Mean Bean: Rapidly test Java object equals, hashCode & property methods

Learn how to rapidly and reliably test your Java domain objects. Increase test coverage, create a solid regression test base and simplify your test/development cycle with Mean Bean.

What's My Motivation?

Picture the scene: You pick up the next task from the Sprint task board, it reads "Create Cashflow domain object". You call it out to your teammates and double check what's involved. It sounds pretty straightforward: create a new Java class with eight properties. Hang fire, Cashflow objects aren't inherently unique and you're going to store them in a `HashSet`, so you'll need to override `equals` and `hashCode` too.

You create a test class and test method, generating the domain class with the first of its fields and getter/setter pairs from the test. You flip back and forth between classes in your IDE, executing the test and correcting the implementation until you get the green light. You then move on to the second field, repeating the process… Then the third field…

Eventually you've created all the necessary tests for all eight properties, but your code coverage metrics remind you to test the overridden `equals` and `hashCode` methods too. "Great", you groan. You know only too well that you really *should* test all Equals and HashCode Contract items. Time is not on your side though. You've already spent too long writing reams of tests for *measly* getters and setters, and the prospect of delving into the murky world of contract compliance is less than appealing.

You look back at the task board only to find a stream of further domain object creation tasks. If only there was a better way to test your domain objects…

Elevator Pitch

Mean Bean is an open source Java test library that helps you rapidly and reliably test fundamental objects within Java software systems, namely domain and data objects. Mean Bean:

1. Tests that the getter and setter method pairs of a JavaBean or POJO function correctly.
2. Verifies that the `equals` method of a class complies with the Equals Contract.
3. Verifies that the `hashCode` method of a class complies with the HashCode Contract.
4. Verifies property significance in object equality.

With just three lines of code, you can be confident that your bean is well behaved. (See Listing 1.)

Listing 1. Testing a domain object using Mean Bean

```
new BeanTester().testBean(MyDomainObject.class);
new EqualsMethodTester().testEqualsMethod(MyDomainObject.class);
new HashCodeMethodTester().testHashCodeMethod(MyDomainObject.class);
```

Provided under the Apache License V2.0 and hosted at SourceForge, you can download Mean Bean from Maven Central repo now (GroupId: org.meanbean).

Be Mean to the Beans

Mean Bean helps you test three aspects of your objects: getters/setters, `equals` and `hashCode`. Each can be tested independently and is implemented by a different class: `BeanTester`,

`EqualsMethodTester` and `HashCodeMethodTester`, respectively. The core API is outlined in Table 1.

Table 1. Core Mean Bean Tester API

Class	Method
`BeanTester`	`testBean(Class<?>)`
`EqualsMethodTester`	`testEqualsMethod(Class<?>)`
`HashCodeMethodTester`	`testHashCodeMethod(Class<?>)`

Each test method takes the class to test as a parameter and throws a `java.lang.AssertionError` if the test fails.

Now for the Science Bit

Getter and Setter Testing

To test the public getter/setter methods of a class simply pass the class in question to the `testBean` method of `BeanTester`. (See Listing 2.)

Listing 2. Testing getter/setter methods of a class using Mean Bean

```
new BeanTester().testBean(MyDomainObject.class);
```

The `testBean` method implements the algorithm shown in Listing 3.

Listing 3. Algorithm used by Mean Bean to test getter/setter methods

```
for number of tests-per-property do
    create instance of object to test, x
    for each property in public getter/setter method pairs do
        generate suitable test data for property
        invoke setter with test data on x
        assert that getter on x returns same value as passed to setter
    end for
 end for
```

If a getter fails to return the same value passed to its setter, the test is failed and an `AssertionError` is thrown with a useful message. Test data is generated randomly by type-specific factories preregistered in Mean Bean. This randomisation coupled with the number of tests-per-property (default: 100) reduces the risk of type II (false negative) errors occurring whereby a getter and/or setter uses a hard-coded value that just happens to match the test data value used. As with all randomisation, it is possible for this still to occur, however collision detection logic will be added in the near future to ensure a level of randomisation is achieved. All necessary information is provided in the `AssertionError` so that failed tests can be replicated.

Equals Method Testing

To test the overridden `equals` method of a class pass the class to the `testEqualsMethod` method of `EqualsMethodTester`. (See Listing 4.)

Listing 4. Testing equals logic of a class using Mean Bean

```
new EqualsMethodTester().testEqualsMethod(MyDomainObject.class);
```

The `testEqualsMethod` method tests Equals Contract compliance as well as property significance, implementing the algorithm shown in Listing 5.

Listing 5. Algorithm used by Mean Bean to test equals logic

```
test Reflexive Equals Contract item
test Symmetric Equals Contract item
test Transitive Equals Contract item
test Consistent Equals Contract item
test Nullity Equals Contract item
test equals logic is correct for different types
for number of tests-per-property do
```

```
    test property significance in equals
end for
```

Unless specified otherwise, all properties are assumed to be significant during property significance testing. A "significant" property is one that is considered by an `equals` method and consequently one that should affect the logical equivalence of two objects. Conversely, an insignificant property is one that is not considered by an `equals` method and should have no bearing on the logical equivalence of two objects. Property significance testing implements the algorithm shown in Listing 6.

Listing 6. Algorithm used by Mean Bean to test property significance

```
for each property do
    create object x
    create object y (logically equivalent to x)
    assert x.equals(y)
    change property of y to contain a different value
    if property is insignificant then
        assert x.equals(y)
    else
        assert x.equals(y) is false
    end if
end for
```

If a property fails to have the anticipated effect on logical equality then an `AssertionError` is thrown with sufficient information to replicate the problem. To specify a property as insignificant (not considered by `equals`) simply pass its name to the `testEqualsMethod` method, which takes a vararg of insignificant property names. (See Listing 7.) All properties are considered significant unless `testEqualsMethod` is told otherwise.

Listing 7. Testing the equals logic of a class that has an insignificant property, "id", using Mean Bean

```
new EqualsMethodTester().testEqualsMethod(MyDomainClass.class, "id");
```

If a property name is passed to `testEqualsMethod` that does not exist on the class being tested, an `IllegalArgumentException` is thrown, ensuring tests and codebase are kept in sync.

HashCode Method Testing

Testing the overridden `hashCode` method of a class is akin to the previously discussed scenarios: pass the class to the `testHashCodeMethod` method of `HashCodeMethodTester`, which will throw an `AssertionError` if the test fails. (See Listing 8.)

Listing 8. Testing hashCode logic of a class using Mean Bean

```
new HashCodeMethodTester().testHashCodeMethod(MyDomainObject.class);
```

This tester only tests that logically equivalent (equal) objects have equal hashCodes and that the hashCode of an object is consistent if the object is unchanged between invocations. There is no concept of property significance in hashCode testing yet, however this will be added in the near future.

Your Flexible Friend

Deployed with sensible defaults, Mean Bean also allows you to adjust most test parameters, including:

- Number of tests-per-property
- Properties to ignore
- Use of custom test data factories
- Overriding built-in test data factories

Where appropriate, configuration changes can be made at a global level, for a single isolated test invocation, or even for just a specific property.

Mean Bean ships with a default registry of test data factories for all common Java types. Additional custom data type factories can be registered if needed, but Mean Bean will attempt to build

a factory on the fly when it encounters an unrecognised data type. By and large this will work so long as the class has a public no-arg constructor.

The User Guide provided at www.meanbean.org is the best source of information on advanced configuration and custom factory creation.

Plays Well with Others

Mean Bean is test framework-agnostic. When a badly behaved bean is found, it simply throws an `AssertionError`. Integrating Mean Bean with test frameworks like JUnit or TestNG is as simple as invoking the appropriate tester within a test method. (See Listing 9.)

Listing 9. JUnit 4 tests that test equals, hashCode and getters/setters using Mean Bean

```
public class DomainObjectTest {

    @Test public void testProperties() {
        new BeanTester().testBean(DomainObject.class);
    }

    @Test public void testEqualsMethod() {
        new EqualsMethodTester().testEqualsMethod(DomainObject.class);
    }

    @Test public void testHashCodeMethod() {
        new HashCodeMethodTester().testHashCodeMethod(DomainObject.class);
    }
}
```

A useful technique to cover your entire domain is to iterate over an array of domain classes. (See Listing 10.)

Listing 10. Testing multiple classes using Mean Bean

```
private Class<?>[] domainClasses = { Address.class, Company.class, ...};
...
for (Class<?> domainClass : domainClasses) {
    beanTester.testBean(domainClass);
    equalsTester.testEqualsMethod(domainClass);
    hashCodeTester.testHashCodeMethod(domainClass);
}
```

Mean Bean uses Apache Commons Logging, allowing you to tweak logging and inspect what's going on inside the "black box" to considerable depth.

Further Information

For more information, including a detailed User Guide, FAQ and support forum, visit www.meanbean.org. To download Mean Bean, visit the Maven Central repository at http://search.maven.org (GroupId: org.meanbean). If you need any help, you can reach the Mean Bean team at help@meanbean.org.

About the Author

Graham Williamson is the creator and lead developer of Mean Bean. He is the owner of Figment Software, a technology consultancy that provides technical architecture and software development services. Graham has over ten years experience creating technical solutions for a demanding client base in diverse environments. He holds a BSc Hons in Computing Science, collects technical certifications, and is a certified ScrumMaster and Agile advocate. Graham can be found on LinkedIn at http://uk.linkedin.com/in/grahamwilliamson.