Association Matchers

> belong_to

belong_to asserts a belongs_to is declared in the class.

```
describe Post do
  it { should belong_to(:author) }
end
```

> have_many

have_many asserts a has_many is declared in the class.

```
describe User do
  it { should have_many(:posts) }
end
```

> have_one

have_one asserts a has_one is declared in the class.

```
describe User do
  it { should have_one(:avatar) }
end
```

Association Options

Other options are available on most of the association matchers, including tests for through relationships, order, conditions, and dependent

```
describe User do
  it { should have_one(:profile).dependent(:destroy) }
  it { should have_many(:orders).order('orders.fulfilled_at DESC') }
  it { should have_many(:receipts).through(:orders) }
  it { should have_many(:active_receipts).conditions('receipts.active = 1') }
end
```

Validation Matchers

> validate_acceptance_of

Ensures the model has validation in place for accepting a question (EULA or ToS typically).

```
describe User do
  it { should validate_acceptance_of(:terms) }
end
```

> validate_format_of

Ensures the model has validation in place matching against a regular expression.

```
describe User do
  it { should validate_format_of(:ssn).with(/\A\d{3}-\d{2}-\d{4}\Z/) }
end
```

> validate_presence_of

Ensures the model has validation in place ensuring a value is set.

```
describe User do
  it { should validate_presence_of(:email) }
end
```

> validate_uniqueness_of

Ensures the model has validation in place ensuring an attribute is unique. There's a caveat: a record must already exist in the database for this test to work.

```
describe User do
  before { User.create(email: 'john@example.com') }
  it     { should validate_uniqueness_of(:email) }
end
```