FactoryBot Cheat Sheet

by Ash Angell via cheatography.com/120958/cs/21962/

Defining factories

```
FactoryBot.define do
  # Default
  # It will use the User class
  factory :user do
    first_name { "John" }
    last_name { "Doe" }
    admin { false }
  end
  # Specifying the class
  # It will use the User class, instead of Admin
class
  factory :admin, class: User do
    first_name { "Admin" }
    last_name { "User" }
    admin { true }
  end
end
```

Options (transients)

```
factory :user do
  transient do
    upcased true
  end
  after :create do |user, options|
    user.name.upcase! if options.upcased
  end
end

create(user, upcased: true)
```

Transient attributes will not get passed to the model, but will be available in after-create hooks.

Nested Factories

```
factory :user do
  first_name 'John'
  factory :sample_user do
    first_name { Faker::Name.first_name }
  end
end
create :sample_user
```

Traits

```
FactoryBot.define do
  factory :post do
    title { 'An awesome post' }
    body { 'Lorem Ipsum...' }
    trait :published do
      status { :published }
    end
    trait :unpublished do
      status { :draft }
    end

    trait :with_comments do
      after(:create) do |post|
        create_list :comment, 2, todo_item: post
      end
    end
  end
end
# then in your test
let(:post) { create(:post, :published) }
# or even with
let(:post) { create(:post, :published, :with_com-
ments) }
```

Trait helps you to remove duplication.

Building factories

```
# Returns an User instance that's not saved
user = FactoryBot.build(:user)
# Returns a saved User instance
user = FactoryBot.create(:user)
# Returns a hash of attributes that can be used to
build an User instance
attrs = FactoryBot.attributes_for(:user)
# Returns an object with all defined attributes
stubbed out
stub = FactoryBot.build_stubbed(:user)
# Passing a block to any of the methods above will
yield the return object
FactoryBot.create(:user) do |user|
  user.posts.create(attributes_for(:post))
end
# Overriding attributes of a factory
user = FactoryBot.build(:user, first_name: "Joe")
```

Building factories (cont)

```
user.first_name
# => "Joe"
# No matter which build strategy is used to
override attributes
user = FactoryBot.create(:user, first_name: "Joe")
user.first_name
# => "Joe"
```

Associations

```
# If the factory name is the same as the
association name, it's simple
factory :post do
  author
end
# You can also specify a different factory or
override attributes
factory :post do
  # ...
  association :author, factory: :user, last_name:
"Writely"
end
# Builds and saves a User and a Post
post = FactoryBot.create(:post)
post.new_record? # => false
post.author.new_record? # => false
# Builds and saves a User, and then builds but does
not save a Post
post = FactoryBot.build(:post)
post.new_record? # => true
post.author.new_record? # => false
```

Dependent attributes

```
factory :user do
  first_name { "Joe" }
  last_name { "Blow" }
  email { "#{first_name}.#{last_name}@example.c-
om".downcase }
end
```

Attributes can be based on the values of other attributes.

Aliases

```
factory :user, aliases: [:author, :commenter] do
  first_name { "John" }
  last_name { "Doe" }
  date_of_birth { 18.years.ago }
end
factory :post do
  # instead of association :author, factory: :user
  author
  title { "How to read a book effectively" }
  body { "There are five steps involved." }
end
factory :comment do
  # instead of association :commenter, factory:
:user
  commenter
  body { "Great article!" }
end
```

Aliases allow to use named associations more easily.