UNI 6

Junit – Unit J

Overview -- Gambaran

Kerangka kerja pengujian unit JUnit adalah implementasi referensi xUnit. Seperti namanya menyiratkan, ini dikembangkan dan digunakan dengan Java. Ini tidak diragukan lagi yang paling luas kerangka kerja yang digunakan, diperluas, dan dibahas untuk pengujian unit perangkat lunak saat ini. **JUnit adalah** dasar untuk alat pengujian unit yang lebih khusus, termasuk Cactus, Jester, JUnit-Perf, dan banyak lagi, dan terintegrasi erat dengan yang lain, seperti Ant. Komunitas besar pengguna JUnit berarti bahwa ini adalah dasar bagi banyak ide baru dan perkembangan teknologi pengujian unit.

Arsitektur xUnit generik, yang dijelaskan di Bab 3, mencerminkan arsitektur JUnit. Java dirancang dari bawah ke atas sebagai bahasa berorientasi objek yang sebenarnya, yang digabungkan banyak fitur modern seperti kelas abstrak murni, refleksi objek, dan penanganan pengecualian asli. JUnit memanfaatkan fitur-fitur ini sepenuhnya.

Tujuan JUnit adalah menyediakan kerangka kerja untuk membangun dan menjalankan pengujian unit. Distribusi JUnit juga merupakan contoh yang bagus dari produk perangkat lunak yang sederhana dan solid dibuat menggunakan pengembangan yang didorong oleh pengujian. Memeriksa kode sumbernya bersifat instruktif.

JUnit adalah perangkat lunak sumber terbuka yang dirilis di bawah Lisensi Publik Umum. Ini lisensi membebaskan semua kontributor dari kewajiban atau tanggung jawab apa pun atas kode, dan membuat pengguna bebas untuk mendistribusikan, menyalin, mengubah, menjual, dan sebaliknya. Untuk detailnya, lihat http://www.opensource.org.

Sumber definitif untuk segala sesuatu yang berhubungan dengan JUnit adalah http://www.junit.org. Itu Informasi yang diberikan dalam buku ini didasarkan pada JUnit 3.8.1, versi JUnit as saat ini tulisan ini.

Architecture -- Arsitektur

JUnit berisi sekitar 75 kelas bernama ditambah sejumlah kelas dan antarmuka dalam. Ini diatur ke dalam paket-paket, seperti yang ditunjukkan pada Gambar 6-1.

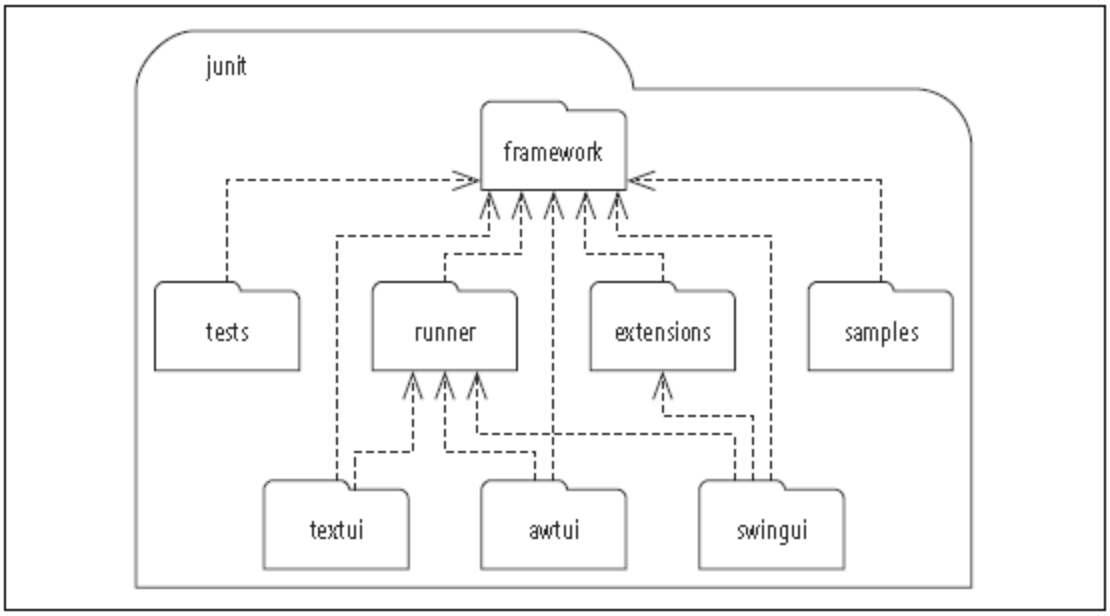
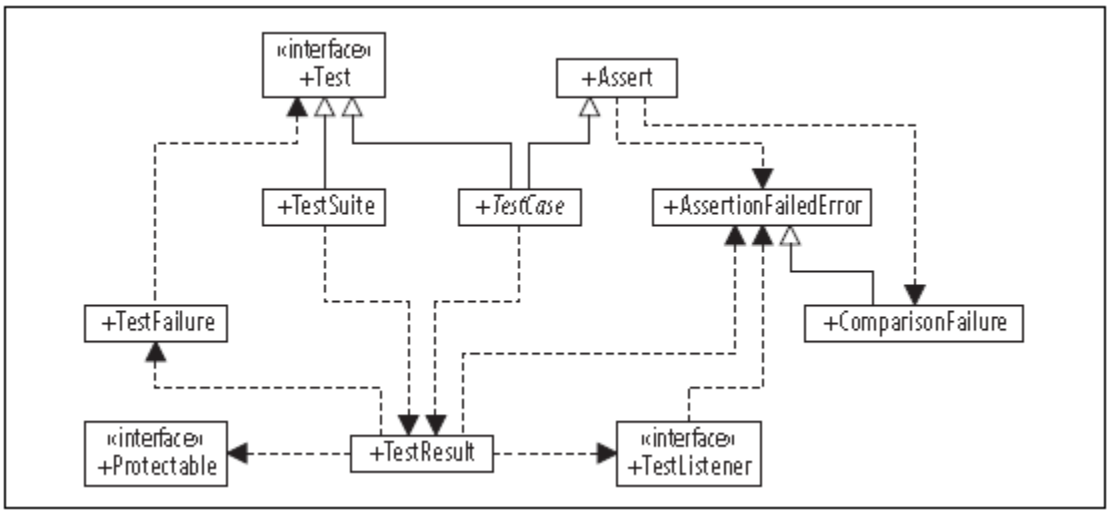


Figure 6-1. The JUnit packages and their import dependencies

Package junit.framework mewakili fungsionalitas inti, dan fondasinya yang menjadi dasar pengujian unit. Sebagian besar kode lainnya relatif kompleks Swing, AWT, dan Teks User Interface (UI), package junit.samples (berisi contoh pengujian unit), dan paket junit.tests (berisi milik JUnit sendiri tes unit). Pengembang JUnit "makan makanan anjing mereka sendiri" dengan menyediakan satu set lengkap pengujian unit untuk semua fungsinya.

Arsitektur kelas untuk paket junit.framework ditunjukkan pada Gambar 6-2.



Arsitektur junit.framework mengikuti model arsitektur xUnit generik

dijelaskan dalam Bab 3. Secara khusus, perhatikan elemen arsitektur kunci, Test antarmuka, yang diimplementasikan oleh kelas TestCase dan TestSuite. Kelas abstrak TestCase adalah induk dari semua kelas pengujian unit.

Seperti yang dijelaskan di Bab 3, kelas kunci yang digunakan saat membangun pengujian unit adalah TestCase, TestSuite, dan TestRunners. Lampiran B adalah referensi kelas rinci untuk package junit.framework.

Usage -- Pemakaian

Bagian ini menyajikan gambaran umum singkat tentang penggunaan JUnit dasar. Contoh xUnit di bab sebelumnya memberikan ulasan yang lebih detail tentang cara menggunakan JUnit.

Kelas pengujian di JUnit adalah subkelas dari TestCase. Tes dapat dibangun di sejumlah cara, tetapi pendekatan konvensional dijelaskan di sini. Nama kelas tes dimulai dengan nama objek yang diuji dan diakhiri dengan Test. Kelas tes berisi metode pengujian terpisah untuk setiap perilaku yang diuji. Metode pengujian diberi nama dimulai dengan tes.

Kondisi pengujian diperiksa dengan metode test assert. Tes menegaskan hasil dalam keberhasilan tes atau kegagalan. Jika pernyataan gagal, metode pengujian segera kembali. Jika berhasil, eksekusi metode pengujian terus berlanjut. Karena metode pengujian seharusnya hanya menguji a perilaku tunggal, dalam banyak kasus, setiap metode pengujian hanya boleh berisi satu pernyataan pengujian.

Jika ada objek yang dibagikan oleh metode pengujian, mereka harus diinisialisasi di metode setUp() dan dimusnahkan di tearDown(). Metode ini dipanggil sebelumnya dan setelah setiap panggilan metode pengujian, secara efektif membuat ulang perlengkapan pengujian untuk setiap pengujian, sehingga memberikan isolasi uji.

Contoh 6-1 menunjukkan kelas pengujian yang dibuat mengikuti model ini, yang disebut LibraryTest, itu (secara alami) menguji Perpustakaan kelas.

Example 6-1. The test class LibraryTest

LibraryTest.java

```
import junit.framework.*;
import java.util.*;

public class LibraryTest extends TestCase {

   private Library library;

   public void setUp( ) throws Exception {
      library = new Library( );
      library.addBook(new Book( "Cosmos", "Carl Sagan" ));
      library.addBook(new Book( "Contact", "Carl Sagan" ));
      library.addBook(new Book( "Solaris", "Stanislaw Lem" ));
      library.addBook(new Book( "American Beauty", "Allen M Steele" ));
      library.addBook(new Book( "American Beauty", "Edna Ferber" ));
   }
```

```
    public void tearDown( ) {
        library.empty( );
        library = null;
    }

    public void testGetBooksByTitle( ) {
        Vector books = library.getBooksByTitle( "American Beauty" );
        assertEquals( "wrong number of books found", 2, books.size( ) );
    }

    public void testGetBooksByAuthor( ) {
        Vector books = library.getBooksByAuthor( "Carl Sagan" );
        assertEquals( "2 books not found", 2, books.size( ) );
    }

    public void testEmpty( ) {
        library.empty( );
        assertEquals( "library not empty", 0, library.getNumBooks( ) );
    }
}
```

LibraryTest adalah perlengkapan uji karena ada beberapa metode pengujian yang berbagi file objek — dalam hal ini, instance Library. Perpustakaan dibuat dan dimuat dengan file set Buku di setUp () dan dikosongkan serta dibuang di tearDown().

Setiap metode pengujian memverifikasi perilaku berbeda dari kelas Library dengan satu pengujian menegaskan. Meskipun pengujian dapat mengubah Library, seperti yang ditunjukkan oleh testEmpty(), pengujian tersebut fungsionalitas fixture menjamin bahwa setiap pengujian berjalan di fixture yang bersih, tanpa ketergantungan pada hasil lainnya.

Pengujian dijalankan menggunakan salah satu alat TestRunner yang disediakan dengan JUnit. Yang paling sederhana dan yang paling mudah otomatis adalah TextTestRunner. Contoh 6-2 menunjukkan menggunakan TextTestRunner untuk menjalankan LibraryTest.

Example 6-2. Running LibraryTest with the TextTestRunner

```
$ java junit.textui.TestRunner LibraryTest
.........
Time: 0.01

OK (3 tests)
```

Seringkali berguna untuk menggabungkan beberapa pengujian sehingga dapat dijalankan bersama. Kelas TestSuite digunakan untuk menampung kumpulan tes. Contoh 6-3 menunjukkan sebuah kelas, LibraryTests, yang membuat TestSuite yang berisi sejumlah kelas pengujian. Itu static method suite() membuat dan mengembalikan TestSuite.

LibraryTests.java

```
import junit.framework.*;

public class LibraryTests extends TestSuite {

   public static Test suite( ) {
      TestSuite suite = new TestSuite( );
      suite.addTest(new TestSuite(BookTest.class));
      suite.addTest(new TestSuite(LibraryTest.class));
      suite.addTest(new TestSuite(LibraryDBTest.class));
      suite.addTest(new TestSuite(LibraryPerfTest.class));
      return suite;
   }
}
```

Saat LibraryTests dijalankan menggunakan TextTestRunner, semua metode pengujian di setiap pengujian kelas ditemukan dan dijalankan, seperti yang ditunjukkan pada Contoh 6-4.

Example 6-4. Running the LibraryTests test suite

```
$ java junit.textui.TestRunner LibraryTests
.................
Time: 0.851

OK (17 tests)
```

GUI TestRunner dapat digunakan sebagai pengganti TextTestRunner. Versi Swing dari TestRunner dipanggil untuk LibraryTest seperti yang ditunjukkan:

```
$ java junit.swingui.TestRunner LibraryTest
```

Gambar 6-3 menunjukkan hasil setelah GUI TestRunner menjalankan LibraryTest.

Test Assert Methods -- Uji Metode Penegasan

Berbagai metode pernyataan pengujian disediakan oleh JUnit. Mereka diterapkan sebagai publik metode statis kelas Assert, yang merupakan kelas induk dari TestCase. Jadi, setiap kelas pengujian mewarisi metode ini.

Metode pengujian yang paling umum adalah assertTrue(), yang berhasil atau gagal berdasarkan nilai argumen Boolean. Metode test assert lainnya dikhususkan versi assertTrue() yang menangani tipe kondisi pengujian tertentu. Untuk Misalnya, pernyataan pernyataan uji berikut ini setara:

```
assertTrue( book.title.equals("Cosmos") );
assertEquals( "Cosmos", book.title );
```

These statements are equivalent as well:

```
assertTrue( false )
fail( )
```

Metode pengujian khusus berguna karena mereka menghemat upaya pengkodean lebih mudah dibaca, dan memungkinkan pelaporan hasil yang lebih spesifik.

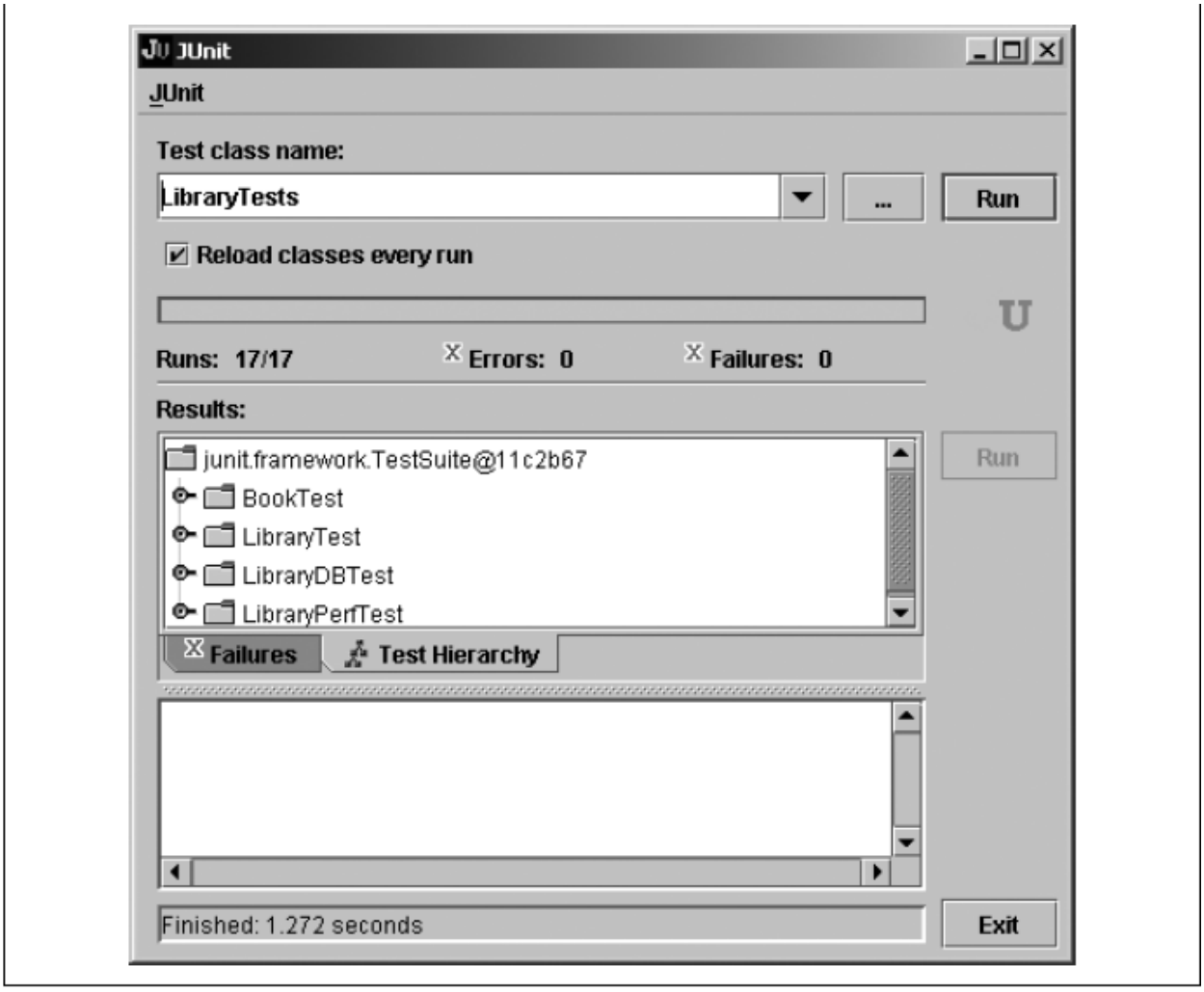


Figure 6-3. The Swing TestRunner after LibraryTest is run

Semua metode assert memiliki dua varian, yang menggunakan pesan String sebagai yang pertama argumen, dan yang tidak. Pesan tersebut memungkinkan Anda untuk memberikan penjelasan yang lebih rinci deskripsi kegagalan pernyataan.

Metode assertEquals() membandingkan nilai dari dua argumen. Mereka berasumsi bahwa nilai pertama adalah nilai yang benar atau "diharapkan", yang kedua "sebenarnya" nilai harus

dibandingkan. Metode ini akan berfungsi jika argumen dibalik, tetapi pesan kegagalan akan menyesatkan.

Metode pernyataan JUnit dijelaskan dalam daftar berikut ini:

`static void assertTrue(boolean condition)`
`static void assertTrue(message, boolean condition)`
The assertTrue assertion passes if condition is true. This is the most generic type of assertion.

`static void assertFalse(boolean condition)`
`static void assertFalse(message, boolean condition)`
Test passes if condition is false.

`static void assertEquals(expected, actual)`
`static void assertEquals(message, expected, actual)`
Test passes if expected and actual are equal. Versions of this method exist to compare arguments of type boolean, byte, char, int, long, Object, short, or String.

`static void assertEquals(expected, actual, delta)`
`static void assertEquals(message, expected, actual, delta)`
Asserts equality of two values within a tolerance of delta. A delta of 0.0 tests exact equality. Versions of this method exist to handle arguments of type double or float.

`static void assertNotNull(Object object)`
`static void assertNotNull(message, Object object)`
Test passes if Object is not null.

`static void assertNull(Object object)`
`static void assertNull(message, Object object)`
Test passes if Object is null.

`static void assertNotSame(Object expected, Object actual)`
`static void assertNotSame(message, Object expected, Object actual)`
Test passes if the two Objects are not the same Object, as determined by the == operator.

`static void assertSame(Object expected, Object actual)`
`static void assertSame(message, Object expected, Object actual)`
Test passes if the two Objects are the same Object, as determined by the == operator.

`static void fail()`
`static void fail(message)`
Test that always fails. It is equivalent to assertTrue(false).