Харківський національний університет ім. В.Н. Каразіна
Факультет комп’ютерних наук
Кафедра електроніки та управляючих систем

ЗВІТ
З ЕКЗАМЕНАЦІЙНОЇ РОБОТИ
дисципліни: «Мови прикладного програмування»

Виконав: студентка групи ЗКС31
Москалюк С.Ю.

Перевірив:
Паршенцев Б.В.

Харків
2023

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ (ЗАВДАННЯ) № 11

1. Паттерни(тіп/переваги/недоліки/реалізація) Абстрактна фабрика Ruby (5 балів)
2. Які механізми реалізації спадку і включення модулів в Ruby, і в чому їх відмінності?(5 балів)
3. Як Ruby обробляє виключення (exceptions) і які важливі властивості виключень в Ruby?(5 балів)
4. Як ви визначаєте модулі (modules) в Ruby, і як вони використовуються для досягнення багаторазового успадкування?(5 балів)
5. (Практичне завдання) Структури даних та алгоритми
   - Сортування підрахунком (20 балів)

1. Абстрактна фабрика Ruby – це породжуючий патерн проектування, який надає інтерфейс, що дозволяє створювати сімейства взаємодіючих або взаємозалежних об’єктів без уточнення їх класів. Отже, абстрактна фабрика є патерном проектування типу породжуючий(creational), оскільки такий тип забезпечує механізм створення об’єктів.

Переваги:

- інкапсулює деталі створення об’єкту, забезпечуючи чистий інтерфейс для створення сімейств пов’язаних або залежних об’єктів;
- гарантує, що створені об’єкти належить до того самого сімейства та мають узгоджені інтерфейси, що сприяє їх взаємозамінності;
- дозволяє впроваджувати нові варіанти продуктів без зміни існуючого клієнтського коду;
- клієнтський код ізольований від специфіки продукту реалізації, спрощуючи заміну або вдосконалення окремих компонентів.

Недоліки:

- може внести додаткові труднощі, особливо працюючи з великою кількістю типів та сімейств продуктів;

- тісний зв’язок, що може бути між клієнтським кодом та конкретними реалізаціями продуктів, ускладнюючи перемикання між різними сімействами продуктів.

Приклад реалізації:

```
class AbstractFactory     #абстрактна фабрика
  def create_product_a
    raise NotImplementedError, "#{self.class} has not implemented
method '#{__method__}'"
  end

  def create_product_b
    raise NotImplementedError, "#{self.class} has not implemented
method '#{__method__}'"
  end
end

class ConcreteFactory1 < AbstractFactory      #конкретна фабрика1
  def create_product_a
    ProductA1.new
  end

  def create_product_b
    ProductB1.new
  end
end

class ConcreteFactory2 < AbstractFactory      #конкретна фабрика2
  def create_product_a
    ProductA2.new
  end

  def create_product_b
    ProductB2.new
  end
end

class AbstractProductA      #абстрактний продуктA
  def operation_a
    raise NotImplementedError, "#{self.class} has not implemented
method '#{__method__}'"
  end
end

class ProductA1 < AbstractProductA      #конкретний продуктA1
  def operation_a
    "Product A1 operation"
  end
end

class ProductA2 < AbstractProductA      #конкретний продуктA2
  def operation_a
    "Product A2 operation"
  end
end

class AbstractProductB      #абстрактний продукт B
  def operation_b
    raise NotImplementedError, "#{self.class} has not implemented
method '#{__method__}'"
```

```
  end
end

class ProductB1 < AbstractProductB      #конкретний продуктB1
  def operation_b
    "Product B1 operation"
  end
end

class ProductB2 < AbstractProductB      #конкретний продуктB2
  def operation_b
    "Product B2 operation"
  end
end
```

2. Механізми спадкування та включення модулів у Ruby – це два механізми повторного використання коду, які використовуються для різних цілей. Розглянемо кожен механізм, їх реалізацію та відмінності.

Почнемо з спадкування:

Реалізація:

```
class ParentClass
  # Код батьківського класу
end

class ChildClass < ParentClass
  # Код дочірнього класу
end
```

Спадкування формує ієрархію класів, де дочірній клас унаслідує властивості та методи з батьківського класу. Також кожен клас може мати лише один батьківський клас, тому що Ruby не підтримує множинну спадковість. При використанні ключового слова “super”, викликається метод батьківського класу в дочірній клас. Якщо метод не має в собі жодних аргументів, він автоматично передає всі його аргументи.

Включення:

Реалізація:

```
module MyModule
  # Код модуля
end

class MyClass
  include MyModule
  # Код класу
end
```

Модуль може бути включений в декілька класів, що дозволяє розділити функціональність між різними класами. З цього витікає те, що клас може включати декілька модулів, що дозволяє використовувати

функціональність з різних джерел. Також, для того, щоб включити метод з модулю, треба використовувати ключове слово “include”.

Отже, різниця механізмів полягає в тому, що, наприклад, спадкування створює ієрархію класів, у той час, коли модуль включення більш стосується композиції та не вимагає строгої ієрархії. Також, спадкування в Ruby може виконуватись лише з одного класу, коли модуль включення підтримує декілька модулів, що включені в клас, впроваджуючи форму множинного успадкування. Більш того, модулі включення є більш гнучкими за спадкування, оскільки вони дозволяють змішувати та поєднувати поведінку в класі. Така гнучкість може бути корисною при розробці систем, що вимагають більш динамічну композицію функціональності.

3. В Ruby, обробка винятків використовує блоки begin, rescue, та ensure. Ось приклад структури обробки винятків:

```ruby
begin
  # Код, в якому може виникнути виняток
  result = 10 / 0
rescue ZeroDivisionError => e
  # Обробка винятку ZeroDivisionError
  puts "Помилка ділення на нуль: #{e.message}"
rescue => e
  # Обробка будь-якого іншого винятку
  puts "Виникла помилка: #{e.message}"
else
  # Викликається, якщо винятку не виникло
  puts "Все виконано без помилок"
ensure
  # Викликається завжди, незалежно від того, чи був виняток
  puts "Завершується обробка винятків"
end
```

де begin: починає блок, в якому може виникнути виняток;

rescue: визначає блок, який виконується при виникненні конкретного винятку. Може бути кілька блоків rescue для обробки різних видів винятків. Також його можна використовувати без вказівки конкретного типу винятку, щоб «ловити» будь-які винятки;

else: викликається, якщо винятків в блоку begin не виявлено;

ensure: викликається завжди, незалежно від того, чи був виняток чи ні. Використовується для виконання завершальних операцій.

Властивості:

- в Ruby винятки можуть бути будь-якого типу, але зазвичай успадковуються від класу Exception або StandartError. До поширених класів винятків належать RuntimeError, NoMethodError, ArgumentError та інші;
- винятки, успадковані від класу Exception, містять в собі інформацію про помилку, як-от: повідомлення про помилку та тип винятку;
- коли виникає виняток, Ruby зберігає інформацію про те, де він виник, у вигляді трасування стеку;
- можна вручну згенерувати виняток, використовуючи ключове слово “raise”. Також існує конструкція “throw” для генерування винятків на вищому рівні програми;
- використання “rescue” без аргументу перехоплює усі винятки, що успадковані від класу StandartError. А також використовуючи таке ключове слово, можна вказати як сами обробляти той чи інший виняток.

4. Модулі, які використовуються в Ruby, є інструментами для групування та організації коду, який містить методи, константи та інші функціональні елементи. Модулі дозволяють збирати код і використовувати його в інших програмах, класах або модулях.

Приклад створення та використання модулю:

```
module MyModule
  MY_CONSTANT = 42
  def my_method
    puts "This is a method from MyModule"end
end
class MyClass
  include MyModule
  def another_method
    puts "This is another method in MyClass"
  end
end
obj = MyClass.new
obj.my_method
puts MyModule::MY_CONSTANT
```

Використання модулів для багаторазового успадкування:

Для того щоб включити модуль у клас, треба використовувати ключове слово “include”, що дозволяє класу використовувати методи та константи з модулю, ніби вони є частиною класу. Можна включати як один модуль, так і декілька. Наприклад:

```ruby
module ModuleA
  def method_a
    puts "Метод з ModuleA"
  end
end

module ModuleB
  def method_b
    puts "Метод з ModuleB"
  end
end

class MyClass
  include ModuleA
  include ModuleB
end

obj = MyClass.new
obj.method_a
obj.method_b
```

Якщо клас містить декілька модулів, методи яких мають однакову назву, пріоритет має останній включений метод.

5.

```
D:\Ruby32-x64\bin\ruby.exe D:/practices/3year/ruby/exam/exam.rb
Enter your array divided by coma:
3,7,8,9,3,5,8,2,3,8,0,2,4,8,1,4,9,3,8
Your original array: [3, 7, 8, 9, 3, 5, 8, 2, 3, 8, 0, 2, 4, 8, 1, 4, 9, 3, 8]
Your sorted array: [0, 1, 2, 2, 3, 3, 3, 3, 4, 4, 5, 7, 8, 8, 8, 8, 8, 9, 9]

Process finished with exit code 0
```

Рисунок 1 – результат виконання програми до завдання 5

Лістинг 1:

```ruby
def counting_sort(arr)
  max_value = arr.max
  min_value = arr.min
  range = max_value - min_value + 1

  count = Array.new(range, 0)
  output = Array.new(arr.length)

  arr.each { |value| count[value - min_value] += 1 }

  (1...range).each { |i| count[i] += count[i - 1] }

  (arr.length - 1).downto(0) do |i|
    output[count[arr[i] - min_value] - 1] = arr[i]
```

```ruby
    count[arr[i] - min_value] -= 1
  end

  output
end

puts "Enter your array divided by coma:"
input_str = gets.chomp
input_arr = input_str.split(',').map(&:to_i)

sorted_arr = counting_sort(input_arr)

puts "Your original array: #{input_arr}"
puts "Your sorted array: #{sorted_arr}"
```