Extending FEST-Assert with Custom Assertions

FEST Assertions Module has moved to Github !

Check this page :https://github.com/alexruiz/fest-assert-2.x/wiki

The documentation below is for Fest 1.x which is no more maintained, we are focusing our effort to the 2.x version !

FEST-Assert can also "decorate" user-defined assertion classes with the prefix `assertThat`, resulting in improved readability of test code.

The only condition is that user-defined assertion classes must extend `org.fest.assertions.GenericAssert` and set in first generic parameter the custom assertion class and in second the class you want to make assertions on.

The following example demonstrates a custom assertion class `CharacterAssert` that verifies the name of a `Character`, code is available here :

- Character.java : Character class definition
- CharacterAssert.java : Character custom assertions definition
- CustomFestAssertExamples.java : Character custom assertions usage
- Race.java : Race class definition

```
import org.fest.assertions.*;

public class CharacterAssert extends GenericAssert<CharacterAssert, Character> {

  /**
   * Creates a new </code>{@link CharacterAssert}</code> to make assertions on actual
Character.
   * @param actual the Character we want to make assertions on.
   */
  public CharacterAssert(Character actual) {
    super(CharacterAssert.class, actual);
  }

  /**
   * an entry point for CharacterAssert to follow Fest standard
<code>assertThat()</code> statements.<br>
   * With a static import, one's can write directly :
<code>assertThat(frodo).hasName("Frodo");</code>
   * @param actual the Character we want to make assertions on.
   * @return a new </code>{@link CharacterAssert}</code>
   */
  public static CharacterAssert assertThat(Character actual) {
    return new CharacterAssert(actual);
  }

  /**
   * Verifies that the actual Character's name is equal to the given one.
   * @param name the given name to compare the actual Character's name to.
   * @return this assertion object.
   * @throws AssertionError - if the actual Character's name is not equal to the given
one.
   */
  public CharacterAssert hasName(String name) {
    // check that actual Character we want to make assertions on is not null.
    isNotNull();

    // check that actual Character's name is equal to the given name (we assume here
that Character's name is not null).
    // we overrides the default error message with a more explicit one
    String errorMessage = String.format("Expected character's name to be <%s> but was
<%s>",name, actual.getName());

Assertions.assertThat(actual.getName()).overridingErrorMessage(errorMessage).isEqualTo
(name);

    // we could have used other basic assertions, for example :

Assertions.assertThat(actual.getName().equals(name)).overridingErrorMessage(errorMessa
ge).isTrue();

    // return the current assertion for method chaining
    return this;
  }
}
```

Let's play with our Character custom assertions !

```
import static examples.CharacterAssert.assertThat;
import static org.fest.assertions.Assertions.assertThat;
import org.junit.Test;

public class CustomFestAssertExamples {

  private final Race hobbit = new Race("Hobbit", false);

  @Test
  public void succesful_custom_assertion_example() {
    Character frodo = new Character("Frodo", 33, hobbit);
    Character merry = new Character("Merry", 36, hobbit);
    // custom assertion : assertThat is resolved from CharacterAssert static import
    assertThat(frodo).hasName("Frodo");  //
    // Fest standard assertion : assertThat is resolved from
org.fest.assertions.Assertions static import
    assertThat(frodo.getAge()).isEqualTo(33);
    // generic assertion like 'isNotEqualTo' are available for CharacterAssert
    assertThat(frodo).isNotEqualTo(merry);
  }

  @Test
  public void failed_custom_assertion_example() {
    Character sam = new Character("Merry", 38, hobbit); // oops wrong name !
    try {
      // custom assertion : assertThat is resolved from CharacterAssert static import
      assertThat(sam).hasName("Sam");
    } catch (AssertionError e) {
      // see that error message is explicit
      assertThat(e).hasMessage("Expected character's name to be <Sam> but was
<Merry>");
    }
  }
}
```

The previous example shows few interesting things :

- it is easy to call your custom assertions in Fest standard way with *assertThat()*
- you keep taking advantage of Fest standard assertions.

Old documentation without self type (to use before Fest 1.4).

FEST-Assert can also "decorate" user-defined assertion classes with the prefix `assertThat`, resulting in improved readability of test code. The only condition is that user-defined assertion classes must implement the marker interface `org.fest.assertions.AssertExtension`.

The following example demonstrates an assertion class that verifies the state of a `ServerSocket`:

```
public class ServerSocketAssertion implements AssertExtension {
  private final ServerSocket socket;

  public ServerSocketAssertion(ServerSocket socket) {
    this.socket = socket;
  }

  public ServerSocketAssert isConnectedTo(int port) {
    assertThat(socket.isBound()).isTrue();
    assertThat(socket.getLocalPort()).isEqualTo(port);
    assertThat(socket.isClosed()).isFalse();
    return this;
  }
}
```

We can decorate that assertion class with `assertThat`:

```
ServerSocketAssertion socket =
    new ServerSocketAssertion(server.getSocket());
assertThat(socket).isConnectedTo(2000);
```