9. Ladění a testování programů

- Ladění programů s debuggerem jdb
- Nástroje ověřování podmínek za běhu - klíčové slovo assert
- Nástroje testování jednotek (tříd, balíků) - junit
- Pokročilé systémy dynamického ověřování podmínek - jass

Ladění programů v Javě

Je mnoho způsobů...

- kontrolní tisky - `System.err.println(...)`
- řádkovým debuggerem **`jdb`**
- integrovaným debuggerem v IDE
- pomocí speciálních nástrojů na záznam běhu pg.:

  nejrůznější "loggery" - standardní poskytuje od JDK1.4 balík `java.util.logging` nebo alternativní a zdařilejší **`log4j`**

Ještě lepší...

- je používat systémy pro běhovou kontrolu platnosti podmínek:
  - vstupní podmínka metody (zda je volána s přípustnými parametry)
  - výstupní podmínka metody (zda jsou dosažené výstupy správné)
  - a podmínka kdekoli jinde - např. invariant cyklu...
- K tomuto slouží jednak
  - standardní klíčové slovo (od JDK1.4) `assert` ***`booleovský výraz`***
  - testovací nástroje typu **`JUnit`** (a varianty - `HttpUnit`,...) - s metodami `assertEquals()` apod.
  - pokročilé nástroje na běhovou kontrolu platnosti invariantů, vstupních, výstupních a dalších podmínek - např. `jass` (Java with ASSertions), http://csd.informatik.uni-oldenburg.de/~jass/.
- Ukázka testu jednotky:Test třídy `ChovatelPsu`

Postup při práci s `assert`

Postup:

1. Napsat zdrojový program užívající klíčové slovo **assert** (pouze od verze Java2 v1.4 výše). Nepotřebujeme žádné speciální běhové knihovny, vše je součástí Javy; musíme ovšem mít překladové i běhové prostředí v1.4 a vyšší.
2. Přeložit jej s volbou `-source 1.4`

3. Spustit jej s volbou `-ea` (`-enableassertions`).

   Aktivovat aserce lze i selektivně pro některé třídy (`-ea název_třídy` nebo `-ea název_balíku...` - tři tečky na konci!!!).

4. Dojde-li za ***běhu programu*** k porušení podmínky stanovené za assert, vznikne běhová chyba (AssertionError) a program skončí.

Ukázka použití assert (1)

Třída Zlomek používá assert k ověření, že zlomek není (chybou uživatele) vytvářen s nulovým jmenovatelem.

Za **assert** uvedeme, co musí v daném místě za běhu programu platit.

Obrázek 8.1. Třídy AssertDemo, Zlomek

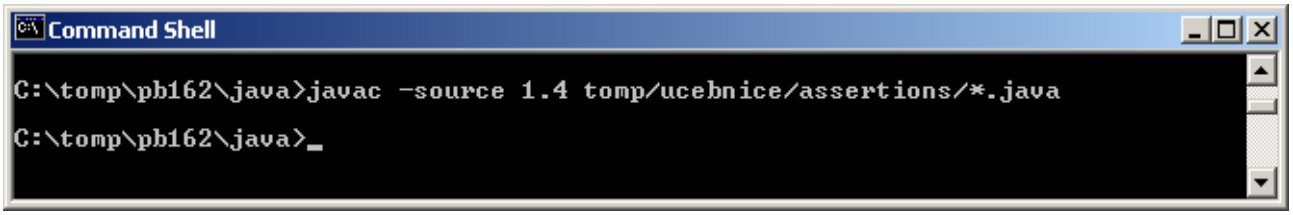
AssertDemo.java

```
package tomp.ucebnice.assertions;

public class AssertDemo {
    public static void main(String[] args) {

        Zlomek x = new Zlomek(1,2);
        System.out.println(x);

        Zlomek y = new Zlomek(1,0);
        System.out.println(y);

    }
}
class Zlomek {

    int citatel, jmenovatel;

    public Zlomek(int c, int j) {

        // ujistíme se, že jmenovatel nebude nula
        assert j != 0;

        citatel = c;
        jmenovatel = j;
    }

    public String toString() {
        return citatel+"/"+jmenovatel;
    }
}
```

Ukázka použití assert (2)

Program přeložíme (s volbou `-source 1.4`):

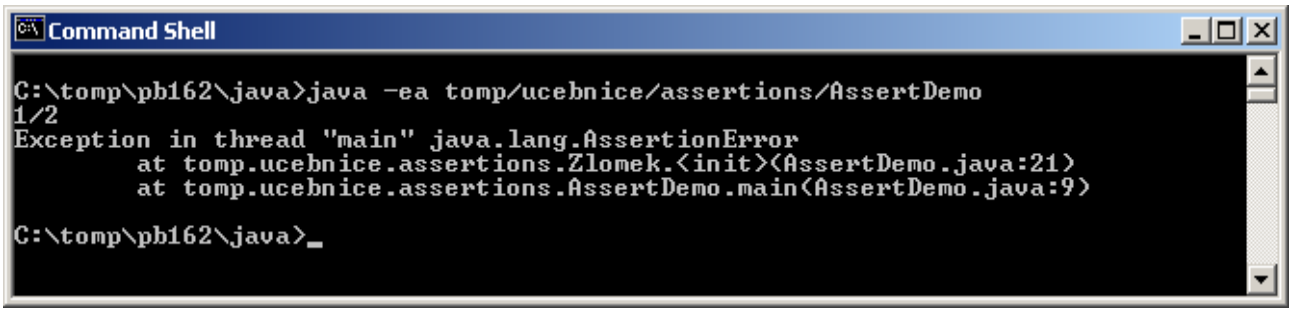
Obrázek 8.2. Správný postup překladu AssertDemo



Ukázka použití assert (3)

Program spustíme (s volbou `-ea` nebo selektivním `-ea:NázevTřídy`):

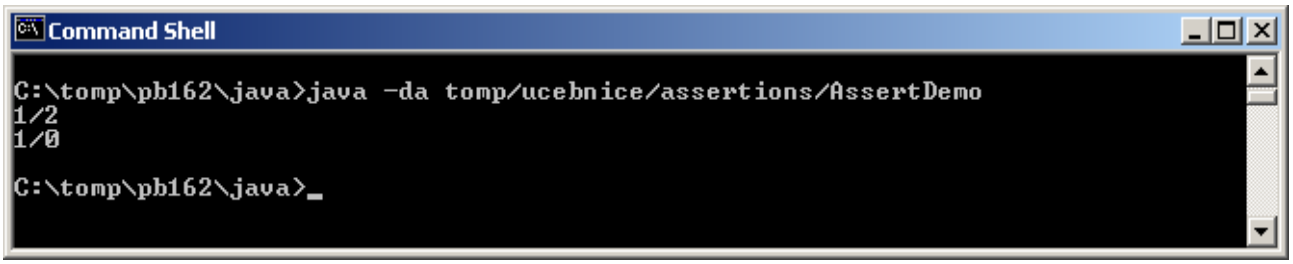
Obrázek 8.3. Spuštění AssertDemo s povolením assert



Ukázka použití assert (4)

Spustíme-li bez povolení assert (bez volby `-ea` nebo naopak s explicitním zákazem: `-da`), pak program podmínky za assert neověřuje:

Obrázek 8.4. Spuštění AssertDemo bez povolení assert



Postup při práci s `JUnit`

Uvědomit si, že žádný nástroj za nás nevymyslí, JAK máme své třídy testovat. Pouze nám napomůže ke snadnějšímu sestavení a spuštění testu.

Postup:

1. Stáhnout si z http://junit.org poslední (stačí binární) distribuci testovacího prostředí.
2. Nainstalovat `JUnit` (stačí rozbalit do samostatného adresáře).
3. Napsat testovací třídu/třídy - buďto implementují rozhraní `junit.framework.Test` nebo obvykleji rovnou rozšiřují třídu `junit.framework.TestCase`
4. Testovací třída obsahuje metodu na nastavení testu (`setUp`), testovací metody (test***Neco***) a úklidovou metodu (`tearDown`).
5. Testovací třídu spustit v prostředí (řádkovém nebo GUI) - `junit.textui.TestRunner`, `junit.swingui.TestRunner`...

6. Testovač zobrazí, které testovací metody případně selhaly.

Ukázka použití JUnit (1)

Třída Zlomek zůstává zhruba jako v předchozím příkladu, přibývají však metody `equals` (porovnává dva zlomky, zda je jejich číselná hodnota stejná) a `soucet` (sečítá dva zlomky, součet vrací jako výsledek).

Obrázek 8.5. Upravená třída Zlomek s porovnáním a součtem

JUnitDemo.java | Assert.java

```
}
class Zlomek {

    int citatel, jmenovatel;

    public Zlomek(int c, int j) {

        assert j != 0;

        citatel = c;
        jmenovatel = j;
    }

    public boolean equals(Object o) {
        if (o instanceof Zlomek) {
            Zlomek z = (Zlomek)o;
            return citatel * z.jmenovatel == z.citatel * jmenovatel;
        } else
            return false;
    }

    public static Zlomek soucet(Zlomek a, Zlomek b) {
        return new Zlomek(a.citatel*b.jmenovatel + b.citatel*a.jmenovatel,
                          a.jmenovatel*b.jmenovatel);
    }

    public String toString() {
        return citatel+"/"+jmenovatel;
    }
}
```

Ukázka použití JUnit (2)

Testovací třída JUnitDemo má „přípravnou" metodu `setUp`, `tearDown` a testovací metody.

Obrázek 8.6. Testovací třída JUnitDemo

JUnitDemo.java | Assert.java

```
package tomp.ucebnice.junit;

public class JUnitDemo extends junit.framework.TestCase {

    private Zlomek x, y, soucet, xx;

    public void setUp() {
         x = new Zlomek(1,2);
         xx = new Zlomek(6, 12);
         y = new Zlomek(1,3);
         soucet = new Zlomek(5,6);
    }

    public void testRovna() {
        assertEquals("1/2 a 6/12 se musi rovnat", x, xx);
    }

    public void testSoucetRovna() {
        Zlomek s = Zlomek.soucet(x, y);
        assertEquals("Soucet 1/2 a 1/3 se musi rovnat 5/6", soucet, s);
    }

    public void testNelzeVytvorit() {
        try {
            Zlomek bad = new Zlomek(10,0);
            fail("Zlomek s nulovym jmenovatelem nemel jit vytvorit");
        } catch(AssertionError ae) {
            // OK!
        }
    }

    public void tearDown() {
    }

}
```

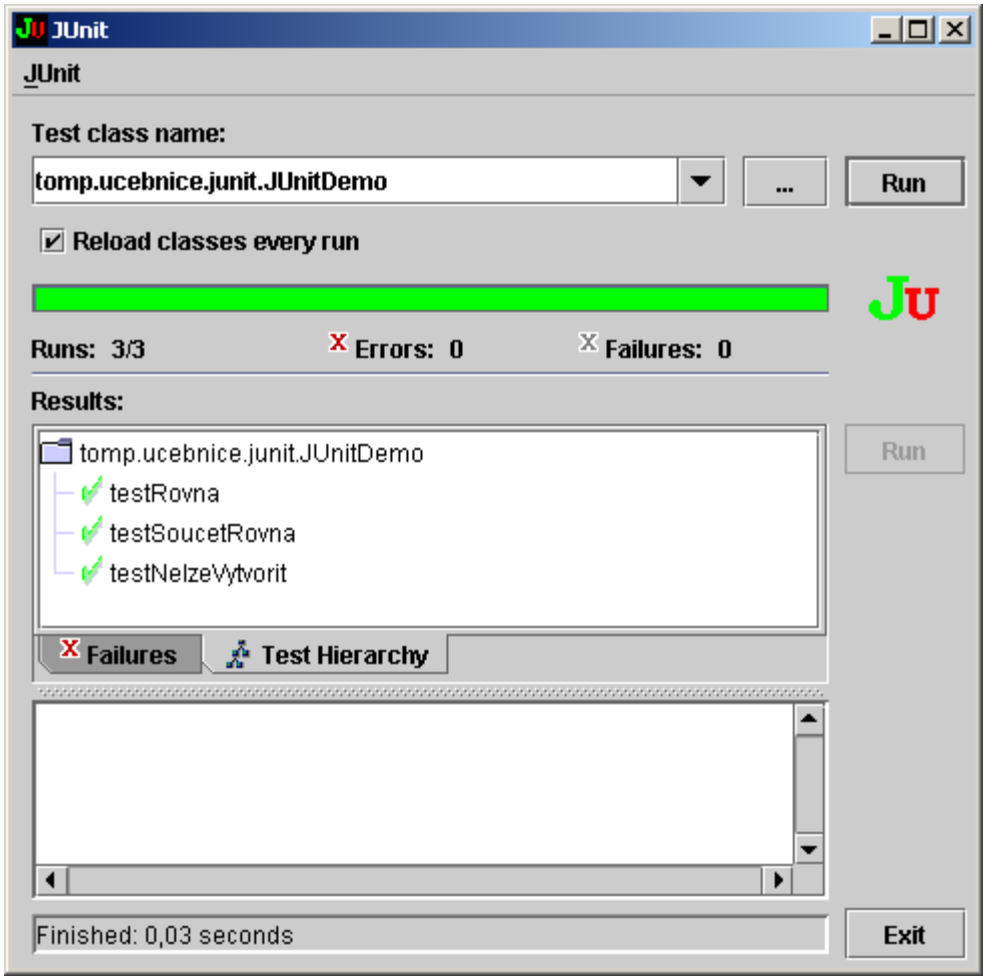
Ukázka použití JUnit (3)

Spuštění testovače v prostředí GUI Swing nad testovací třídou JUnitDemo.

Obrázek 8.7. Spouštění testovače nad testovací třídou JUnitDemo

Pokud testovací třída prověří, že testovaná třída/y je/jsou OK, vypadá to přibližně takto:

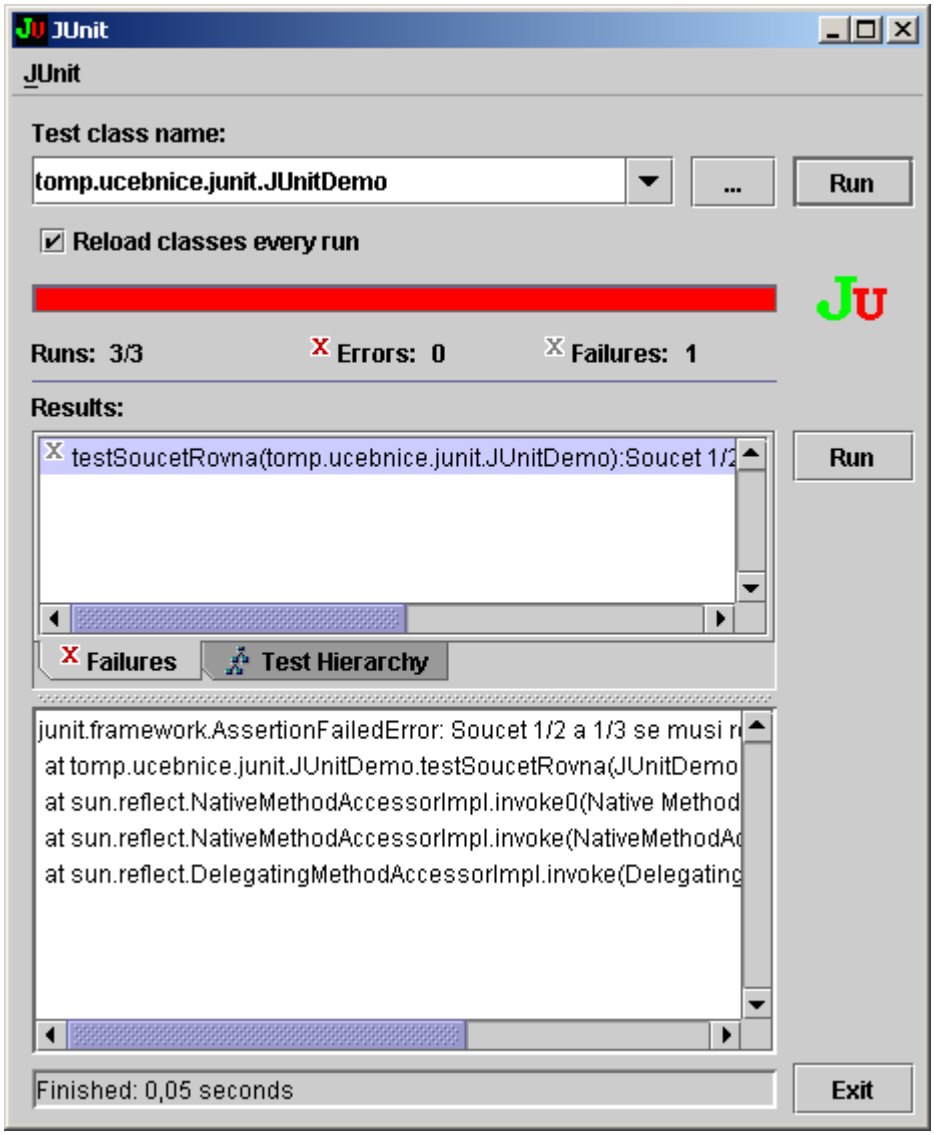
Obrázek 8.8. Testovač spouštějící třídu JUnitDemo



Ukázka použití JUnit (4)

Má-li testovaná třída/y chyby a zjistí-li to testovač, vypadá to třeba takto:

Obrázek 8.9. Testovací třída JUnitDemo našla chybu



Postup při práci s `jass`

jass je preprocesor javového zdrojového textu. Umožňuje ve zdrojovém textu programu vyznačit podmínky, jejichž splnění je za běhu kontrolováno.

Podmínkami se rozumí:

- pre- a postconditions u metod (vstupní a výstupní podmínky metod)
- invarianty objektů - podmínky, které zůstávají pro objekt v platnosti mezi jednotlivými operacemi nad objektem

Postup práce s jass:

- stažení a instalace balíku z http://csd.informatik.uni-oldenburg.de/~jass/
- vytvoření zdrojového textu s příponou `.jass`, javovou syntaxí s použitím speciálních komentářových značek
- takový zdrojový text je přeložitelný i normálním překladačem **javac**, ale v takovém případě ztrácíme možnosti jass
- proto nejprve `.jass` souboru převedeme preprocesorem jass na javový (`.java`) soubor
- ten již přeložíme **javac** a spustíme **java**, tedy jako každý jiný zdrojový soubor v Javě

- z `.jass` zdrojů je možné vytvořit také dokumentaci API obsahující jass značky, tj. informace, co kde musí platit za podmínky atd. - vynikající možnost!

Odkazy

- JUnit homepage
- Java 1.4 logging API guide
- Log4j homepage
- jass homepage
- úvodní materiálek k použití junit (v němčině, jako PDF)