Refactoring: A Process of Ongoing Improvement

Dinah Shi
@dinahshi

A refactor is a change made to the internal structure of software without changing its observable behaviour.

Make code easier to change

SOLID Principles

- coupling (the degree to which modules depend on one another)

+ cohesion (the degree to which elements in a module are related)

Make software easier to understand

Improves the design of software

A process of ongoing improvement

When?

When you’re adding a new feature

When you’re fixing a bug

When you’re reviewing code

Pick one goal

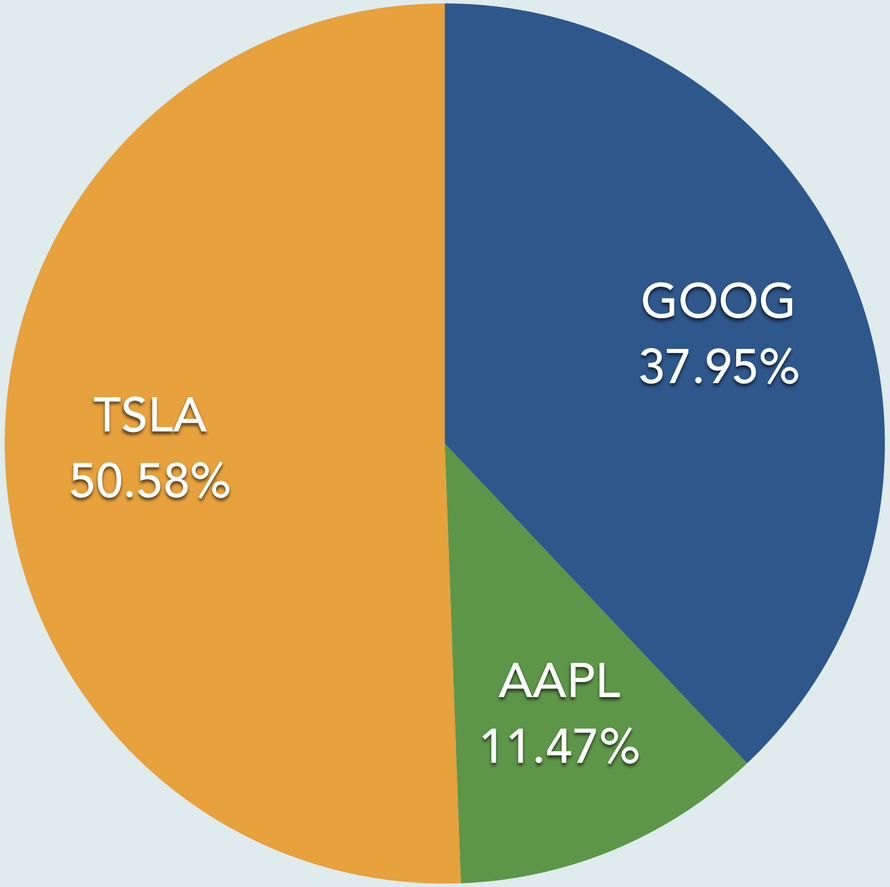
Write tests

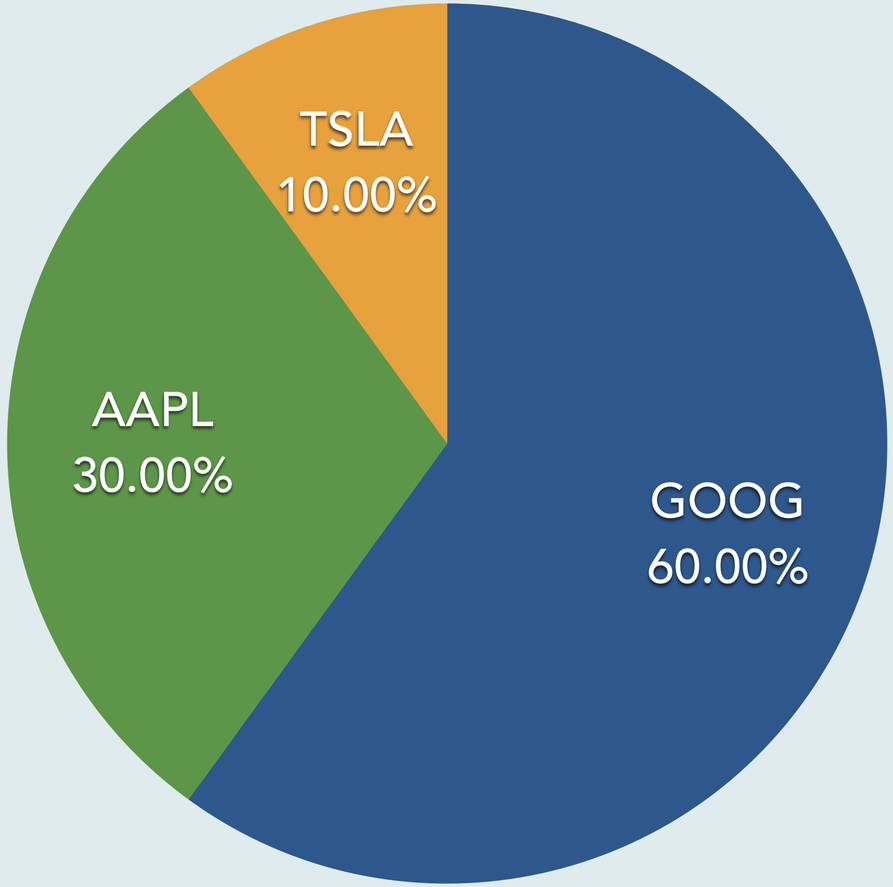
One step at a time

Keep your two hats separate

Example: account rebalancing

Target Allocation	Symbol	Shares	Price
0.6	GOOG	5	794
0.3	APPL	10	120
0.1	TSLA	21	252


GOOG
37.95%
AAPL
11.47%
TSLA
50.58%


TSLA
10.00%
AAPL
30.00%
GOOG
60.00%

calculate desired # of shares
get difference between desired and current
buy or sell

Buy 2 shares of GOOG
Buy 16 shares of AAPL
Sell 17 shares of TSLA

```ruby
class Account
  def initialize(investments)
    @investments = investments
  end

  def rebalance
    quotes = @investments.map { |investment| [investment[:symbol], QuoteService::getPrice(investment[:symbol])] }.to_h
    total = @investments.reduce(0) { |sum, investment| sum + investment[:shares_owned] * quotes[investment[:symbol]] }

     @investments.map do |investment|
      ideal_shares = investment[:target_allocation] * total / quotes[investment[:symbol]]
      difference = ideal_shares.floor - investment[:shares_owned]

      if difference < 0
        "sell #{difference * -1} shares of #{investment[:symbol]}"
      else
        "buy #{difference} shares of #{investment[:symbol]}"
      end
    end
  end
end

```

```ruby
class Account
  def initialize(investments)
    @investments = investments
  end

  def rebalance
    format_orders(calculate_orders)
  end

private

  def calculate_orders
    @investments.map do |investment|
      Order.new(difference(investment), investment.symbol)
    end
  end

  def difference(investment)
    ideal_shares = investment.target_allocation * total_value / investment.price
    ideal_shares.floor - investment.shares_owned
  end

  def format_orders(orders)
    orders.map(&:formatted_string)
  end

  def total_value
    @investments.reduce(0) do |sum, investment|
      sum + investment.shares_owned * investment.price
    end
  end
end

class Order < Struct.new(:difference, :symbol)
  def formatted_string
    "#{instruction} #{difference.abs} shares of #{symbol}"
  end

private

  def instruction
    difference < 0 ? "sell" : "buy"
  end
end

class Investment < Struct.new(:target_allocation, :symbol, :shares_owned)
  def price
    QuoteService::getPrice(symbol)
  end
end
```

Replace hash with object
Replace temp with query
Move method
Extract method
Extract data object
Extract object

Introduce Null Object

```ruby
class Guest
  attr_reader :email, :dietary_preference
  def initialize(email, dietary_preference)
    @email = email
    @dietary_preference = dietary_preference
  end

  def meal
    if dietary_preference
      dietary_preference.meal
    else
      MissingMealNotificationWorker.perform(self)
      "not specified"
    end
  end
end

```

```ruby
class Guest
  attr_reader :email, :dietary_preference
  def initialize(email, dietary_preference)
    @email = email
    @dietary_preference = dietary_preference || NullDietaryPreference.new
  end

  def meal
    dietary_preference.meal
  end
end

class NullDietaryPreference
  def meal
    MissingMealNotificationWorker.perform(self)
    "not specified"
  end
end

```

Refactoring: Improving the Design of Existing Code
(Martin Fowler)

Refactoring: Ruby Edition (Jay Fields)

Refactoring from Good to Great (Ben Orenstein)

Questions?