Харківський національний університет імені В. Н. Каразіна
Факультет комп’ютерних наук
Кафедра штучного інтелекту та програмного забезпечення

ЗВІТ
З ЗАЛІКОВОЇ РОБОТИ
З дисципліни “ Стек технологій програмування”

Виконав: студент групи КС32
Бабенко Андрій
Перевірив:
Поршанцев Б.

Харків
2024

Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Спеціальність 122 Комп'ютерні науки
Спеціалізація / освітня програма Інформаційні управляючі системи та технології
Семестр 1-й
Форма навчання: денна, заочна
Рівень вищої освіти (освітньо-кваліфікаційний рівень): бакалавр
Навчальна дисципліна: «Стек технологій програмування»

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 1

1. Що таке класи та об'єкти в Ruby? Як створити клас?(5 балів)
2. Як працюють гетери та сетери? Що таке `attr_reader`, `attr_writer` та `attr_accessor`?(5 балів)
3. Як працює механізм виділення пам'яті під об'єкти в Ruby?(5 балів)
4. Поясніть, як можна змінити стандартну поведінку методу за допомогою метапрограмування.(5 балів)
5. (Практичне завдання) Реалізуйте програму, яка:Запускає три потоки, кожен з яких записує різні дані у спільний файл. Використовує механізм блокування для уникнення конфліктів. (20 балів).

Затверджено на засіданні кафедри інтелектуальних програмних систем і технологій
Протокол № 3 від «7» листопада 2024 р.

В.о. зав. кафедри  Олег ОЛЕШКО Екзаменатор  Богдан Паршенцев

1. Що таке класи та об’єкти в Ruby? Як створити клас?(5 балів)

Класи

Клас у Ruby — це шаблон (або "чертеж"), який визначає властивості (атрибути) та поведінку (методи) об'єктів. Класи використовуються для створення об'єктів із подібними характеристиками. Атрибути описують стан об'єкта (змінні екземпляра). Методи визначають поведінку об'єкта (що він може робити).

Об’єкти

Об’єкт — це конкретний екземпляр класу. Об’єкти мають доступ до методів і змінних, визначених у класі.

Створення класу

Клас створюється за допомогою ключового слова class. Ім'я класу починається з великої літери за конвенцією.

```
class Car

  # Конструктор для ініціалізації об’єкта

  def initialize(make, model)

    @make = make      # Змінна екземпляра

    @model = model

  end

  # Метод для виведення інформації

  def details

    “This car is a #{@make} #{@model}.”

  end

end
```

Приклад створення класу в Рубі

2. Як працюють гетери та сетери? Що таке attr_reader, attr_writer та attr_accessor?(5 балів)

Гетери та сетери

Гетери та сетери — це методи, які дозволяють отримувати getter та змінювати setter значення змінних екземпляра класу. У Ruby змінні екземпляра @variable є приватними за замовчуванням, тому прямий доступ до них неможливий.

```
class Person
  # Геттер
  def name
    @name
  end
  # Сетер
  def name=(value)
    @name = value
  end
end
person = Person.new
person.name = “Alice”   # Виклик сетера
puts person.name        # Виклик гетера => “Alice”
```

attr_reader - творює гетер для змінної. Використовується, якщо потрібно лише читати значення.

attr_writer - створює сетер для змінної. Використовується, якщо потрібно лише змінювати значення.

attr_accessor - створює і гетер, і сетер для змінної. Це найчастіше використовуваний варіант.

Метод	Геттер	Сетер	Використання
attr_reader	✓	❌	Для читання значень
attr_writer	❌	✓	Для запису значень
attr_accessor	✓	✓	Для читання і запису значень

3. Як працює механізм виділення пам’яті під об’єкти в Ruby?(5 балів)

У Ruby пам’ять для об’єктів управляється автоматично за допомогою динамічного виділення пам’яті та системи збирача сміття

Динамічне виділення пам’яті

Коли в Ruby створюється об’єкт, система автоматично виділяє пам’ять у купі (heap memory).

- Купа — це область оперативної пам’яті, призначена для зберігання об’єктів, які створюються під час виконання програми.
- Ruby виділяє достатньо пам’яті для збереження об’єкта, включаючи всі його змінні екземпляра.

Наприклад

```
class Car
  def initialize(make, model)
    @make = make
    @model = model
  end
end


car = Car.new(“Toyota”, “Corolla”)  # Створення об’єкта
```

Рубі виділяє пам’ять для об’єкта car у купі

Системи збирача сміття

Ruby використовує збирання сміття (Garbage Collection), щоб автоматично звільняти пам’ять від об’єктів, які більше не використовуються.

- Якщо об’єкт не має посилань або не доступний із жодної частини програми, він вважається "сміттям" і його пам’ять звільняється.
- GC працює автоматично.

```
car = Car.new(“Toyota”, “Corolla”)  # Створення об’єкта

car = nil                          # Об’єкт більше не використовується

# GC автоматично звільнить пам’ять
```

4. Поясніть, як можна змінити стандартну поведінку методу за допомогою метапрограмування.(5 балів)

Метапрограмування у Ruby дозволяє писати код, який змінює або створює інший код під час виконання програми. Це потужний інструмент, що дозволяє змінювати стандартну поведінку методів за допомогою таких прийомів, як переозначення методів, використання method_missing, або додавання методів динамічно.

Ruby дозволяє повторно визначити вже існуючий метод, змінюючи його стандартну поведінку.

```
class String

  # Переозначення стандартного методу ‘reverse’

  def reverse

    “This method has been overridden!”

  end

end
```

```
puts “hello”.reverse  # => “This method has been overridden!”
```

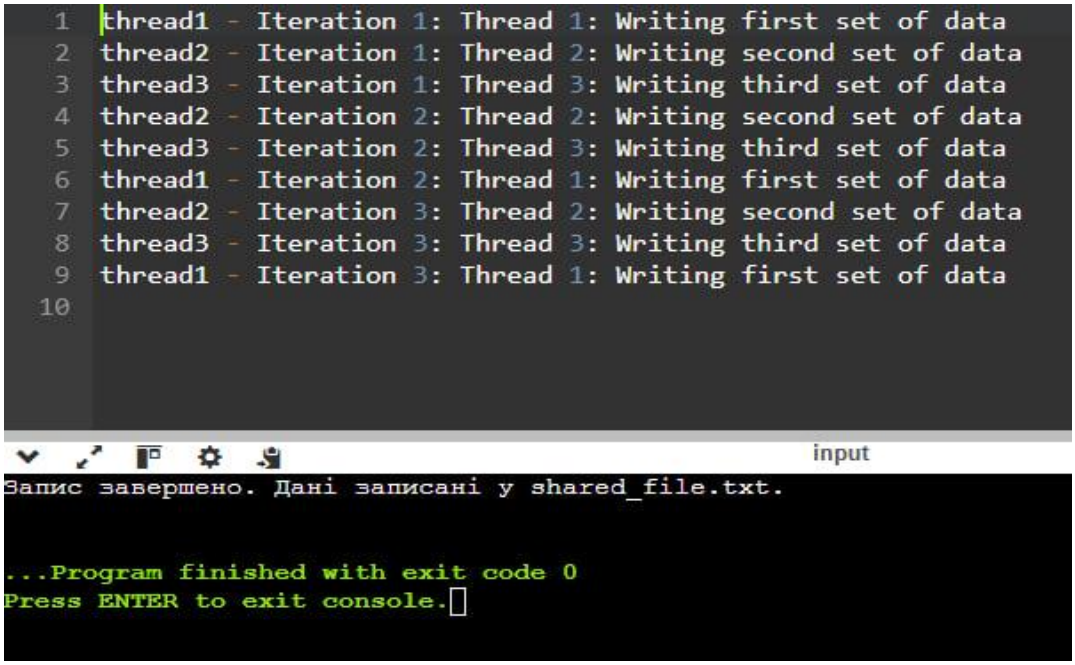
Метод reverse для класу String більше не повертає перевернутий рядок, а виконує нову логіку.

5. (Практичне завдання) Реалізуйте програму, яка:Запускає три потоки, кожен з яких записує різні дані у спільний файл. Використовує механізм блокування для уникнення конфліктів. (20 балів).

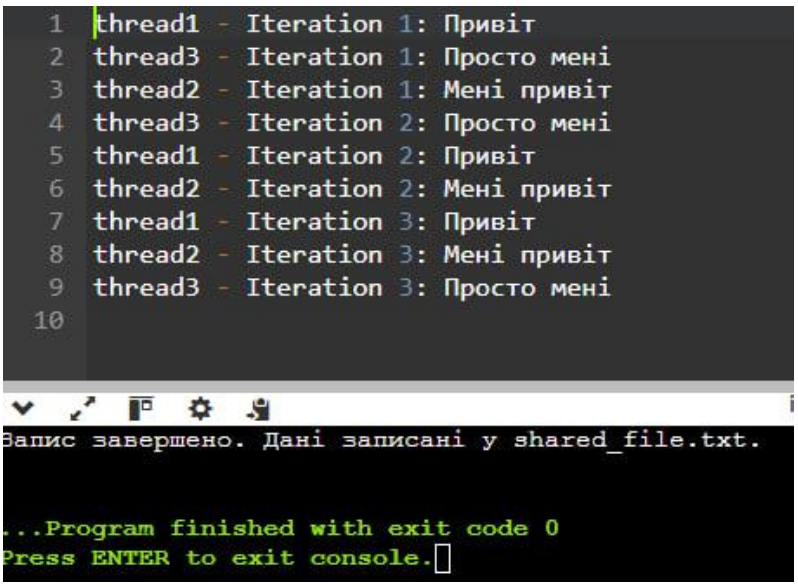
```ruby
require 'thread'

# Ім'я файлу для запису
file_name = "shared_file.txt"

# Блокування для уникнення конфліктів
mutex = Mutex.new

# Дані, які будуть записуватися потоками
data_to_write = {
  thread1: "Thread 1: Writing first set of data\n",
  thread2: "Thread 2: Writing second set of data\n",
  thread3: "Thread 3: Writing third set of data\n"
}

# Очищення файлу перед початком роботи
File.open(file_name, "w") {}

# Потоки
threads = []

data_to_write.each do |thread_name, data|
  threads << Thread.new do
    3.times do |i| # Повторюємо запис для наочності
      mutex.synchronize do
        File.open(file_name, "a") do |file|
          file.puts("#{thread_name} - Iteration #{i + 1}: #{data}")
        end
      end
      sleep(rand(0.1..0.5)) # Затримка для демонстрації потокової роботи
```

```ruby
    end
  end
end

# Очікуємо завершення всіх потоків
threads.each(&:join)

puts "Запис завершено. Дані записані у #{file_name}."
```

Лістинг 1 – вхідний код програми



Малюнок 1 – результати програми



Малюнок 2 – результати програми