Pauta Pregunta 4

a) El objetivo de esta pregunta es que el alumno modele un problema, reconociendo las relaciones que existen entre las clases y las restricciones de negocio. Las relaciones se traducen en la asociaciones en Rails, y las restricciones en las validaciones. Es requisito que el alumno haya ocupado las clases descritas para solucionar el problema. Si agregó más clases no hay ningún problema.

Distribución de Puntajes:

Clases y migraciones implementadas:
Por cada clase implementada (sin considerar asociaciones ni validaciones) se suma 0.5 puntos, y por cada migración se suma otros 0.5 puntos. Entonces tenemos:

- **(1 punto)** Client
- **(1 punto)** DeliveryWorker
- **(1 punto)** Order
- **(1 punto)** Pizza
- **(1 punto)** PizzaPart
- **(1 punto)** Ingredient
- **(1 punto)** Solución a la relación entre PizzaPart e Ingredient (Una clase PizzaPartIngredient por ejemplo)

Asociaciones implementadas:

- **(1 punto)** Un cliente puede pedir muchas órdenes, y una orden pertenece a un cliente.
- **(1 punto)** Un repartidor puede entregar muchas órdenes, y una orden es entregada por un repartidor.
- **(1 punto)** Una orden puede tener muchas pizzas, y una pizza pertenece a una orden.
- **(1 punto)** Una pizza puede tener muchas partes, y cada parte sólo pertenece a una pizza.
- **(2 punto)** Una parte puede tener muchos ingredientes, y un ingrediente puede pertenecer a muchas partes de pizza.

Las validaciones mínimas necesarias:

- **(1 punto)** Identificar el cliente y dirección: Un rut o nombre + dirección.
- **(1 punto)** Identificar al repartidor: Un rut o nombre + tipo de vehículo.
- **(1 punto)** Una orden debe tener al menos una pizza.
- **(1 punto)** Una pizza debe estar compuesta por 1, 2, 4 o 8 partes.
- **(1 punto)** La pizza debe tener tipo de pizza o tamaño, tipo de masa y tipo de queso.
- **(1 punto)** Una parte de pizza (PizzaPart) debe tener al menos un ingrediente.
- **(1 punto)** Identificar el ingrediente por un nombre.

Puntaje total parte a) (20 puntos)

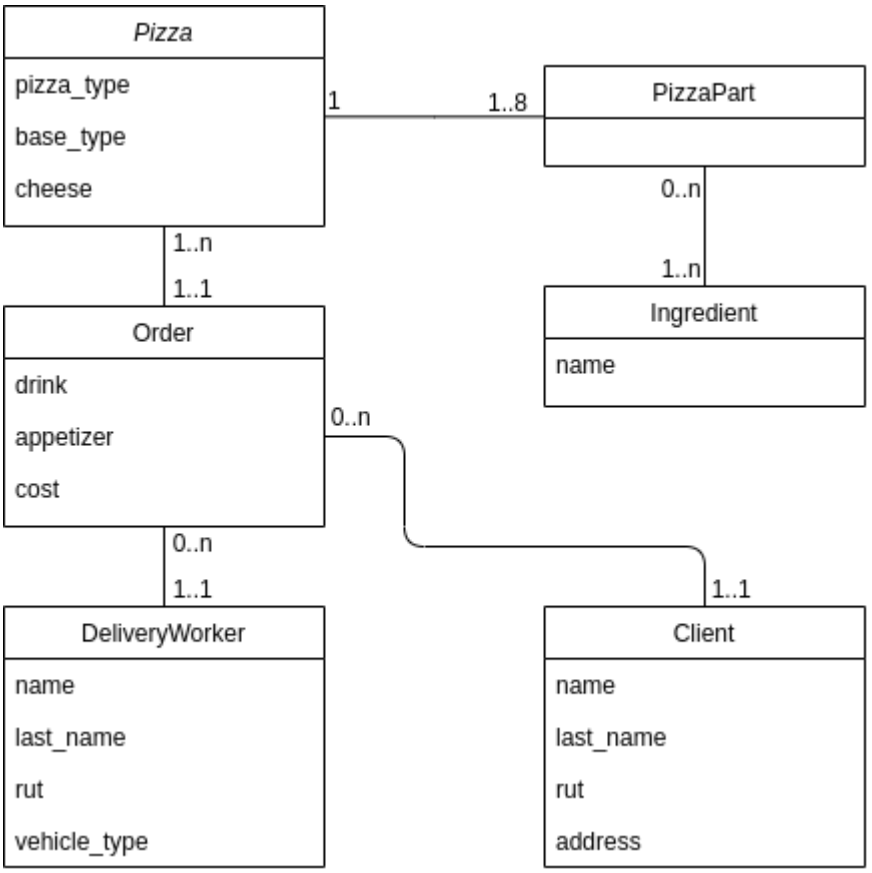
b) El objetivo de esta pregunta es que el alumno pueda obtener información de valor para un cliente con la solución propuesta en la parte a). Si por alguna razón no completó la parte a), se puede corregir asumiendo que existen las clases mencionadas en el enunciado y con los supuestos que el alumno haya dejado explícitamente.
    i) **(3 puntos)** ¿Qué ingrediente es el más pedido?
    ii) **(2 puntos)** ¿Cuantos pedidos llevan bebidas o aperitivos?
    iii) **(2 puntos)** ¿Cuánto dinero ha ganado con pedidos que sólo fueron repartidos por repartidores en motos?

Puntaje total parte b) (7 puntos)

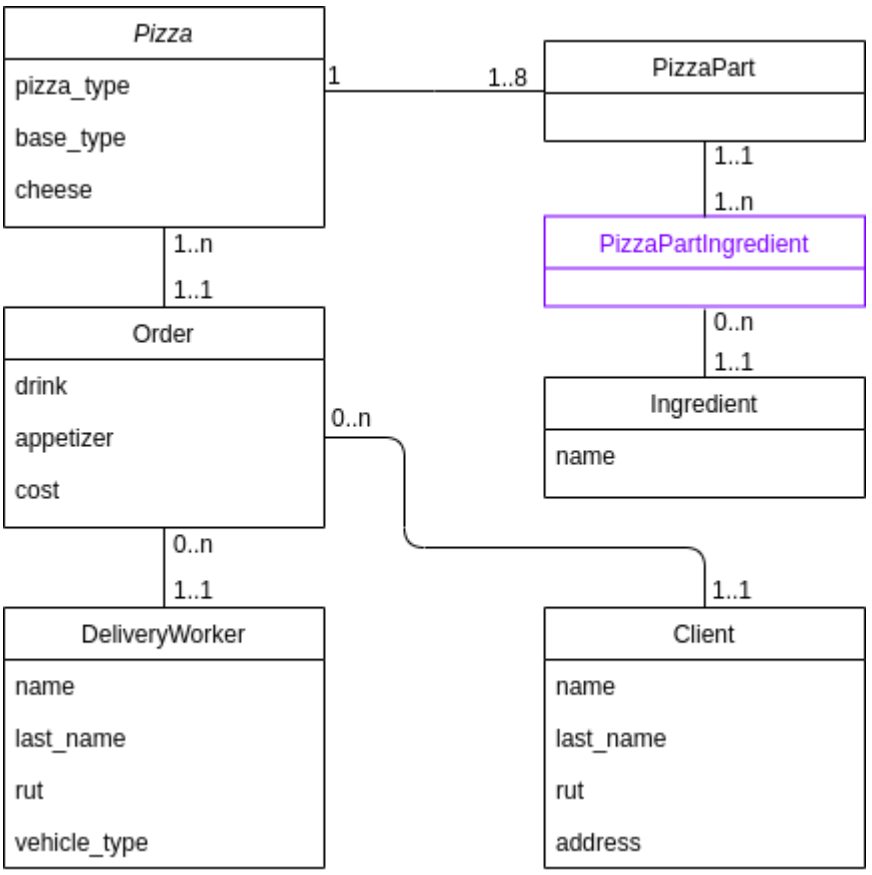
c) El objetivo de esta pregunta es que el alumno reconozca cuál es la parte crítica de su código e implemente uno o varios tests que prueben ese código más vulnerable.
    i) **(1 punto)** Es código crítico
    ii) **(1 punto)** El test está correctamente implementado
    iii) **(1 punto)** El test prueba todos los posibles casos para ese método, callback o validación

Puntaje total parte c) (3 puntos)

Solución Propuesta: La modelación del problema con las clases mencionadas queda detallada en el siguiente diagrama de clases:



Como la relación entre PizzaPart e Ingredient es de N-N, para modelarse en Rails debemos crear una tabla intermedia. Siguiendo las convenciones de Rails, la tabla intermedia sería PizzaPartIngredient. El diagrama de clases y sus relaciones queda de la siguiente forma:



A continuación se detallan las migraciones y clases de esta solución:
Migraciones:

```ruby
class CreateClients < ActiveRecord::Migration[5.2]
 def change
   create_table :clients do |t|
     t.string :name
     t.string :rut
     t.string :last_name
     t.string :address

     t.timestamps
   end
 end
end
```

```ruby
class CreateDeliveryWorkers < ActiveRecord::Migration[5.2]
 def change
   create_table :delivery_workers do |t|
     t.string :name
     t.string :rut
     t.string :last_name
     t.integer :vehicle_type
```

```
    t.timestamps
  end
 end
end
```

```
class CreateOrders < ActiveRecord::Migration[5.2]
 def change
   create_table :orders do |t|
     t.belongs_to :client, index: true
     t.belongs_to :delivery_worker, index: true
     t.integer :drink
     t.integer :appetizer
     t.float :cost

     t.timestamps
   end
 end
end
```

```
class CreatePizzas < ActiveRecord::Migration[5.2]
 def change
   create_table :pizzas do |t|
     t.belongs_to :order, index: true
     t.integer :pizza_type
     t.integer :base_type
     t.integer :cheese

     t.timestamps
   end
 end
end
```

```
class CreateIngredients < ActiveRecord::Migration[5.2]
 def change
   create_table :ingredients do |t|
     t.string :name

     t.timestamps
   end
 end
end
```

```ruby
class CreatePizzaParts < ActiveRecord::Migration[5.2]
 def change
   create_table :pizza_parts do |t|
     t.belongs_to :pizza
     t.timestamps
   end
 end
end
```

```ruby
class CreatePizzaPartIngredients < ActiveRecord::Migration[5.2]
 def change
   create_table :pizza_part_ingredients do |t|
     t.belongs_to :pizza_part
     t.belongs_to :ingredient
     t.timestamps
   end
 end
end
```

Clases con asociaciones y validaciones:

models/client.rb

```ruby
class Client < ApplicationRecord
 has_many :orders
 validates :rut, uniqueness: true
 validates :name, :last_name, :address, :rut, presence: true
 validates :rut, format: { with: /\d{1,3}(?:\d{1,3}){2}-[\dkK]/ }
end
```

models/delivery_worker.rb

```ruby
class DeliveryWorker < ApplicationRecord
 enum vehicle_type: { car: 0, motorbike: 1, bike: 2 }
 has_many :orders

 validates :rut, uniqueness: true
 validates :name, :last_name, :vehicle_type, :rut, presence: true
 validates :rut, format: { with: /\d{1,3}(?:\d{1,3}){2}-[\dkK]/ }
end
```

models/order.rb

```
class Order < ApplicationRecord
 enum drink: { cocacola: 0, fanta: 1, sprite: 2 }
 enum appetizer: { garlic_sticks: 0, cinnamon_sticks: 1, cheese_rolls: 2 }

 belongs_to :client
 belongs_to :delivery_worker
 has_many :pizzas

 before_validation :calculate_cost

 validates :pizzas, :length => { minimum: 1 }
 validates :cost, presence: true


 private
 def calculate_drink_cost
   drink.present? ? 1500 : 0
 end

 def calculate_appetizer_cost
   appetizer.present? ? 2000 : 0
 end

 def calculate_pizzas_cost
   total = 0
   pizzas.each{ |pizza|
     total += pizza.calculate_cost
   }
   return total
 end

 def calculate_cost
   self.cost = calculate_drink_cost + calculate_appetizer_cost +
calculate_pizzas_cost
 end
end
```

models/pizza.rb

```
class Pizza < ApplicationRecord
 enum pizza_type: { small: 0, medium: 1, large: 2 }
 enum base_type: { thin: 0, regular: 1, thick: 2 }
 enum cheese: { no_cheese: 0, mozzarella: 1, parmesan: 2 }
 belongs_to :order
 has_many :pizza_parts

 validates :pizza_type, :base_type, :cheese, presence: true
 validate :complete_pizza

 def complete_pizza
   unless pizza_parts.size == 1 || pizza_parts.size == 2 || pizza_parts.size
== 4 || pizza_parts.size == 8
     errors.add(:base, "pizza is not complete")
   end
 end

 def calculate_cost
   total = 0
   case pizza_type
   when Pizza.pizza_types[:small]
     total = 9990
   when Pizza.pizza_types[:medium]
     total = 14590
   else
     total = 17390
   end
   return total
 end

end
```

models/pizza_part.rb

```
class PizzaPart < ApplicationRecord
 belongs_to :pizza
 has_many :pizza_part_ingredients
 has_many :ingredients, through: :pizza_part_ingredients

 validates :pizza_part_ingredients, :length => { minimum: 1 }
end
```

models/ingredient.rb

```
class Ingredient < ApplicationRecord
 has_many :pizza_part_ingredients
 has_many :pizza_parts, through: :pizza_part_ingredients

 validates :name, presence: true
end
```

models/pizza_part_ingredient.rb

```
class PizzaPartIngredient < ApplicationRecord
 belongs_to :pizza_part
 belongs_to :ingredient
end
```

d) Los métodos pedidos:

¿Qué ingrediente es el más pedido?

opcion 1: Usando sólo una consulta:

```
def most_requested_ingredient
Ingredient.joins(:pizza_part_ingredients).group("pizza_part_ingredients.ingredient_
id").order("count(pizza_part_ingredients.ingredient_id) desc").first[:name]
end
```

opción 2: Iterando sobre una consulta:

```
def most_requested_ingredient2
 all_ingredients = Ingredient.includes(:pizza_part_ingredients).all
 ingredients = []
 all_ingredients.each { |ingredient|
   ingredients.push({ ingredient: ingredient.name, times_ordered:
ingredient.pizza_part_ingredients.size })
 }
 ingredients_ordered = ingredients.sort_by { |i| i[:times_ordered] }
 return ingredients[0][:ingredient]
end
```

¿Cuantos pedidos llevan bebidas o aperitivos?

```
def orders_with_drinks_or_appetizers
 Order.where.not(drink: nil).or(Order.where.not(appetizer: nil)).size
end
```

¿Cuánto dinero ha ganado con pedidos que sólo fueron repartidos por repartidores en motos?

```
def money_by_motorbikes
 orders = Order.joins(:delivery_worker).where(:delivery_workers => {
:vehicle_type => DeliveryWorker.vehicle_types[:motorbike] })
 total = 0
 orders.each { |order| total += order.cost }
 return total
end
```

e) Test a una parte crítica del código. Lo más importante es que una pizza pase a la cocina de forma completa. Por lo que se testea la validación custom complete_pizza:

```ruby
require 'rails_helper'

RSpec.describe Pizza, type: :model do
 describe 'complete_pizza' do
   let(:client) { create(:client) }
   let(:delivery_worker) { create(:delivery_worker) }
   let(:ingredient) { create(:ingredient) }

   before(:all) do
     @default_pizza_params = {
       pizza_type: Pizza.pizza_types[:small],
       base_type: Pizza.base_types[:regular],
       cheese: Pizza.cheeses[:parmesan]
     }
   end

   before(:each) do
     @order = Order.new(
       client: client,
       delivery_worker: delivery_worker
     )
     @pizza = Pizza.new(@default_pizza_params)
   end

   def add_part_to_pizza
     part = PizzaPart.new(pizza: @pizza)
     part.pizza_part_ingredients.build(ingredient: ingredient)
     @pizza.pizza_parts << part
   end

   def add_pizza_to_order
     @order.pizzas << @pizza
   end

   def add_parts_to_pizza (parts)
     parts.times do
       add_part_to_pizza
     end
     add_pizza_to_order
   end

   context 'when creating a pizza with valid divisions' do
     it 'saves the record with 1 part' do
       add_parts_to_pizza(1)
       expect(@order.save).to be(true)
     end
```

```
    it 'saves the record with 2 parts' do
      add_parts_to_pizza(2)
      expect(@order.save).to be(true)
    end

    it 'saves the record with 4 parts' do
      add_parts_to_pizza(4)
      expect(@order.save).to be(true)
    end

    it 'saves the record with 8 parts' do
      add_parts_to_pizza(8)
      expect(@order.save).to be(true)
    end
  end

  context 'when creating a pizza with invalid divisions' do
    it 'fails to save the record with 3 parts' do
      add_parts_to_pizza(3)
      expect(@order.save).to be(false)
    end

    it 'fails to save the record with 5 parts' do
      add_parts_to_pizza(5)
      expect(@order.save).to be(false)
    end

    it 'fails to save the record with 100 parts' do
      add_parts_to_pizza(100)
      expect(@order.save).to be(false)
    end
  end

  context 'when creating a pizza with no parts' do
    it 'fails to save the record' do
      add_parts_to_pizza(0)
      expect(@order.save).to be(false)
    end
  end
 end
end
```