

New chat

Crear Granja Sostenible

Polimorfismo en Ruby ejemplo

Function for summing array

Function to Sum Array

Verificar media aritmética re

Valor de Integral Míad

Integrales y su fórmula

Resolver integrales impropias

Ejemplo de Límites Matemati

Comprobar Función Densidad

"Entiendo No La Pregunta"

MCM de 2 3 4 5

Integrales Con Ejemplos

Progresión geométrica razón

Convert CFG to CNF

Ajustando modelo logístico

Convertir una CFG a PNC

New chat

Coefficiente de Determinación

Coefficiente Correlation Pear

Show more

La jerarquía de clases para esta tarea podría ser la siguiente:

1. Clase principal: "SustainableFarm"

2. Clase hija: "CarbonFootprint" y "WaterFootprint"

3. Clase nieta: "Animal"

La estructura de directorio podría ser la siguiente:

```
|-- project_directory
|   |-- sustainable_farm.rb
|   |-- carbon_footprint.rb
|   |-- water_footprint.rb
|   |-- animal.rb
|   |-- tests
|   |
|   |-- sustainable_farm_test.rb
|   |-- carbon_footprint_test.rb
|   |-- water_footprint_test.rb
|   |-- animal_test.rb
|
|--
```

La clase "SustainableFarm" tendría los siguientes atributos:

- "name": identificador de la granja
- "carbon\_footprint": instancia de la clase "CarbonFootprint"
- "water\_footprint": instancia de la clase "WaterFootprint"
- "animals": arreglo de instancias de la clase "Animal"

La clase "CarbonFootprint" y "WaterFootprint" tendrían un atributo "value" que representaría el valor de huella correspondiente.

La clase "Animal" tendría los siguientes atributos:

- "identifier": identificador del animal
- "type": tipo de animal
- "price": precio del animal

Para el caso de las pruebas unitarias se utilizaría la librería "minitest" para el desarrollo de las pruebas unitarias, y para los casos de pruebas se utilizarían diferentes escenarios para cada clase, para poder cubrir todas las posibilidades y validar el correcto funcionamiento de cada uno de los métodos de la clase.

Me das el código de las diferentes clases y del unit test

Clase "SustainableFarm":

```
class SustainableFarm
  attr_accessor :name, :carbon_footprint, :water_footprint, :animals

  def initialize(name, carbon_footprint, water_footprint, animals)
    @name = name
    @carbon_footprint = carbon_footprint
    @water_footprint = water_footprint
    @animals = animals
  end

  def add_animal(animal)
    @animals << animal
  end

  def remove_animal(animal)
    @animals.delete(animal)
  end

  def total_animals_price
    total_price = 0
    @animals.each { |animal| total_price += animal.price }
    total_price
  end
end
```

Clase "CarbonFootprint":

```
class CarbonFootprint
  attr_accessor :value

  def initialize(value)
    @value = value
  end
end
```

Clase "WaterFootprint":

```
class WaterFootprint
  attr_accessor :value

  def initialize(value)
    @value = value
  end
end
```

Clase "Animal":

```
class Animal
  attr_accessor :identifier, :type, :price

  def initialize(identifier, type, price)
    @identifier = identifier
    @type = type
    @price = price
  end
end
```

Pruebas unitarias:

```
require "minitest/autorun"
require_relative "sustainable_farm"
require_relative "carbon_footprint"
require_relative "water_footprint"
require_relative "animal"

class SustainableFarmTest < Minitest::Test
  def setup
    @carbon_footprint = CarbonFootprint.new(10)
    @water_footprint = WaterFootprint.new(5)
    @animal1 = Animal.new("1", "cow", 10)
    @animal2 = Animal.new("2", "pig", 5)

    @farm = SustainableFarm.new("Farm 1", @carbon_footprint, @water_footprint, [@animal1, @animal2])
  end

  def test_add_animal
    @animal3 = Animal.new("3", "sheep", 7)
    @farm.add_animal(@animal3)
    assert_equal 3, @farm.animals.length
  end

  def test_remove_animal
    @animal = @farm.animals[0]
  end
```