Observers

Designing and Maintaining Software (DAMS)

Louis Rose


4e Arr.
QUAI D'ANJOU
4e Arr.
QUAI D'ANJOU

Delivery people need to know when pizzas are ready

```
class Pizza
 def initialize(delivery_person)
  @delivery_person = delivery_person
 end

 def bake
  cook # blocking call
  @delivery_person.deliver(self)
 end
end
```

As does the web app.
As does…

```
class Pizza
  def initialize(delivery_person, website)
    @delivery_person = delivery_person
    @website = website
  end

  def bake
    cook # blocking call
    @delivery_person.deliver(self)
    @website.update_status(self, :baked)
  end
end
```

How do we avoid this coupling?

Switch to a polling model?

```ruby
class Pizza
  def initialize
    @baked = false
  end

  def bake
    cook # blocking call
    @baked = true
  end

  def baked?
    @baked
  end
end
```

```ruby
class DeliveryPerson
  def wait_for_delivery(pizza)
    sleep(1) until pizza.baked?
    deliver(pizza)
  end

  def deliver(pizza)
    puts "Delivering #{pizza}"
  end
end
```

DeliveryPerson is an *observer*

```ruby
class Pizza
  def observers
    @observers ||= []
  end

  def add_observer(object, message=:update)
    observers << [object, message]
  end

  def bake
    cook # blocking call
    notify_observers(self)
  end

  def notify_observers(*args)
    observers.each do |object, message|
      object.send(message, *args)
    end
  end
end
```

```ruby
class DeliveryPerson
  def wait_for_delivery(pizza)
    pizza.add_observer(self, :deliver)
  end

  def deliver(pizza)
    puts "Delivering #{pizza}"
  end
end
```

Ruby has an *observer* library

```ruby
require "observer"

class Pizza
  include Observable

  def bake
    cook # blocking call
    changed
    notify_observers(self)
  end
end
```

```ruby
class DeliveryPerson
  def wait_for_delivery(pizza)
    pizza.add_observer(self, :deliver)
  end

  def deliver(pizza)
    puts "Delivering #{pizza}"
  end
end
```

Don’t forget to call changed when using Observable!

A potential downside

```ruby
require "observer"

class Pizza
  include Observable

  def bake
    cook # blocking call
    changed
    notify_observers(self)
  end
end
```

```ruby
class DeliveryPerson
  def wait_for_delivery(pizza)
    pizza.add_observer(self, :deliver)
  end

  def deliver(pizza)
    puts "Delivering #{pizza}"
    address = pizza.customer.address
    deadline = pizza.order.deadline
    collect(pizza.id)
    navigate_to(address, by: deadline)
  end
end
```

A potential downside

```
require "observer"

class Pizza
  include Observable

  def bake
    cook # blocking call
    changed
    notify_observers(self)
  end
end
```

```
class DeliveryPerson
  def wait_for_delivery(pizza)
    pizza.add_observer(self, :deliver)
  end

  def deliver(pizza)
    puts "Delivering #{pizza}"
    address = pizza.customer.address
    deadline = pizza.order.deadline
    collect(pizza.id)
    navigate_to(address, by: deadline)
  end
end
```

Use a push model to avoid coupling the observer to the observable's API

Pull model

```
require "observer"

class Pizza
  include Observable

  def bake
    cook # blocking call
    changed
    notify_observers(self)
  end
end
```

```
class DeliveryPerson
  def wait_for_delivery(pizza)
    pizza.add_observer(self, :deliver)
  end

  def deliver(pizza)
    puts "Delivering #{pizza}"
    address = pizza.customer.address
    deadline = pizza.order.deadline
    collect(pizza.id)
    navigate_to(address, by: deadline)
  end
end
```

Push model

```
require "observer"

class Pizza
  include Observable

  def bake
    cook # blocking call
    changed
    address = customer.address
    deadline = order.deadline
    notify_observers(id, address, deadline)
  end
end
```

```
class DeliveryPerson
  def wait_for_delivery(pizza)
    pizza.add_observer(self, :deliver)
  end

  def deliver(id, address, deadline)
    puts "Delivering #{pizza}"
    collect(id)
    navigate_to(address, by: deadline)
  end
end
```

Pull vs Push

Pull	Push
notify_observers(self)	notify_observers(data, more_data, …)
Observable is not coupled to the data needed by the observers	Observers are not coupled to the observable's API
Prefer when: observers require different sets of data	Prefer when: all observers require same data and observers can be reused

Callback style

(Lightweight, multimodal observers)

```ruby
class Pizza
  def observers
    @observers ||= Hash.new { |h,k| h[k] = [] }
  end

  def before_baking(&callback)
    observers[:before_baking] << callback
  end

  def after_baking(&callback)
    observers[:after_baking] << callback
  end

  def bake
    notify_observers(:before_baking, self)
    cook # blocking call
    notify_observers(:after_baking, self)
  end

  def notify_observers(event, *args)
    observers[event].each do |o|
      o.call(args)
    end
  end
end
```

```ruby
class Website
  def track(pizza)
    pizza.before_baking do |pizza|
      update_status_page(pizza, "in the oven")
    end
    pizza.after_baking do |pizza|
      update_status_page(pizza, "on its way")
    end
  end
end
```

Summary

Use observers to avoid coupling an object to other objects that care about its state

Push observers avoid coupling the observers to the observable's API. Pull observers facilitate lots of different types of observer.

Callbacks are lightweight and multimodal observers that are fairly popular in Ruby code