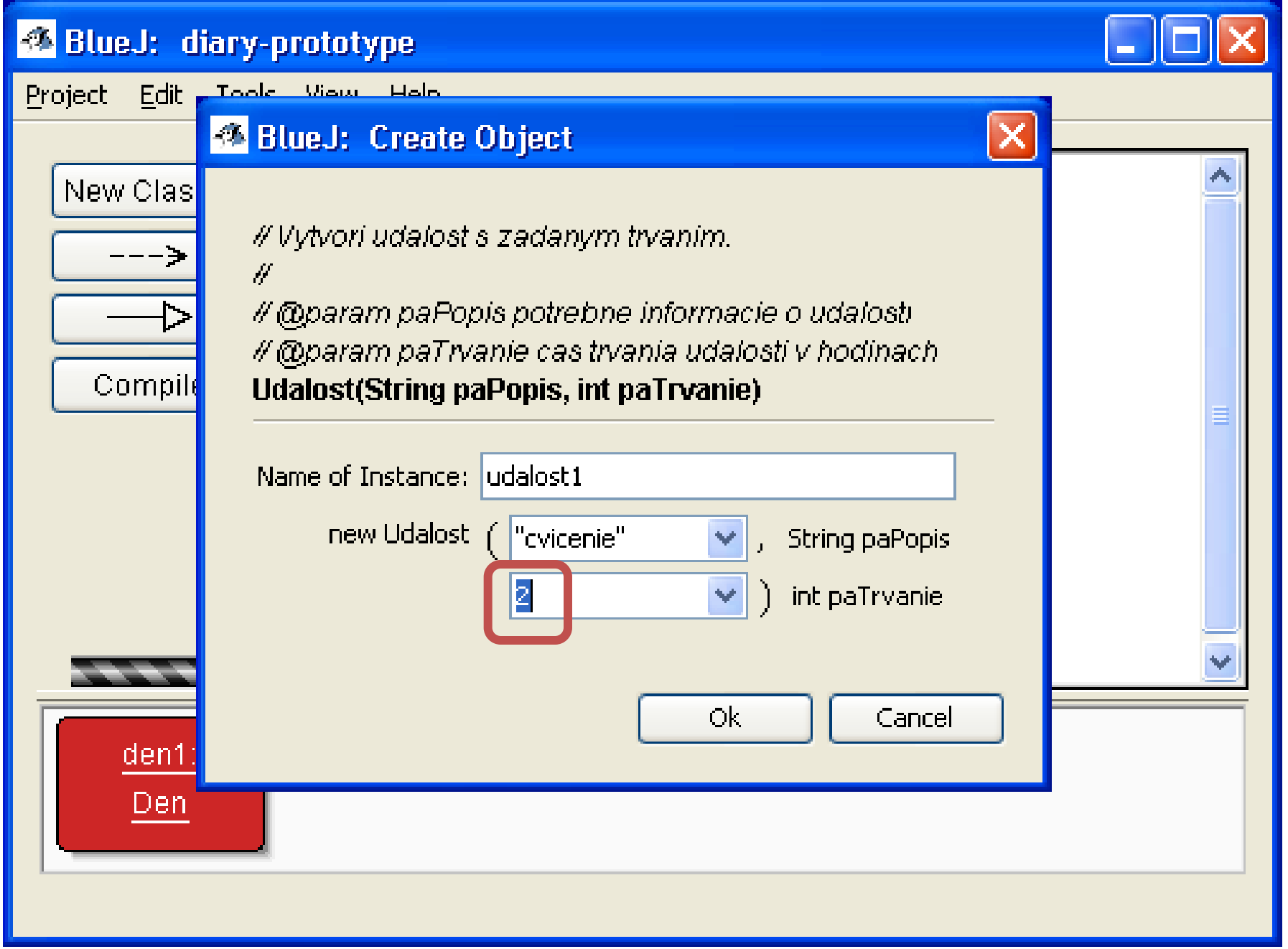
Ladiace výpisy

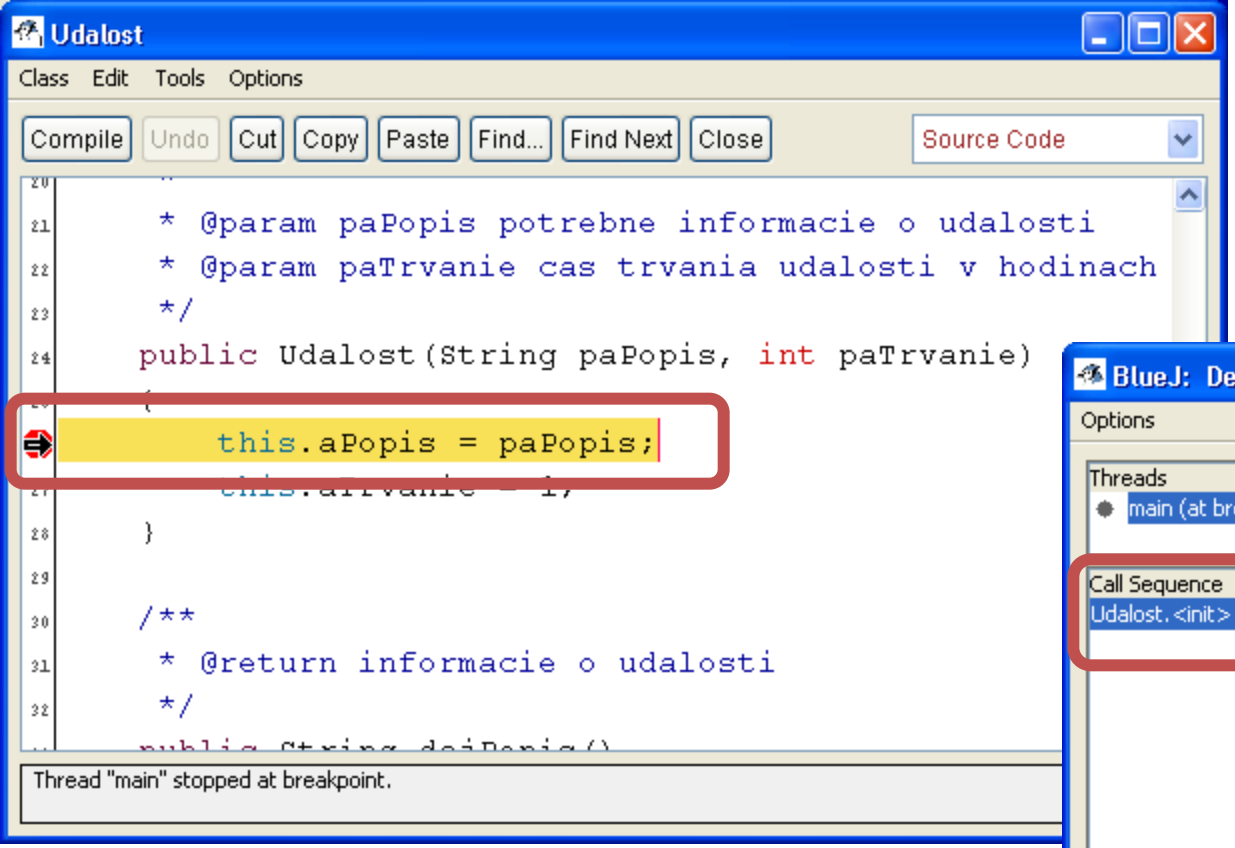
- rozšírenie programu o výpisy aktuálneho stavu objektov a algoritmov pomocou správy System.out.println
- programátor vo výpise vidí, kde sa objekty/algoritmy dostali do nesprávneho stavu
- ladiace výpisy môžu byť podmienené
  - zapoznámkovanie
  - ako vetva neúplného podmieneného príkazu
- ladiace výpisy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

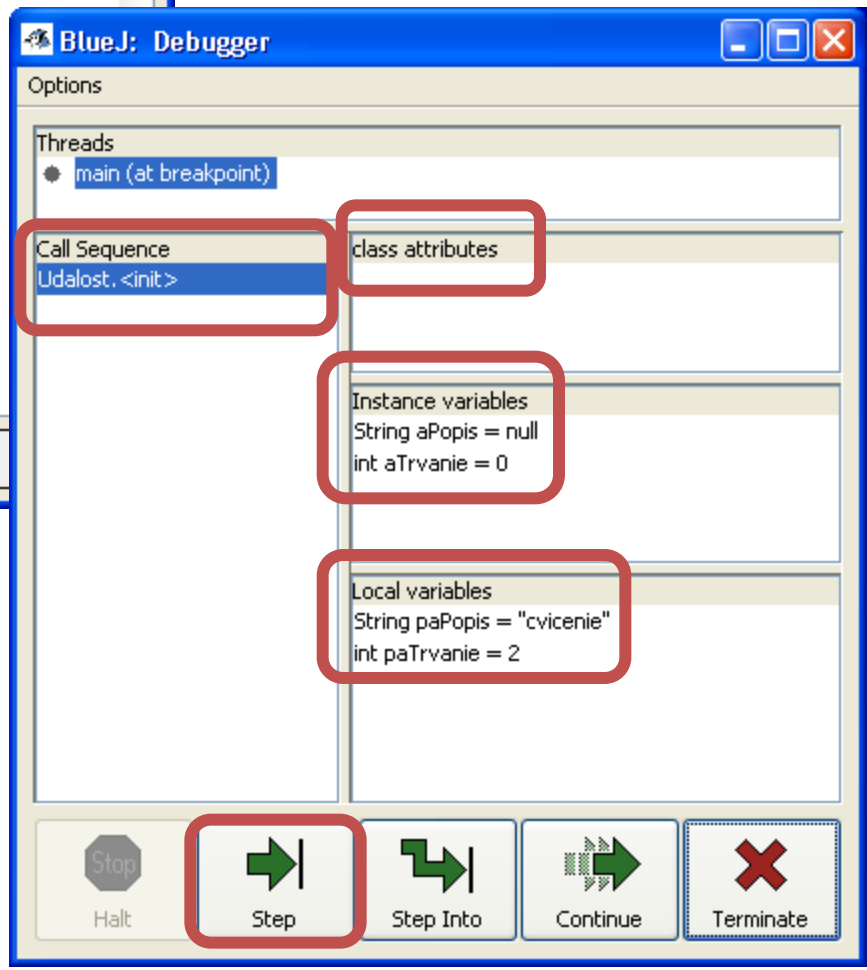
Debugger

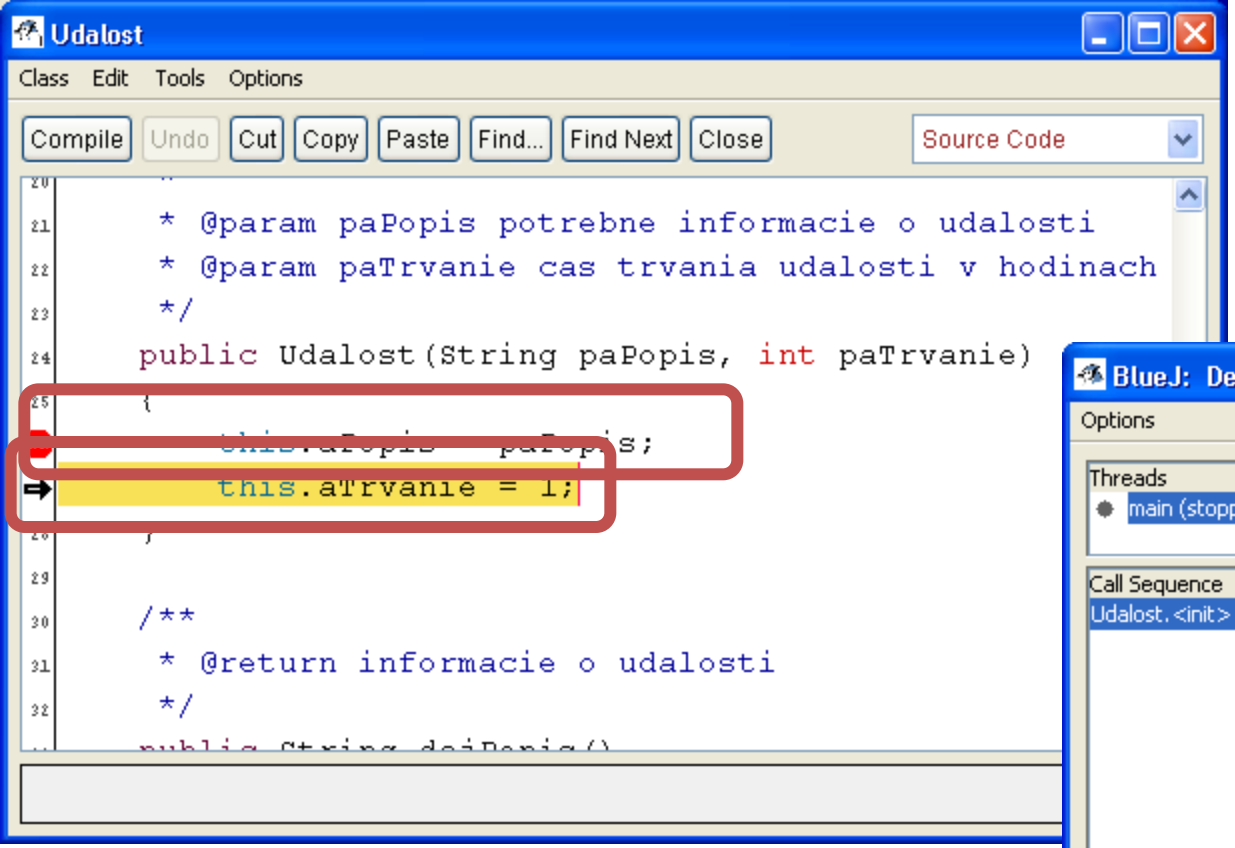
- bug (chyba) = chyba v programe
- debugger – program asistujúci pri hľadaní chýb
  - zobrazuje hodnoty všetkých dostupných premenných
  - označuje príkaz, ktorý má byť aktuálne vykonaný
- „krokovanie“ programu
- možnosť nastavenia zarážok (breakpoint)
- programátor vyhodnocuje správnosť dosiahnutého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

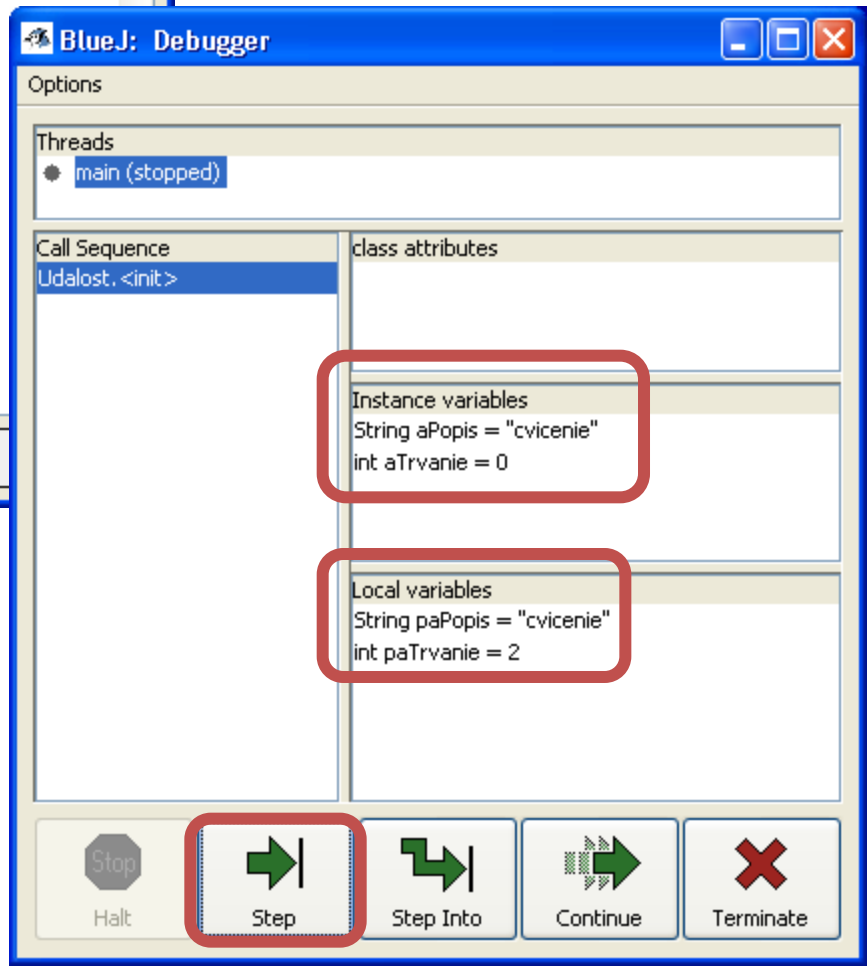

Udalost
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close
Source Code
20
21 * @param paPopis potrebne informacie o udalosti
22 * @param paTrvanie cas trvania udalosti v hodinach
23 */
24 public Udalost(String paPopis, int paTrvanie)
25 {
26 this.aPopis = paPopis;
27 this.aTrvanie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


BlueJ: diary-prototype
Project Edit Tools View Help
New Clas
Compile
den1:
Den
BlueJ: Create Object
// Vytvori udalost s zadanym trvanim.
//
// @param paPopis potrebne informacie o udalosti
// @param paTrvanie cas trvania udalosti v hodinach
Udalost(String paPopis, int paTrvanie)
Name of Instance: udalost1
new Udalost ("cvicenie" , String paPopis
2) int paTrvanie
Ok
Cancel

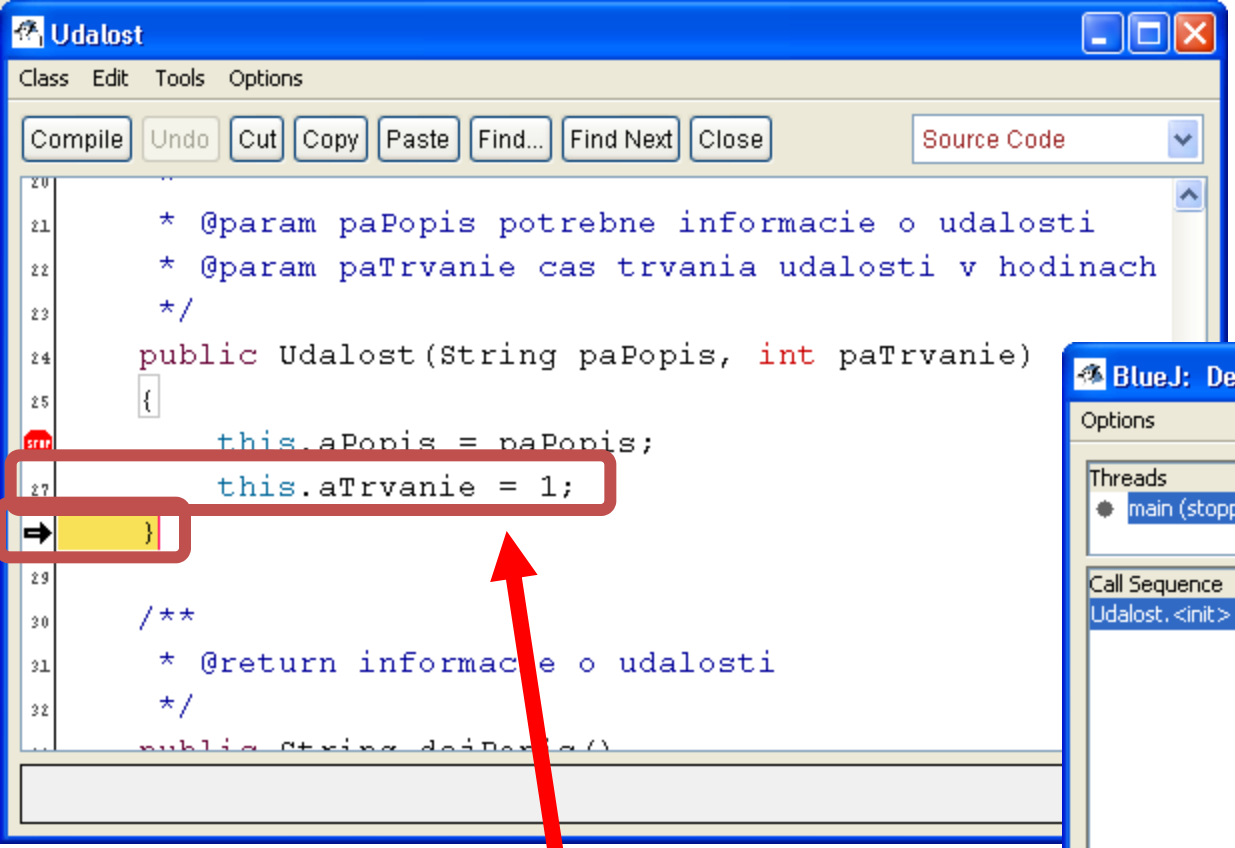

Udalost
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close
Source Code
* @param paPopis potrebne informacie o udalosti
* @param paTrvanie cas trvania udalosti v hodinach
*/
public Udalost(String paPopis, int paTrvanie)
{
this.aPopis = paPopis;
}
/**
* @return informacie o udalosti
*/
Thread "main" stopped at breakpoint.

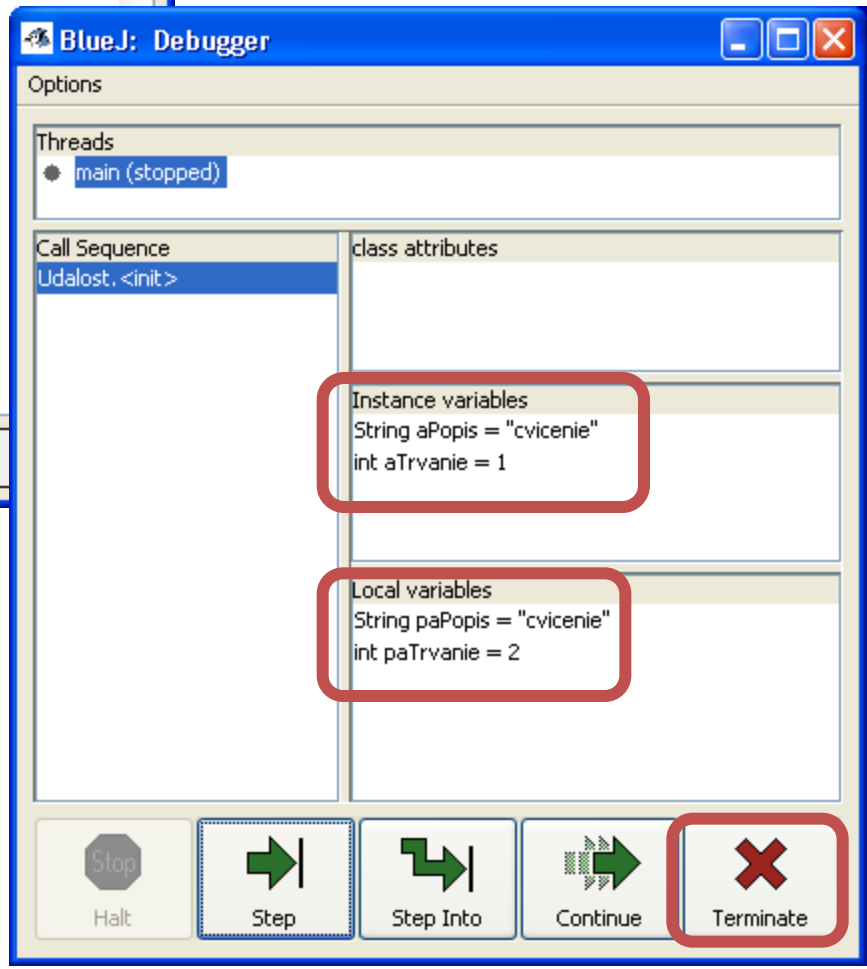

BlueJ: Debugger
Options
Threads
main (at breakpoint)
Call Sequence
Udalost.<init>
class attributes
Instance variables
String aPopis = null
int aTrvanie = 0
Local variables
String paPopis = "cvicenie"
int paTrvanie = 2
Halt
Step
Step Into
Continue
Terminate



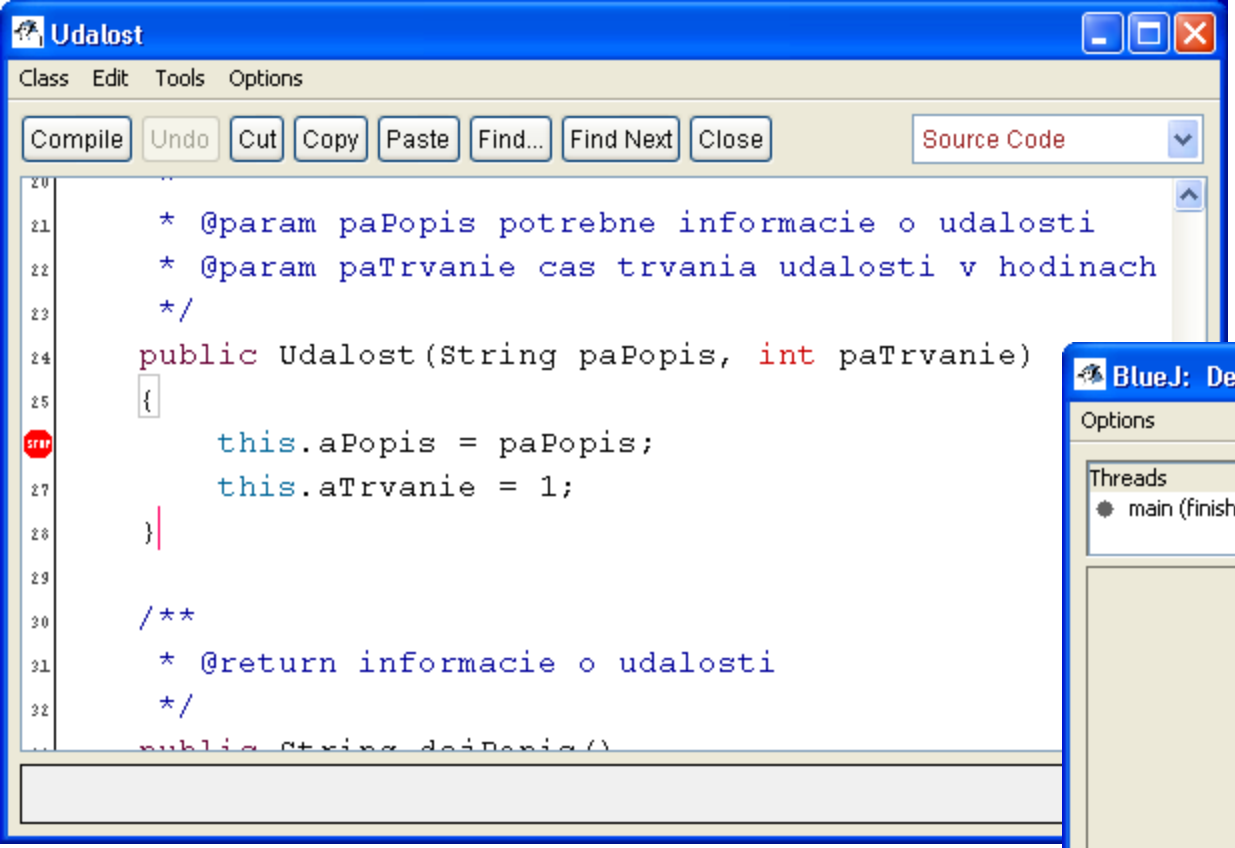


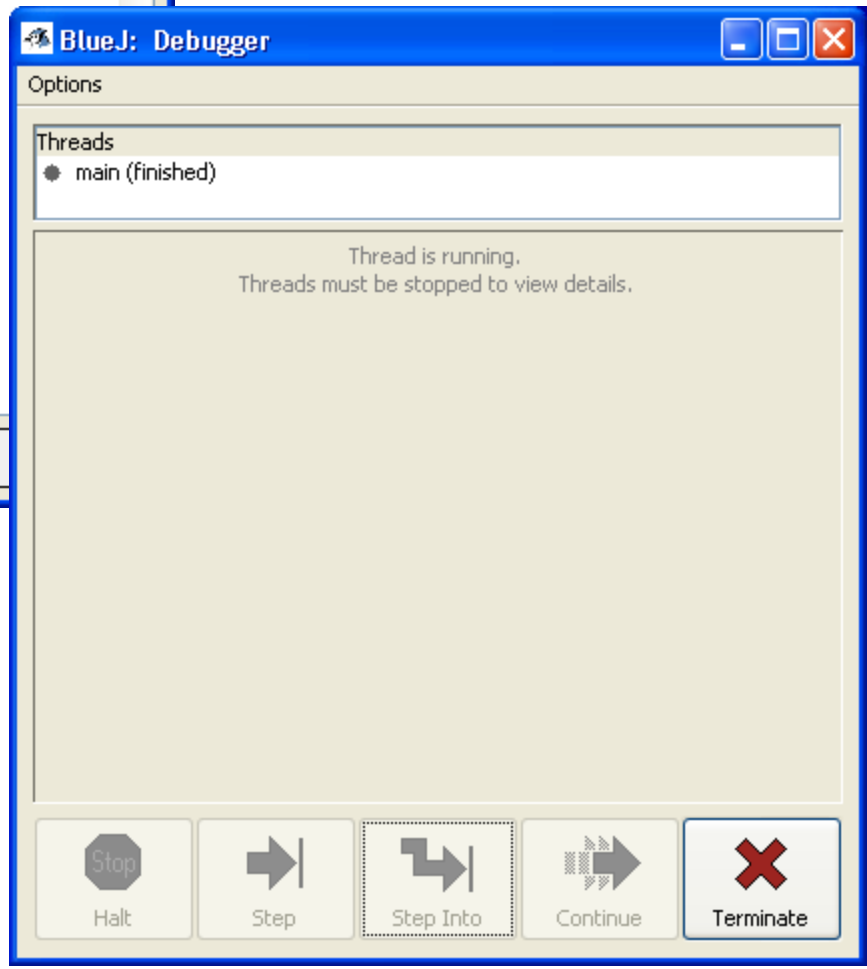
KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline


Udalost
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close
Source Code
* @param paPopis potrebne informacie o udalosti
* @param paTrvanie cas trvania udalosti v hodinach
*/
public Udalost(String paPopis, int paTrvanie)
{
this.aPopis = paPopis;
this.aTrvanie = 1;
}
/**
* @return informacie o udalosti
*/


BlueJ: Debugger
Options
Threads
main (stopped)
Call Sequence
Udalost.<init>
class attributes
Instance variables
String aPopis = "cvicenie"
int aTrvanie = 1
Local variables
String paPopis = "cvicenie"
int paTrvanie = 2
Halt
Step
Step Into
Continue
Terminate

chyba

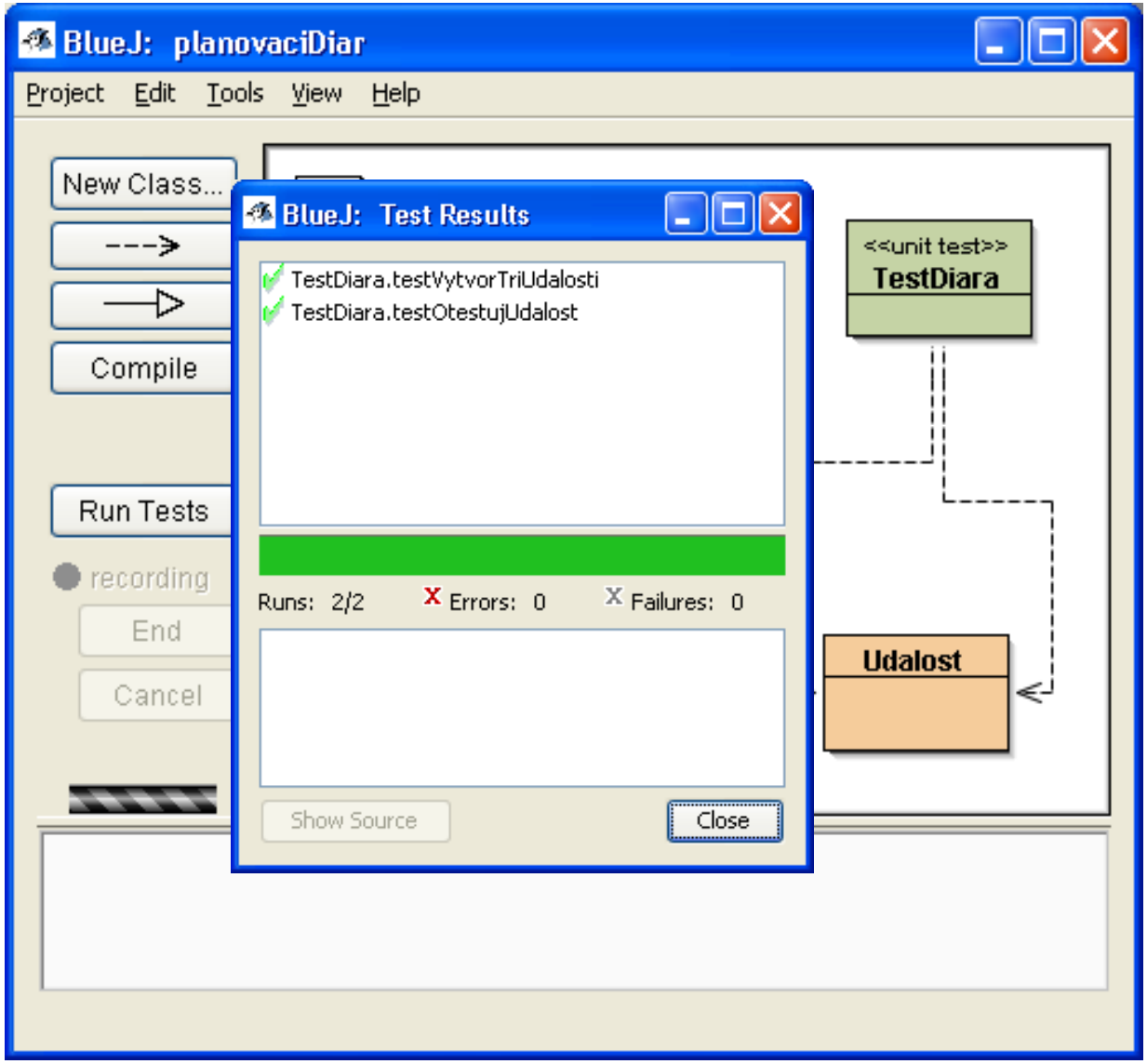

Udalost
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close
Source Code
* @param paPopis potrebne informacie o udalosti
* @param paTrvanie cas trvania udalosti v hodinach
*/
public Udalost(String paPopis, int paTrvanie)
{
this.aPopis = paPopis;
this.aTrvanie = 1;
}
/**
* @return informacie o udalosti
*/


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

Výsledok po oprave



KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

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