Property Randomizers

Generating large amounts of meaningful test data can be tedious and sometimes error prone if your test values arbitrarily fit your code and don't properly exercise it. Randomly generating test data objects can be a simple way to test different scenarios across the same tests and also simplify your test writing.

The `PropertyRandomizer` can be used as a builder by using the classes directly or as a DSL.

Operations

The `PropertyRandomizer` instances have six configuration methods and three builder methods. The configuration methods are:

`ignoringTypes(Class…)`

Specifies the given property types should be ignored and not randomized.

`ignoringProperties(String…)`

Specifies that the given property names should be ignored and not randomized.

`typeRandomizers(Map<Class,Object>)`

Allows the specification of multiple type randomizers, where the map is property type to the appropriate randomizer.

`typeRandomizer(Class, Object)`

Specifies a single randomizer for a property type.

`propertyRandomizers(Map<String,Object>)`

Allows the specification of multiple property randomizers, where the map is property name to the appropriate randomizer.

`propertyRandomizer(String, Object)`

Specifies a single randomizer for a property name.

The `Object` configured as a "randomizer" may be either a `PropertyRandomizer` instance or a `Closure`. If a `Closure` is used, it will allow up to two arguments to the `Closure`:

- If no arguments are specified, the returned value must be built using external code
- If one argument is specified, it will be the `Random` instance used by the randomizer for generating random data.
- If two arguments are passed in, the first will be the `Random` as above, and the second will be the instance of the target object being populated.

The allowed builder methods are:

`one()`
: Instantiates a single randomized instance of the target object.

`times(int)`
: Instantiates a collection of multiple randomized instances of size equal to the provided count.

`*(int)`
: An alias to the `times(int)` method for operator-overridden shorthand notation.

Randomizer Builder

The `PropertyRandomizer` class allows randomizers to be built directly from the mutator methods of the class. Pulling an example from the unit tests we can see how the builder uses the configuration and builder operations:

```
PropertyRandomizer rando = PropertyRandomizer.randomize(Person)
    .ignoringProperties('bankPin')
    .typeRandomizers(
        (Date)    : { new Date() },
        (Pet)     : { randomize(Pet).one() },
        (String[]): forStringArray()
    )
    .propertyRandomizers(name: { 'FixedValue' })
```

The code above creates a `PropertyRandomizer` for the `Person` class where the `bankPin` property is ignored, the `name` property is "randomized" to a fixed value "FixedValue", and the properties of type `Date`, `Pet`, and `String[]` are randomized using the provided custom randomizers. To generate random instances using this randomizer you could do any of the following:

```
Person personA = rando.one()
Collection<Person> fourPeople = rando.times(4)
Collection<Person> tenPeople = rando * 10
```

Randomizer DSL

Using the DSL version of the randomizer is not really any different than using the builder style. For the example specified above, we would have the following code to achieve the same randomizer:

```
PropertyRandomizer rando = PropertyRandomizer.randomize(Person){
    ignoringProperties 'bankPin'
    typeRandomizers(
        (Date)    : { new Date() },
        (Pet)     : { randomize(Pet).one() },
        (String[]): forStringArray()
    )
    propertyRandomizers name: { 'FixedValue' }
}
```

The main difference being that the operations are specified within a `Closure`, which could be shared or configured externally to the rest of the code.

Provided Randomizers

The library provides a collection of useful randomizers, some of which are configured by default in the `PropertyRandomizer`, in the `Randomizers` utility class. These may be used directly or as randomizer objects for the `PropertyRandomizer` configurations.

Randomizing Simple Types

The `PropertyRandomizer` is not just useful for complex objects, it will also provide random instances of more simple types, such as `String` and `Date` objects.

If the type to be randomized is one of the built-in class randomizers (`String` and the primitive types and their wrapper classes) you can randomize them directly:

```
PropertyRandomizer stringRando = PropertyRandomizer.randomize(String)
PropertyRandomizer dateRando = PropertyRandomizer.randomize(Date)
```

Other, non-default types may also be configured in a similar manner, by adding a `typeRandomizer(Class,Object)`:

```
Class byteArray = ([] as byte[]).class
PropertyRandomizer stringRando = PropertyRandomizer.randomize(byteArray){
    typeRandomizer byteArray, Randomizers.forByteArray()
}
```

This code will create random instances of `byte` arrays. The extra code for determining the `Class` for the `byte` array is a work-around for an odd issue with Groovy typing and byte arrays - not really related to the example at hand.

Generally, this approach should be left to very simple value-type objects. In many cases, you could also just use the randomizers provided in the `Randomizers` class directly.