Obsah:

Testování

Zadání příkladu

V této kapitole vytvoříme třídu, která bude převádět čísla na/z římská čísla. Římská čísla ze zapisují pomocí velkých písmen, které zastupují jednotlivá čísla:

I = 1
V = 5
X = 10
L = 50
C = 100
D = 500
M = 1000

Pro skládání římských čísel se používají následující pravidla:

1. Hodnoty písmen se až na výjimku popsanou v pravidle 2 sčítají, např. I je 1, II je 2, III jsou 3, VI je 6, VII je 7, VIII je 8.
2. Písmena vyjadřující mocniny 10 (I, X, C, M) se mohou za sebou opakovat maximálně 3. Pokud by měly být za sebou čtyři, vyjádří se pomocí odečtení hodnoty od nejbližší vyšší číslice. Číslo 4 nelze vyjádřit jako IIII, ale jako IV, 40 se zapisuje jako XL (50 bez 10), 41 se zapíše jako XLI, 44 jako XLIV (50 bez 10 a poté 5 bez 1). Obdobně 9 se zapíše jako IX, 90 jako XL, 900 jako CM.
3. Písmena vyjadřující 5, 50 a 500 se neopakují.
4. Písmena římského čísla se zapisují od největšího k nejmenšímu, na pořadí záleží. DC označuje hodnotu 600, číslo CD vyjadřuje úplně odlišnou hodnotu (400). CI znamená 101, zápis IC není přípustný (nelze odečítat 1 od 100, hodnota 99 se zapisuje jako XCIX – 100 bez 10 a k tomu 10 bez 1).
5. Pro každou hodnotu existuje pouze jeden správný zápis římskými číslicemi.
6. Pokud je zápis římského čísla správný, vyjadřuje právě jednu hodnotu.
7. Pomocí římských číslic lze zapsat pouze omezený rozsah hodnot – celá čísla od 1 do 3999

Vytvoření kostry třídy

Prvním krokem po zadání je navržení rozhraní třídy (popř. tříd), která bude zajišťovat potřebné funkce. V našem případě vytvoříme třídu Roman, která bude dvě statické metody toRoman a fromRoman

- metoda **toRoman** převede celočíselný parametr na řetězec (String), ve kterém bude číslo vyjádřeno pomocí římských číslic,
- pokud vstupní parametr metody **toRoman** bude mimo přípustný rozsah (1 až 3999), vyvolá výjimku IllegalArgumentException,
- metoda **fromRoman** převede hodnotu zadanou pomocí římských číslic ve vstupním parametru (String) na hodnotu typu int,
- pokud vstupní parametr **fromRoman** nebude obsahovat přípustnou hodnotu, bude vyvolána výjimka NumberFormatException.

Základní kostra třídy Roman bude vypadat následovně:

```
public class Roman {

   public static int fromRoman (String s) throws NumberFormatException {
       int vysl = 0;
       return vysl;
   }

   public static String toRoman (int i) throws IllegalArgumentException {
      String vysl="";
      return vysl;
   }

}
```

Napsání testů

Základní struktura třídy s testy:

```
import junit.framework.*;

public class RomanTest extends TestCase {                          // 1

   public RomanTest(String name) {                                 // 2
      super(name);
   }

   protected void setUp () {                                       // 3
   }

   protected void tearDown () {                                    // 3
   }

   //  testy .......

   public static Test suite() {                                    // 4
      return new TestSuite(RomanTest.class);
   }

   public static void main (String[] args) {                       // 5
      junit.textui.TestRunner.run (suite());
   }
}
```

komentáře:

1 potomek třídy TestCase

2 musí se předefinovat konstruktor

3 často se definují setUp() – počáteční nastavení (např. se vytvoří objekty či se naváže spojení s databází), a tearDown() pro úklid po testování,

4 vytvoření seznamu testů (suite), většinou se definuje statická metoda pojmenovaná suite, v rámci které se pro vytvoření seznamu testů používá obrat:[1]

```
return new TestSuite(RomanTest.class)
```

kterým se do seznamu testů zahrnou všechny metody začínající řetězcem „test“ ze třídy uvedené jako parametr, toto je preferovaný způsob vytvoření seznamu testů,
alternativou je postupné přidávání jednotlivých testů:

```
TestSuite seznamTestu = new TestSuite();
seznamTestu.addTest(new RomanTest(“testToRomanKnownValues”);
seznamTestu.addTest(new RomanTest(“testFromRomanKnownValues”);
...
return seznamTestu;
```

5 nepovinné vytvoření metody main, v rámci které se spouští seznam testů (který vrací metoda suite) v textovém rozhraní,

[1] Jméno suite se předpokládá při vyvolání testů.

Testy správnosti

Pro naše zadání si nadefinujeme dva testy správnosti výsledků. Jeden bude kontrolovat, zda probíhá správně převod z římských číslic na arabské a druhý bude kontrolovat správnost při opačném převodu. Pro testy správnosti potřebujeme mít vzorek dat se správnými výsledky.

Metoda assertEquals se používá pro porovnání očekávané hodnoty a skutečné hodnoty (v tomto pořadí).

Pro uložení dvojic si nadefinujeme pomocnou třídu Dvojice

```
class Dvojice {
   int arab;
   String roman;
   Dvojice (int arab, String roman) {
     this.arab=arab;
     this.roman=roman;
   }
};
```

A poté můžeme nadefinovat pole se správnými dvojicemi hodnot:

```
Dvojice knownValues [] =  {
   new Dvojice(1,"I"),
   new Dvojice(2,"II"),
   new Dvojice(3,"III"),
   new Dvojice(4,"IV"),
   new Dvojice(5,"V"),
    ......
```

Po nadefinování hodnot v této tabulce (samozřejmě se definuje pouze vzorek hodnot, viz kód celého testu uvedeného na straně 4) je snadné napsat obě metody pro kontrolu správnosti výsledků:

```
public void testToRomanKnownValues () {
   for (int i = 0; i< knownValues.length; i++) {
     String vysl = Roman.toRoman(knownValues[i].arab);
     assertEquals(knownValues[i].roman, vysl);
   }
}

public void testFromRomanKnownValues () {
   for (int i = 0; i< knownValues.length; i++) {
     int vysl = Roman.fromRoman(knownValues[i].roman);
     assertEquals(knownValues[i].arab, vysl);
   }
}
```

Testy výjimek

Vedle testů správnosti výsledků je potřeba též napsat testy, zda v případě chybných vstupních parametrů zahlásí metody chybu. Odchytávání výjimek se provádí ve v kódu testu, v případě, že se výjimka neobjeví, pomocí metody fail(String textChyby) se ohlásí chyba.

Následuje test metody toRoman, zda správně ohlásí výjimku.

```
public void testToRomanException () {
   int badValues [] = { 0, -1, 4000 };
   for (int i = 0; i < badValues.length; i++) {
     try {
        String vysl=Roman.toRoman(badValues[i]);
     }
     catch (IllegalArgumentException e) {
        continue;
     };
     fail("Expected IllegalArgumentException");
   };
```

```
   }
```

Testy metody fromRoman jsou uvedeny v kódu celé třídy na straně ???.

Testy vztahů/závislostí

Metody v rámci třídy se často ovlivňují a je potřeba též testovat vztahy mezi metodami. V našem případě se sice metody neovlivňují, ale platí mezi nimi ??? závislosti, tj.

i = fromRoman(toRoman(i));

pro hodnoty i v rozsahu od 1 do 3999. Můžeme proto vytvořit následující vztah:

```
public void testSanity () {
      for (int i=1; i < 4000; i++) {
        int vysl = Roman.fromRoman(Roman.toRoman(i));
        assertEquals(i, vysl);
      };
   }
```

Zdrojový text testu

Následuje úplný zdrojový text testu:

```
import junit.framework.*;

public class RomanTest extends TestCase {

   class Dvojice {
      int arab;
      String roman;
      Dvojice (int arab, String roman) {
        this.arab=arab;
        this.roman=roman;
      }
   };

   Dvojice knownValues [] =  {
      new Dvojice(1,"I"),
      new Dvojice(2,"II"),
      new Dvojice(3,"III"),
      new Dvojice(4,"IV"),
      new Dvojice(5,"V"),
      new Dvojice(6,"VI"),
      new Dvojice(7,"VII"),
      new Dvojice(8,"VIII"),
      new Dvojice(9,"IX"),
      new Dvojice(10,"X"),
      new Dvojice(50,"L"),
      new Dvojice(100,"C"),
      new Dvojice(500,"D"),
      new Dvojice(1000,"M"),
      new Dvojice(31,"XXXI"),
      new Dvojice(148,"CXLVIII"),
      new Dvojice(294,"CCXCIV"),
      new Dvojice(312,"CCCXII"),
      new Dvojice(421,"CDXXI"),
      new Dvojice(528,"DXXVIII"),
      new Dvojice(621,"DCXXI"),
      new Dvojice(782,"DCCLXXXII"),
      new Dvojice(870,"DCCCLXX"),
      new Dvojice(941,"CMXLI"),
      new Dvojice(1043,"MXLIII"),
      new Dvojice(1110,"MCX"),
      new Dvojice(1226,"MCCXXVI"),
      new Dvojice(1301,"MCCCI"),
      new Dvojice(1485,"MCDLXXXV"),
      new Dvojice(1509,"MDIX"),
```

```
   new Dvojice(1607,"MDCVII"),
   new Dvojice(2499,"MMCDXCIX"),
   new Dvojice(2574,"MMDLXXIV"),
   new Dvojice(2646,"MMDCXLVI"),
   new Dvojice(2723,"MMDCCXXIII"),
   new Dvojice(2892,"MMDCCCXCII"),
   new Dvojice(2975,"MMCMLXXV"),
   new Dvojice(3051,"MMMLI"),
   new Dvojice(3313,"MMMCCCXIII"),
   new Dvojice(3408,"MMMCDVIII"),
   new Dvojice(3501,"MMMDI"),
   new Dvojice(3610,"MMMDCX"),
   new Dvojice(3743,"MMMDCCXLIII"),
   new Dvojice(3844,"MMMDCCCXLIV"),
   new Dvojice(3888,"MMMDCCCLXXXVIII"),
   new Dvojice(3940,"MMMCMXL"),
   new Dvojice(3999,"MMMCMXCIX"),
};

public RomanTest(String name) {
   super(name);
}

protected void setUp () {
}

public void testToRomanKnownValues () {
   for (int i = 0; i< knownValues.length; i++) {
     String vysl = Roman.toRoman(knownValues[i].arab);
     assertEquals(knownValues[i].roman, vysl);
   }
}

public void testFromRomanKnownValues () {
   for (int i = 0; i< knownValues.length; i++) {
     int vysl = Roman.fromRoman(knownValues[i].roman);
     assertEquals(knownValues[i].arab, vysl);
   }
}

public void testToRomanException () {
   int badValues [] = { 0, -1, 4000 };
   for (int i = 0; i < badValues.length; i++) {
     try {
        String vysl=Roman.toRoman(badValues[i]);
     }
     catch (IllegalArgumentException e) {
        continue;
     }
     fail("Expected IllegalArgumentException");
   };
}

public void testIllegalCharacters () {
   String badValues [] = { "I ", "i", "a", "mm", "d", "MCi" };
   for (int i = 0; i < badValues.length; i++) {
     try {
        int vysl=Roman.fromRoman(badValues[i]);
     }
     catch (NumberFormatException e) {
        continue;
     }
     fail("Expected IllegalArgumentException for "+badValues[i]);
   };
}

public void testTooManyRepeatedNumerals () {
   String badValues [] = { "MMMM", "VV", "LL", "CCCC", "DD", "IIII" };
```

```
      for (int i = 0; i < badValues.length; i++) {
        try {
           int vysl=Roman.fromRoman(badValues[i]);
        }
        catch (NumberFormatException e) {
           continue;
        }
        fail("Expected IllegalArgumentException for "+badValues[i]);
      };
   }

   public void testRepeatedPairs () {
      String badValues [] = { "CMCM", "CDCD", "IVIV", "IXIX", "XLXL" };
      for (int i = 0; i < badValues.length; i++) {
        try {
           int vysl=Roman.fromRoman(badValues[i]);
        }
        catch (NumberFormatException e) {
           continue;
        }
        fail("Expected IllegalArgumentException for "+badValues[i]);
      };
   }

   public void testMalformedAntecedent () {
      String badValues [] = { "IIMMCC", "VX", "DCM", "CMM", "CMD", "IXIV", "MCMC",
                              "XCX", "IVI", "LM", "LD", "LC" };
      int vysl;
      for (int i = 0; i < badValues.length; i++) {
        try {
           vysl=Roman.fromRoman(badValues[i]);
        }
        catch (NumberFormatException e) {
           continue;
        }
        fail("Expected IllegalArgumentException for "+badValues[i] + " (" +vysl);
      };
   }

   public void testSanity () {
      for (int i=1; i < 4000; i++) {
        int vysl = Roman.fromRoman(Roman.toRoman(i));
        assertEquals(i, vysl);
      };
   }

   public static void main (String[] args) {
      junit.textui.TestRunner.run (suite());
   }
   public static Test suite() {
      return new TestSuite(RomanTest.class);
   }

};
```

Spuštění testů

Při spouštění testů si můžeme vybrat mezi textovým a grafickým rozhraním. Textové rozhraní lze použít vždy, lze ho též začlenit do dávek. Grafické rozhraní může mít uživatel spuštěné neustále a po každém přeložení testu či vlastní třídy lze stisknutím tlačítka znovu spustit test.

Textové rozhraní testů lze spustit následujícím příkazem:

```
java junit.textui.TestRunner RomanTest
```

Často se též vyvolání testů v textovém rozhraní zahrnuje do metody main třídy s testy, v našem případě metoda main vypadá následovně

```
   public static void main (String[] args) {
      junit.textui.TestRunner.run (suite());
   }
```

poté lze testy v textovém rozhraní vyvolat následovně:

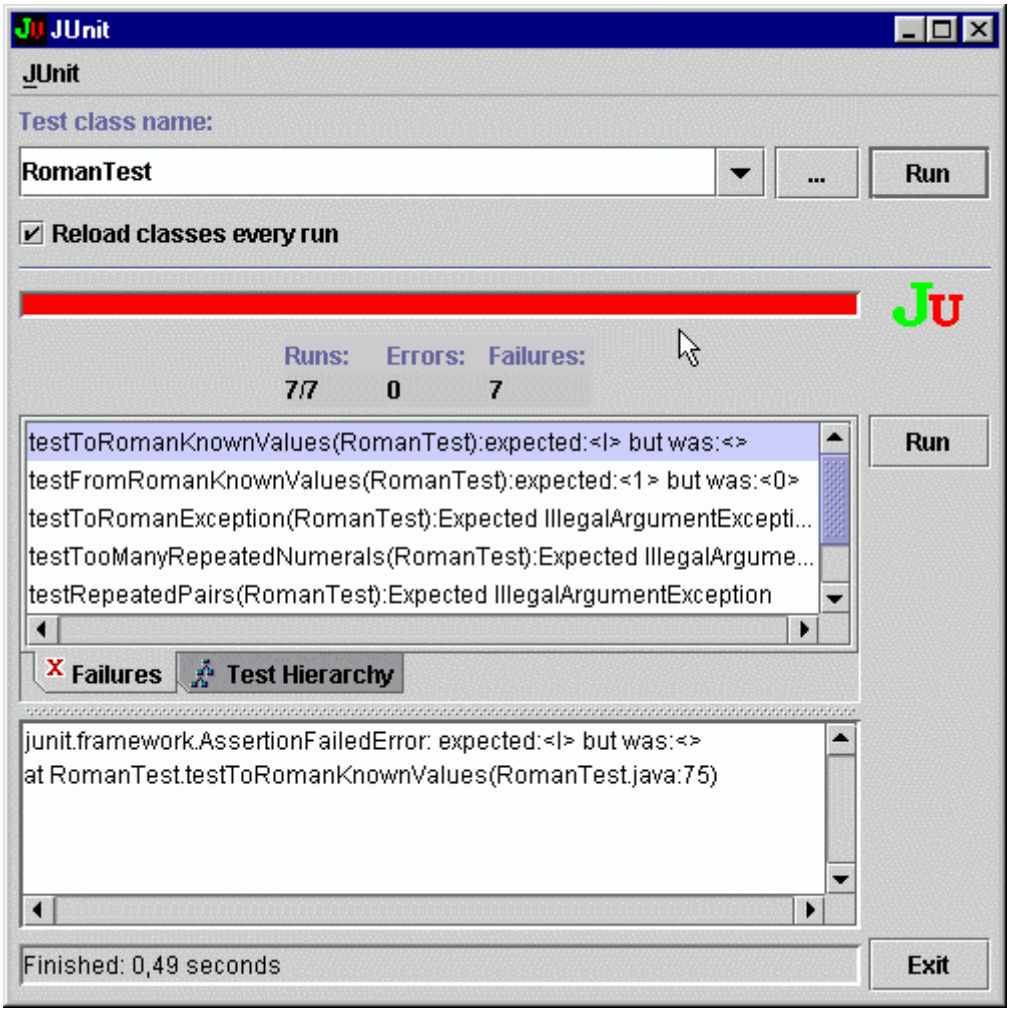
```
java RomanTest
```

Výstup v textovém rozhraní vypadá následovně (všechny testy jsou neúspěšné, neboť stále máme pouze kostru vlastní třídy):

```
.F.F.F.F.F.F.F
Time: 0
There were 7 failures:
1) testToRomanKnownValues(RomanTest)junit.framework.AssertionFailedError:
expected:<I> but was:<>
        at RomanTest.testToRomanKnownValues(RomanTest.java:75)
        at RomanTest.main(RomanTest.java:147)
2) testFromRomanKnownValues(RomanTest)junit.framework.AssertionFailedError:
expected:<1> but was:<0>
        at RomanTest.testFromRomanKnownValues(RomanTest.java:82)
        at RomanTest.main(RomanTest.java:147)
3) testToRomanException(RomanTest)junit.framework.AssertionFailedError: Expected
IllegalArgumentException
        at RomanTest.testToRomanException(RomanTest.java:95)
        at RomanTest.main(RomanTest.java:147)
4) testTooManyRepeatedNumerals(RomanTest)junit.framework.AssertionFailed-Error:
Expected IllegalArgumentException
        at RomanTest.testTooManyRepeatedNumerals(RomanTest.java:108)
        at RomanTest.main(RomanTest.java:147)
5) testRepeatedPairs(RomanTest)junit.framework.AssertionFailedError: Expected
IllegalArgumentException
        at RomanTest.testRepeatedPairs(RomanTest.java:121)
        at RomanTest.main(RomanTest.java:147)
6) testMalformedAntecedent(RomanTest)junit.framework.AssertionFailedError: Expected
IllegalArgumentException
        at RomanTest.testMalformedAntecedent(RomanTest.java:135)
        at RomanTest.main(RomanTest.java:147)
7) testSanity(RomanTest)junit.framework.AssertionFailedError: expected:<1> but
was:<0>
        at RomanTest.testSanity(RomanTest.java:142)
        at RomanTest.main(RomanTest.java:147)

FAILURES!!!
Tests run: 7,  Failures: 7,  Errors: 0
```

Pro spuštění testů se často používá grafické rozhraní, které poskytuje třída JUnit. Testy je možno spustit následujícím příkazem (opět s předpokladem, že je nastavena CLASSPATH na soubor junit.jar):
java junit.swingui.TestRunner RomanTest
(popř. lze spustit grafické rozhraní bez zadání konkrétní třídy s testy **java junit.swingui.TestRunner** a poté zadat třídu s testy v dialogovém okně). Vytvořené grafické rozhraní vypadá následovně (opět všechny testy jsou neúspěšné, neboť máme pouze kostru testované třídy):



Grafické rozhraní má výhodu v tom, že může být neustále spuštěné – po každém překladu testovaných tříd či tříd s testy se po stisku tlačítka Run spustí aktuální verze testů i testovaných tříd.

Vlastní třída a testování

Nyní je čas začít psát vlastní třídu (v praxi se většinou střídá psaní testů s psaním vlastního kódu, náš případ je poměrně jednoduchý a tudíž i z pedagogických důvodů jsme popsali nejdříve všechny testy). Začneme třídu toRoman, která je jednodušší (je rychle vidět pokrok v počtu úspěšných testů):

Nejdříve doplníme test přípustného intervalu hodnot:

```
public static String toRoman  (int i) throws IllegalArgumentException {
   if ((i > 3999)||(i<= 0))throw new IllegalArgumentException();
   else {
      return vysl;
   }
}
```

Program přeložíme a spustíme test – zjistíme, že jeden test již úspěšně prošel. Nyní do metody toRoman doplníme konverzi na římské číslice. Nejdříve si vytvoříme pomocné pole, které bude obsahovat dvojice hodnot, ze kterých lze skládat římské číslice. Nejsou zde uvedeny pouze jednotlivá písmena, ale i dvojice znaků, které mají speciální význam (4, 9, 40 ...). Pro uložení dvojic si nadefinujeme vnitřní třídu Dvojice.

```
static class Dvojice {
   int arab;
   String roman;
   Dvojice (int arab, String roman) {
      this.arab=arab;
```

```
      this.roman=roman;
   }
};

static Dvojice tabulka [] = {
   new Dvojice (1000, "M" ),
   new Dvojice (900, "CM" ),
   new Dvojice (500, "D"  ),
   new Dvojice (400, "CD" ),
   new Dvojice (100, "C"  ),
   new Dvojice ( 90, "XC" ),
   new Dvojice ( 50, "L"  ),
   new Dvojice ( 40, "XL" ),
   new Dvojice ( 10, "X"  ),
   new Dvojice (  9, "IX" ),
   new Dvojice (  5, "V"  ),
   new Dvojice (  4, "IV" ),
   new Dvojice (  1, "I"  )
};
```

Nyní je již poměrně snadné napsat třídu toRoman:

```
public static String toRoman  (int i) throws IllegalArgumentException {
   if ((i > 3999)||(i<= 0))throw new IllegalArgumentException();
   else {
      int cislo = i;
      String vysl="";
      while (cislo > 0) {
         for (int j=0; j < tabulka.length; j++) {
            if (cislo >= tabulka[j].arab) {
               vysl=vysl+tabulka[j].roman;
               cislo=cislo-tabulka[j].arab;
               break;
            };
         };
      };
      return vysl;
   }
}
```

Pokud nyní spustíme testy, budou již úspěšné dva. Metody fromRoman si udělejte sami za domácí úkol.

Úkoly:
a) Napište metodu fromRoman.
b) Při psaní římských číslic se někdy připouští možnost psaní čtyř písmen M za sebou, tj je možno pomocí římských číslic zapsat čísla od 1 do 4999. Upravte testy a třídu Roman tak, aby toto umožňovala.

Možnosti třídy JUnit

Shrňme si základní pravidla pro psaní testů pomocí třídy JUnit:
- pro každou testovanou třídu se obvykle vytváří třída s testy
- třída s testy má obvykle v názvu slovo Test
- třída s testy je potomkem třídy TestCase, musí předefinovat konstruktor s parametrem typu String a musí obsahovat statickou metodu public static Test suite() se seznamem testů,
- každý test je samostatná metoda a obvykle začíná slovem test, v rámci těchto metod se pro vyhodnocení výsledku obvykle používá metoda assertEquals(), v některých případech je výhodné použít metodu fail() pro vyjádření neúspěchu testu,
- pro nastavení počátečního stavu před testem je možné ve třídě s testy nadefinovat metodu setUp() a pro úklid metodu tearDown(),
- pokud máme více tříd s testy, je možné spojit testy v rámci jedné hlavní třídy, která se obvykle nazývá AllTests a obsahuje statickou metodu suite, v rámci které se zahrnou testy z ostatních tříd s testy. Tato třída by mohla vypadat následovně (zahrnuje testy ze dvou testovacích tříd):

```
import junit.framework.*;
```

```
/**
 * TestSuite that runs all the tests
 *
 */
public class AllTests {

	public static void main (String[] args) {
		junit.textui.TestRunner.run (suite());
	}
	public static Test suite ( ) {
		TestSuite suite= new TestSuite("All JUnit Tests");
		suite.addTest(VectorTest.suite());
		suite.addTest(ArrayTest.suite());
		return suite;
	}
}
```

- u větších projektů je zvykem pro každý balík (package) vytvořit speciální balík, který obsahuje testy tříd prvního balíku. Např. pokud máte balík myapp.util, vytvoří se další balík (adresář) myapp.utiltest, který bude obsahovat testy jednotlivých tříd z myapp.util a poté třídu AllTests, která bude odkazovat na všechny testy v tomto balíku. V některých případech může být však výhodnější umístit třídy s testy přímo do balíku s vlastním kódem (důvodem je nutnost testovat metody/objekty, které nejsou public, ale viditelné pouze v rámci balíku).
- při překladu testů i při spouštění testů je potřeba mít nastavenu cestu CLASSPATH na archiv junit.jar,
- pro spouštění testů u větších úloh je výhodné použít program ant a v něm si nadefinovat příslušné úlohy,

Pro JUnit existují různá rozšíření, která umožňují lépe využít možnosti této třídy. Jejich přehled je k dispozici na http://www.junit.org/. Zde jsou uvedeny pouze některé:

- třída JUnitPerf umožňuje testovat výkonnost Vámi vytvářených tříd. Třída je postavená jako dekorátor a umožňuje testovat rychlost třídy (tj. zda proběhne za určený čas) a zátěž třídy (tj. jak dlouho bude trvat běh třídy v případě, že bude testovaná třída bude spuštěna n-krát).
- třída Jester vyhledává kód, který není pokrytý testy,
- třída unittestgen generuje kostry testů v případě, že máme již rozpracovaný projekt a chceme doplnit testy,
- třída NoUnit zjišťuje u jednotlivých tříd a metod, jak jsou pokryté testy. Má hezké výstupy.