Japanese Groovy Mocks

Groovy Mocks

Groovy

Terms

Collaborator

An ordinary Groovy or Java class that's instance or class methods are to be called.
Calling them can be time consuming or produce side effects that are unwanted when testing (e.g. database operations).
GroovyJava
()

Caller

A Groovy Object that calls methods on the Collaborator, i.e. collaborates with it.
GroovyCollaborator(collaborates with it.)

Mock

An object that can be used to augment the Collaborator. Method calls to the Collaborator will be handled by the Mock, showing a *demanded* beha vior. Method calls are *expected* to occur *strictly* in the *demanded* sequence with a given *range* of cardinality. The use of a Mock implicitly ends with *verifying* the expectations.
Collaborator MockCollaborator

Stub

Much like a Mock but the *expectation* about sequences of method calls on the Collaborator is *loose*, i.e. calls may occur out of the *demanded* ord er as long as the *ranges* of cardinality are met. The use of a Stub does not end with an implicit verification since the stubbing effect is typically asserted on the Caller. An explicit call to *verify* can be issued to assert all *demanded* method calls have been effected with the specified cardinality.
MockCollaborator CallerStub .

An extended example

System under test

We will explore a system under test inspired from the JBehave currency example.
JBehave

Our system makes use of a base currency class used to represent the currency of a particular country:

```
class Currency {
    public static final Currency USD = new Currency("USD")
    public static final Currency EUR = new Currency("EUR")
    private String currencyCode
    private Currency(String currencyCode) {
        this.currencyCode = currencyCode
    }
    public String toString() {
        return currencyCode
    }
}
```

and a base exchange rate class which encapsulates buying and selling rates for a currency:
()

```
class ExchangeRate {
    final double fromRate
    final double toRate
    public ExchangeRate(double fromRate, double toRate) {
        this.fromRate = fromRate
        this.toRate = toRate
    }
}
```

We will make use of an exchange rate service collaborator to retrieve the exchange rates for a particular country:

```
interface ExchangeRateService {
    ExchangeRate retrieveRate(Currency c)
}
```

Our class under test is a currency converter. It makes use of the following exception:

```
class InvalidAmountException extends Exception {
    public InvalidAmountException(String message) {super(message);}
}
```

and conforms to the following interface:

```
interface CurrencyConverter {
    double convertFromSterling(double amount, Currency toCurrency) throws
InvalidAmountException
    double convertToSterling(double amount, Currency fromCurrency) throws
InvalidAmountException
}
```

Here is our class under test.

```
class SterlingCurrencyConverter implements CurrencyConverter {
    private ExchangeRateService exchangeRateService

    public SterlingCurrencyConverter(ExchangeRateService exchangeRateService) {
        this.exchangeRateService = exchangeRateService;
    }

    private double convert(double amount, double rate) throws InvalidAmountException {
        if (amount < 0) {
            throw new InvalidAmountException("amount must be non-negative")
        }
        return amount * rate
    }

    public double convertFromSterling(double amount, finance.Currency toCurrency)
throws InvalidAmountException {
        return convert(amount, exchangeRateService.retrieveRate(toCurrency).fromRate)
    }

    public double convertToSterling(double amount, finance.Currency fromCurrency)
throws InvalidAmountException {
        return convert(amount, exchangeRateService.retrieveRate(fromCurrency).toRate)
    }
}
```

Mocking using Map coercion

Mocking using Map coercion

When using Java, Dynamic mocking frameworks are very popular. A key reason for this is that it is hard work creating custom hand-crafted mocks using Java.
Java

Such frameworks can be used easily with Groovy if you choose (as shown in this extended example) but creating custom mocks is much easier in Groovy.
GroovyGroovy

You can often get away with simple maps or closures to build your custom mocks.
MapClosure

Let's consider maps first.
Map

By using maps or expandos, we can incorporate desired behaviour of a collaborator very easily as shown here:
MapExpandoCollaborator

```
def service = [retrieveRate:{ new ExchangeRate(1.45, 0.57) }] as ExchangeRateService
def sterlingConverter = new SterlingCurrencyConverter(service)
double convertedAmount = sterlingConverter.convertFromSterling(10.0, Currency.USD);
assert convertedAmount == 14.50
```

For more details, see Developer Testing using Maps and Expandos instead of Mocks.