
Kapitola 1. Přednáška 7 - testování a ladění programů. junit, assert. Javové operátory, složitější výrazy.

Obsah

Ladění a testování programů	1
Ladění programů v Javě	2
Ještě lepší... ..	2
Postup při práci s <code>assert</code>	2
Ukázka použití <code>assert</code> (1)	3
Ukázka použití <code>assert</code> (2)	4
Ukázka použití <code>assert</code> (3)	5
Ukázka použití <code>assert</code> (4)	5
Postup při práci s <code>JUnit</code>	5
Ukázka použití <code>JUnit</code> (1)	6
Ukázka použití <code>JUnit</code> (2)	7
Ukázka použití <code>JUnit</code> (3)	7
Ukázka použití <code>JUnit</code> (4)	8
Postup při práci s <code>jass</code>	9
Odkazy	10
Operátory a výrazy, porovnávání objektů	10
Aritmetické	11
Logické	11
Relační (porovnávací)	11
Porovnávání objektů	12
Porovnávání objektů - příklad	12
Metoda <code>hashCode</code>	13
Metoda <code>hashCode</code> - příklad	13
Bitové	14
Operátor podmíněného výrazu <code>?:</code>	14
Operátory typové konverze (přetypování)	14
Operátor zřetězení <code>+</code>	15
Priority operátorů a vytváření výrazů	15

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 "logery" - 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`
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/svet/chovatelstvi/psi/ChovatelPsuTest.java>]

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 1.1. 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;

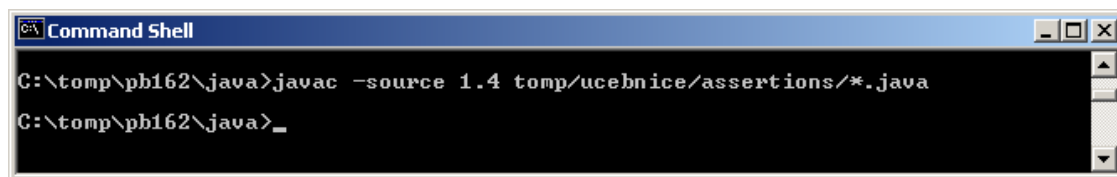
        citatel = c;
        jmenovatel = j;
    }

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

Ukázka použití assert (2)

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

Obrázek 1.2. Správný postup překladu AssertDemo

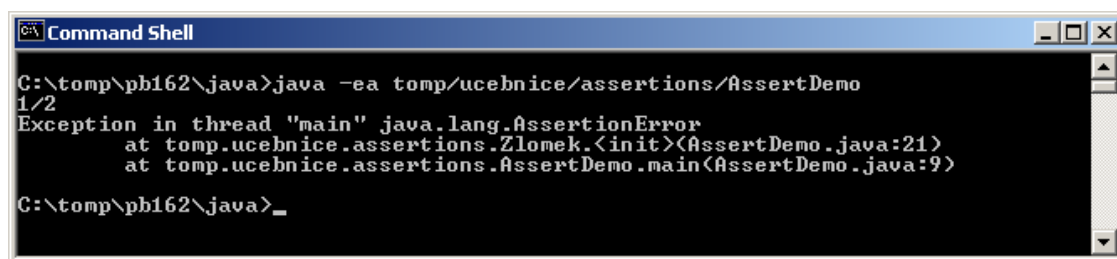


```
Command Shell
C:\tomp\pb162\java>javac -source 1.4 tomp/ucebnice/assertions/*.java
C:\tomp\pb162\java>_
```

Ukázka použití assert (3)

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

Obrázek 1.3. Spuštění AssertDemo s povolením assert

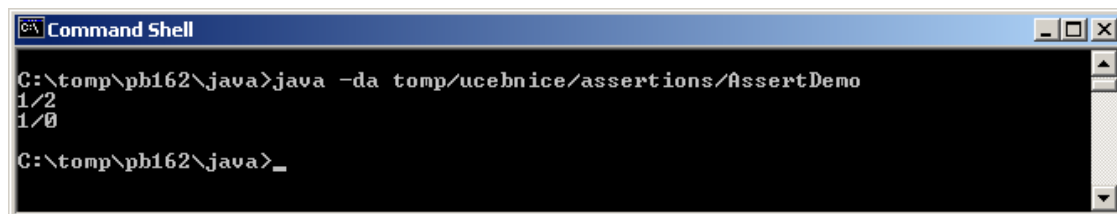


```
Command Shell
C:\tomp\pb162\java>java -ea tomp/ucebnice/assertions/AssertDemo
1/2
Exception in thread "main" java.lang.AssertionError
    at tomp.ucebnice.assertions.Zlomek.<init>(AssertDemo.java:21)
    at tomp.ucebnice.assertions.AssertDemo.main(AssertDemo.java:9)
C:\tomp\pb162\java>_
```

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 1.4. Spuštění AssertDemo bez povolení assert



```
Command Shell
C:\tomp\pb162\java>java -da tomp/ucebnice/assertions/AssertDemo
1/2
1/0
C:\tomp\pb162\java>_
```

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:

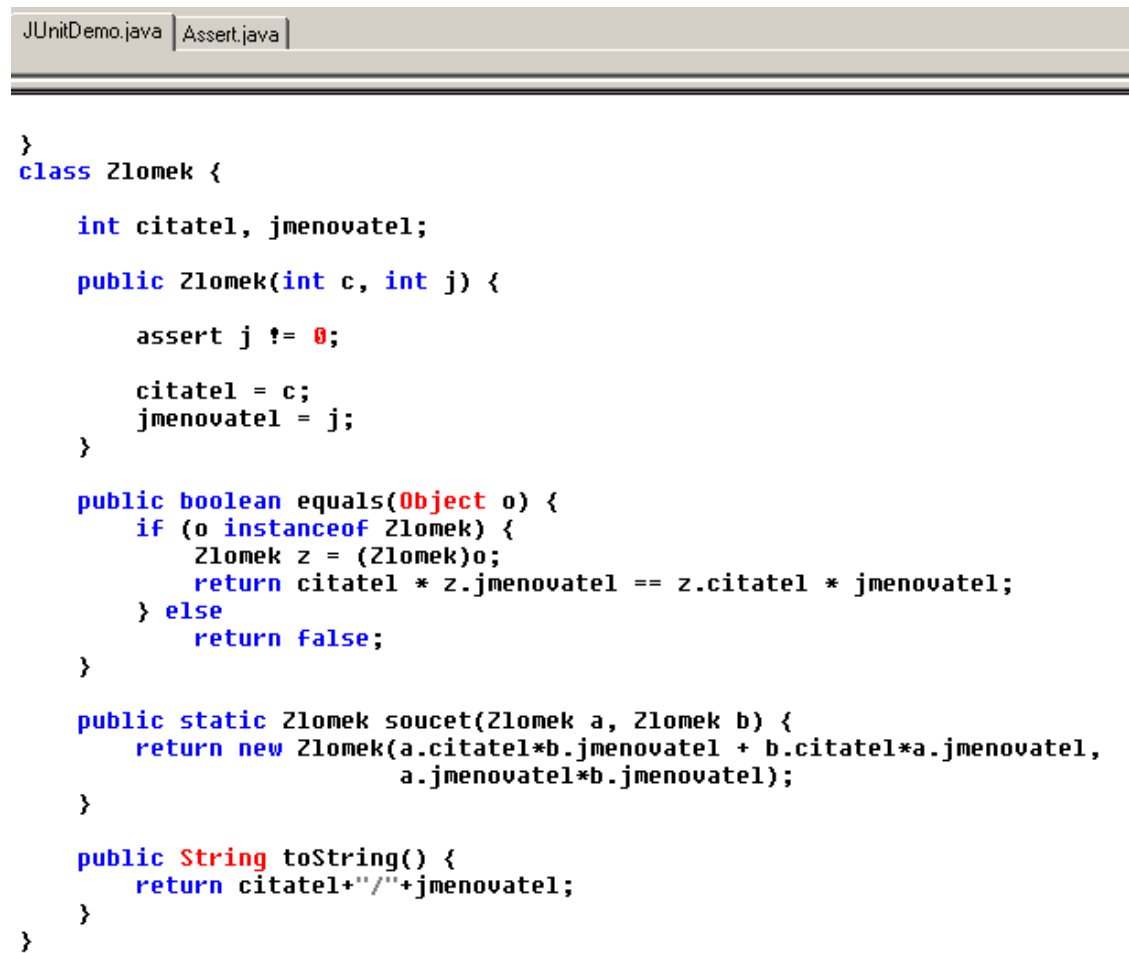
1. Stáhnout si z <http://junit.org> poslední (stačí binární) distribuci testovacího prostředí.
2. Nainstalovat JUnit (stačí rozbalit do samostatného adresáře).

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 (`testNeco`) 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 1.5. 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;
    }

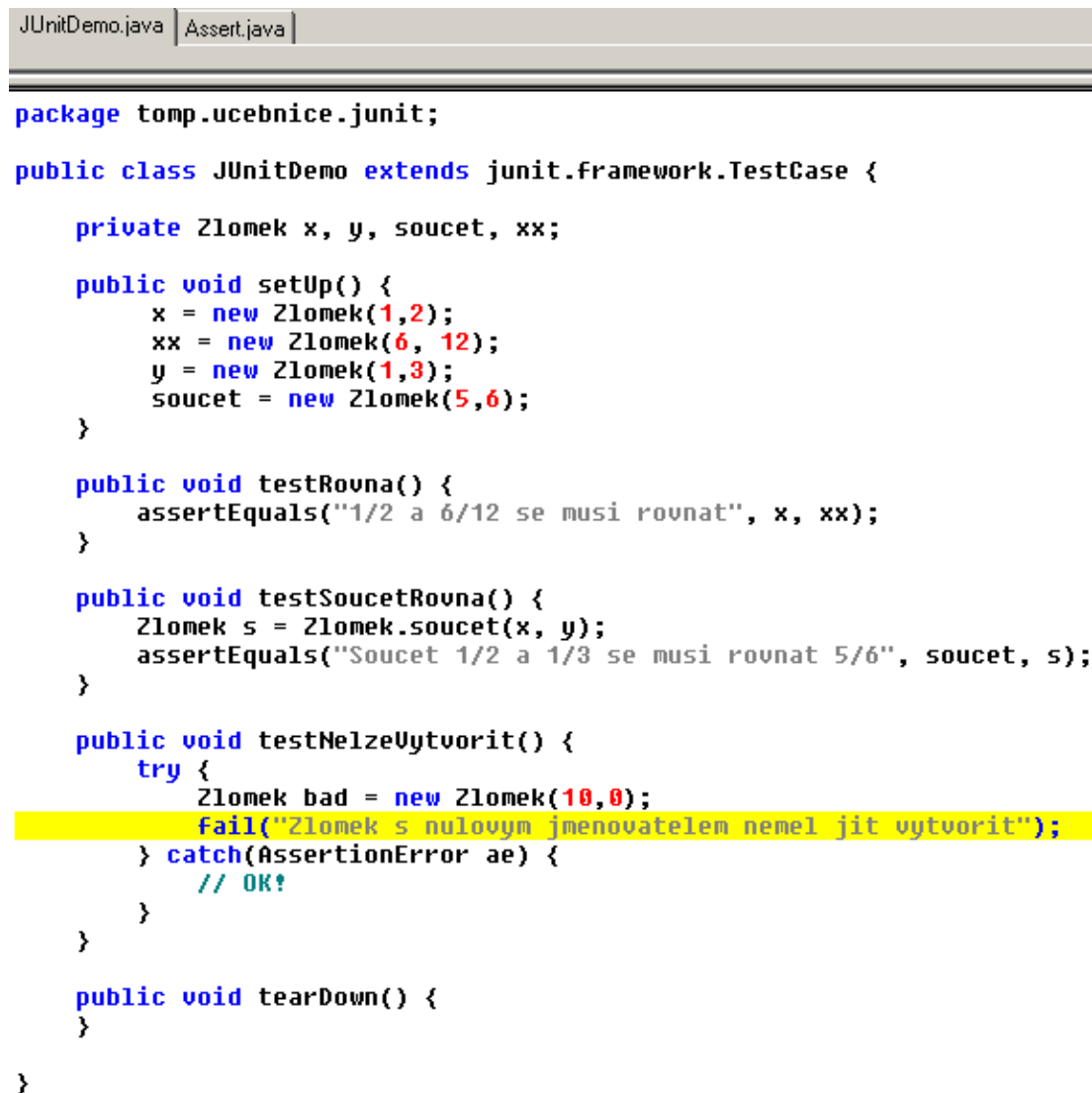
    public static Zlomek soucet(Zlomek a, Zlomek b) {
        return new Zlomek(a.citatel*b.jmenovatel + b.citatel*a.jmenovatel,
            a.jmenovatel*b.jmenovatel);
    }

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

Ukázka použití JUnit (2)

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

Obrázek 1.6. 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 1.7. Spouštění testovače nad testovací třídou JUnitDemo

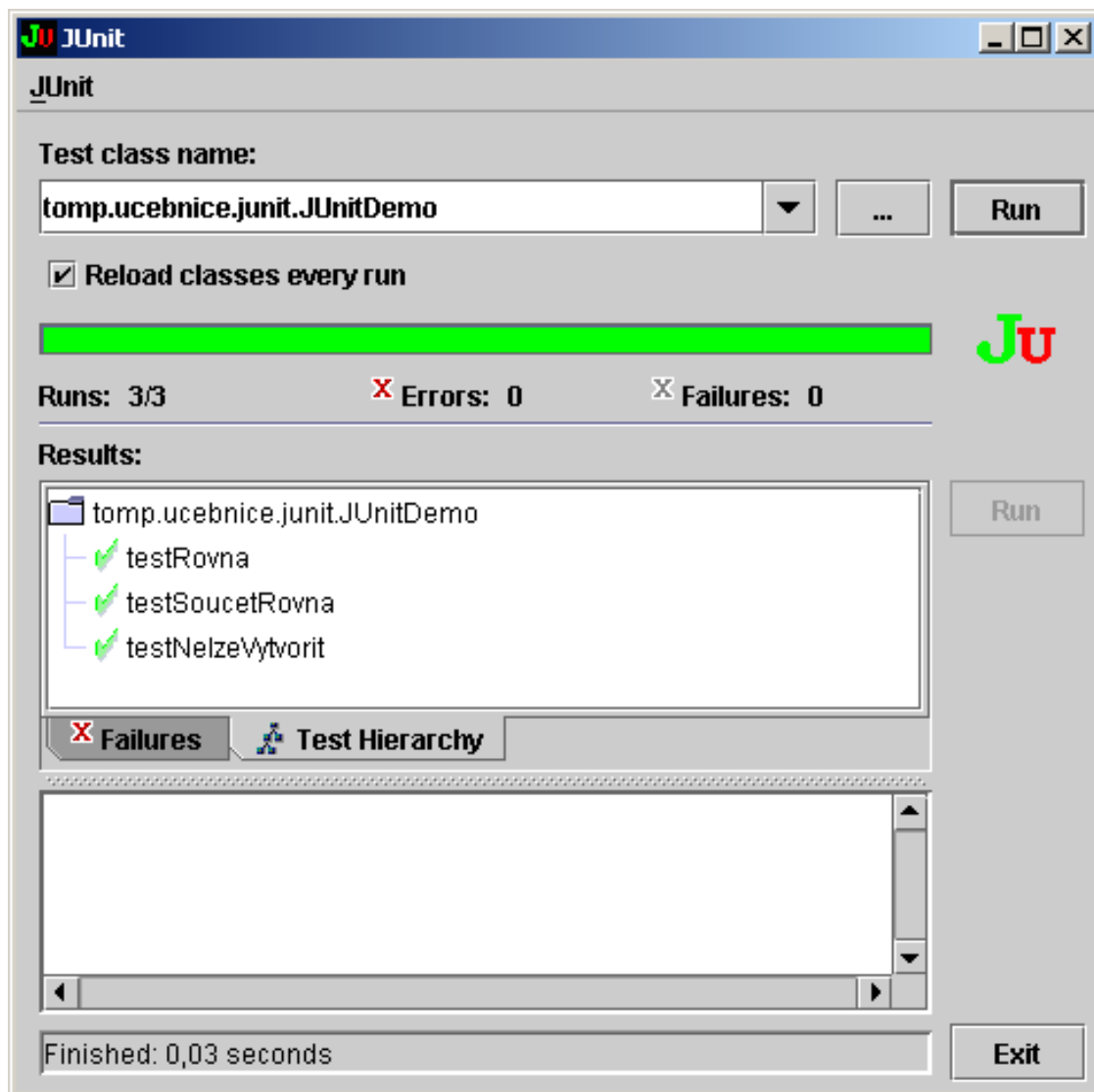


```
Command Shell - java -ea -classpath \devel\junit3.8.1\junit.jar; junit.swingui.TestRunner tomp.ucebn...
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\tomp\pb162\java>java -ea -classpath \devel\junit3.8.1\junit.jar;. junit.swing
ui.TestRunner tomp.ucebnice.junit.JUnitDemo
```

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

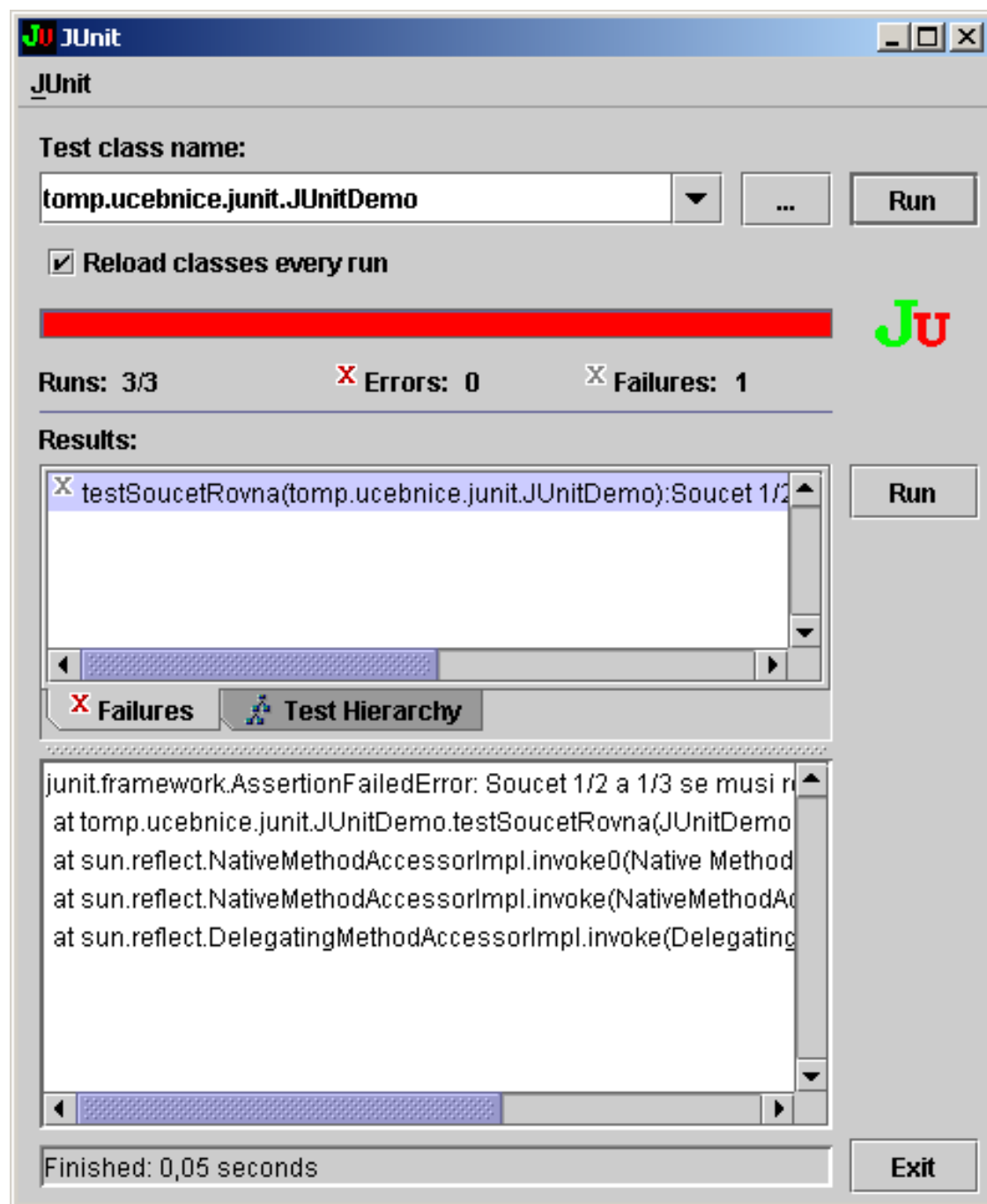
Obrázek 1.8. Testovač spouštějící třídu JUnitDemo



Ukázka použití JUnit (4)

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

Obrázek 1.9. 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 [<http://junit.org>]
- Java 1.4 logging API guide [<http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/>]
- Log4j homepage [<http://jakarta.apache.org/log4j/docs/index.html>]
- jass homepage [<http://csd.informatik.uni-oldenburg.de/~jass/>]
- úvodní materiálek [<http://www.inf.fu-berlin.de/lehre/SS01/VIS/Dokumente/Vortraege/junit.pdf>] k použití junit (v němčině, jako PDF)

Operátory a výrazy, porovnávání objektů

- Operátory v Javě: aritmetické, logické, relační, bitové
- Porovnávání primitivních hodnot a objektů, metody equals a hashCode
- Ternární operátor podmíněného výrazu
- Typové konverze

- Operátor zřetězení

Aritmetické

$+$, $-$, $*$, $/$ a $\%$ (zbytek po celočíselném dělení)

Logické

logické součiny (AND):

- $\&$ (*nepodmíněný* - vždy se vyhodnotí oba operandy),
- $\&\&$ (*podmíněný* - líné vyhodnocování - druhý operand se vyhodnotí, jen nelze-li o výsledku rozhodnout z hodnoty prvního)

logické součty (OR):

- $|$ (*nepodmíněný* - vždy se vyhodnotí oba operandy),
- $||$ (*podmíněný* - líné vyhodnocování - druhý operand se vyhodnotí, jen nelze-li o výsledku rozhodnout z hodnoty prvního)

negace (NOT):

- $!$

Relační (porovnávací)

Tyto lze použít na porovnávání primitivních hodnot:

- $<$, $<=$, $>=$, $>$

Test na rovnost/nerovnost lze použít na porovnávání primitivních hodnot i objektů:

- $==$, $!=$

Upozornění:

- pozor na porovnávání objektů: $==$ vrací true jen při rovnosti odkazů, tj. jsou-li objekty identické. Rovnost obsahu (tedy "rovnocennost") objektů se zjišťuje voláním metody `o1.equals(Object o2)`

- pozor na srovnávání floating-points čísel na rovnost: je třeba počítat s chybami zaokrouhlení; místo porovnání na přesnou rovnost raději použijeme jistou toleranci: $\text{abs}(\text{expected}-\text{actual}) < \text{delta}$

Porovnávání objektů

Použití ==

- Porovnáme-li dva objekty (tzn. odkazy na objekty) prostřednictvím operátoru == dostaneme rovnost jen v případě, jedná-li se o dva odkazy na tentýž objekt - tj. dva *totožné* objekty.
- Jedná-li se o dva obsahově stejné objekty existující samostatně, pak == vrátí false.

Chceme-li (intuitivně) chápat rovnost objektů podle obsahu, tj.

- dva objekty jsou *rovné* (*rovnocenné*, nikoli *totožné*), mají-li stejný obsah, pak
- musíme pro danou třídu překrýt metodu equals, která musí vrátit true, právě když se *obsah* výchozího a srovnávaného objektu rovná.
- Fungování equals lze srovnat s porovnáváním dvou databázových záznamů podle primárního klíče.
- Nepřekryjeme-li equals, funguje původní equals přísným způsobem, tj. *rovné si budou jen totožné objekty*.

Porovnávání objektů - příklad

Příklad: objekt třídy Clovek nese informace o člověku. Dva objekty položíme stejné (rovnocenné), nejsou-li stejná příjmení:

Obrázek 1.10. Dva lidi jsou stejní, mají-li stejná příjmení

```
public class Clovek implements Comparable {
    String jmeno, prijmeni;
    public Clovek (String j, String p) {
        jmeno = j;
        prijmeni = p;
    }
    public boolean equals(Object o) {
        if (o instanceof Clovek) {
            Clovek c = (Clovek)o;
            return prijmeni.equals(c.prijmeni);
        } else
            throw new IllegalArgumentException(
```

```
        "Nelze porovnat objekt typu Clovek s objektem jineho typu");
    }
}
```

Méně agresivní verze by nemusela při porovnávání s jiným objektem než Clovek vyhodit výjimku, pouze vrátit false.

Metoda hashCode

Jakmile u třídy překryjeme metodu equals, měli bychom současně překrýt objektů i metodu hashCode:

- hashCode vrací celé číslo (int) „co nejlépe“ charakterizující obsah objektu, tj.
- pro dva stejné (equals) objekty *musí vždy vrátit stejnou hodnotu*.
- Pro dva obsahově různé objekty by hashCode naopak měl vracet různé hodnoty (ale není to stoprocentně nezbytné a ani nemůže být vždy splněno). Metoda hashCode totiž nemůže vždy být prostá.

Metoda hashCode - příklad

V těle hashCode s oblibou „přehráváme“ (delegujeme) řešení na volání hashCode jednotlivých složek objektu - a to těch, které figurují v equals:

Obrázek 1.11. Třída Clovek s metodami equals a hashCode

```
public class Clovek implements Comparable {
    String jmeno, prijmeni;
    public Clovek (String j, String p) {
        jmeno = j;
        prijmeni = p;
    }
    public boolean equals(Object o) {
        if (o instanceof Clovek) {
            Clovek c = (Clovek)o;
            return prijmeni.equals(c.prijmeni);
        } else
            throw new IllegalArgumentException(
                "Nelze porovnat objekt typu Clovek s objektem jineho typu");
    }
    public int hashCode() {
        return prijmeni.hashCode();
    }
}
```

Bitové

Bitové:

- součin &
- součet |
- exkluzivní součet (XOR) ^ (znak "stříška")
- negace (bitwise-NOT) ~ (znak "tilda")

Posuny:

- vlevo << o stanovený počet bitů
- vpravo >> o stanovený počet bitů s respektováním znaménka
- vpravo >>> o stanovený počet bitů bez respektování znaménka

Dále viz např. Bitové operátory
[<http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/operatory/Bitove.java>]

Operátor podmíněného výrazu ? :

Jediný ternární operátor

Platí-li první operand (má hodnotu `true`) ->

- výsledkem je hodnota druhého operandu
- jinak je výsledkem hodnota třetího operandu

Typ prvního operandu musí být `boolean`, typy druhého a třetího musí být přiřaditelné do výsledku.

Operátory typové konverze (přetypování)

- Podobně jako v C/C++
- Píše se *(typ)hodnota*, např. *(Clovek)o*, kde *o* byla proměnná deklarovaná jako `Object`.
- Pro objektové typy se ve skutečnosti nejedná o žádnou konverzi spojenou se změnou obsahu objektu, nýbrž pouze o potvrzení, že běhový typ objektu je požadovaného typu - např. (viz výše) že *o* je typu `Clovek`.

- Naproti tomu u primitivních typů se jedná o úpravu hodnoty - např. `int` přetypujeme na `short` a „ořeže“ se tím rozsah.

Operátor zřetězení +

Výsledkem je vždy řetězec, ale argumenty mohou být i jiných typů, např.

sekvence `int i = 1; System.out.println("promenna i="+i);` je v pořádku

s řetězcovou konstantou se spojí řetězcová podoba dalších argumentů (např. čísla).

Pokud je argumentem zřetězení odkaz na objekt `o` ->

- je-li `o == null` -> použije se řetězec `null`
- je-li `o != null` -> použije se hodnota vrácená metodou `o.toString()` (tu lze překrýt a dosáhnout tak očekávaného řetězcového výstupu)

Priority operátorů a vytváření výrazů

nejvyšší prioritu má násobení, dělení, nejnižší přiřazení