Харківський національний університет імені В. Н. Каразіна
Факультет комп’ютерних наук
Кафедра моделювання систем і технологій

ЕКЗАМЕНАЦІЙНА РОБОТА
з дисципліни: «Мови прикладного програмування»

Виконав/ла: студент
групи КС32
Коханчук Юрій Сергійович

Перевірив: старший викладач
кафедри МСіТ
Паршенцев Богдан Володимирович

Харків
2023

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ (ЗАВДАННЯ) № 5

1. Паттерни(тип/переваги/недоліки/реалізація) Ланцюжок обов'язків Ruby (5 балів)
2. Принцип підстановки Барбари Лісков (5 балів)

3. Які такі різновиди змінних можна визначити в Ruby і яка їх роль (локальні, глобальні, інстанс змінні, змінні класу, константи)? (5 балів)

4. Які принципи об'єктно-орієнтованого програмування підтримуються в Ruby? Поясніть поліморфізм, спадкування та інкапсуляцію в контексті Ruby. (5 балів)

5. (Практичне завдання) Структури даних та алгоритми - Купа (20 балів)

1. **Ланцюжок обов’язків** – це поведінковий патерн проектування. Сенс цього патерну – це можливість передавати завдання обробникам далі. Кожен наступний патерн вирішує, чи потрібно йому обробити, чи передати наступному.

Переваги:

- Зменшує залежність між клієнтом та обробниками.
- Реалізує принцип єдиного обов’язку.
- Реалізує принцип відкритості/закритості.

Недоліки:

- Подія може залишатися необробленою

Реалізація створюється таким чином

1. Створення модуля(інтерфейса) чи абстрактного обробника(такий варіант і використано в коді), також я зробив метод отримання наступного елемента, що по суті своєю є структурою LinkedList. У конкретних реалізаціях позначено функції, щоб було зрозуміло, який обробник обробив яку подію. В принципі, можна запропонувати вирішення єдиного недоліку, створенням stub-обробника в кінці ланцюга, який би чи видаляв останній, чи виводив би інформацію та сповіщав би про оброблення чи його неможливість. [реалізаця знаходиться у файлі task1.rb]

```
Client: Who wants a Nut?
  Squirrel: I'll eat the Nut
Client: Who wants a Banana?
  Monkey: I'll eat the Banana
Client: Who wants a Cup of coffee?
  Cup of coffee was left untouched.

Subchain: Squirrel > Dog

Client: Who wants a Nut?
  Squirrel: I'll eat the Nut
Client: Who wants a Banana?
  Banana was left untouched.
Client: Who wants a Cup of coffee?
  Cup of coffee was left untouched.
Process finished with exit code 0
```

Рис. 1 – виконання програми для завд.1

2. **Принцип підстановки Барбари Лісков**

Принцип підстановки Барбари Лісков – це метод перевірки правильності проектування системи із наслідуваних класів. Оригінальне формулювання звучало таким чином:

Нехай q(x) є властивістю правильною для об'єктів деякого типу T . Тоді q(y) також має бути правильним для об'єктів у типу , де S — підтип типу T.

Простіше формулювання таке: наслідуючий клас повинен доповнювати, а не заміщати функціонал класу. Якщо є клас T та S < T, тоді за використання класу T замість S, не повинно бути жодних структурних змін у програмі. Гарним прикладом реалізації такого є вбудовані класи в Java, як абстрактний список, ArrayList, LinkedList і так далі.

3. **Які такі різновиди змінних можна визначити в Ruby і яка їх роль (локальні, глобальні, інстанс змінні, змінні класу, константи)**

Локальні змінні – це такі змінні, що існують лише у полі зору функції чи блока, використовуються у якихось обчисленнях чи зберіганні якихось тимчасових значень.

Глобальні змінні : видно у всьому коді програми, починаються з $
Інстанс-змінні:

Область видимості: Інстанс-змінні приналежать об'єкту і видимі в межах цього об'єкта. Починаються з символу @.

Змінні класу:
Область видимості: Змінні класу належать класу і є спільними для всіх об'єктів цього класу. Починаються з @@.

Константи:
Константи є глобальними, але їхній доступ обмежений в межах класу або модулю, де вони визначені. Пишуться великими літерами.

4. Які принципи об'єктно-орієнтованого програмування підтримуються в Ruby? Поясніть поліморфізм, спадкування та інкапсуляцію в контексті Ruby.

Поліморфізм методів: В Ruby методи можуть мати однакові імена в різних класах, і вони викликаються в залежності від типу об'єкта, який викликає метод. Наприклад:

```
class Cat
  def sound
    puts "Meow"
  end
end

class Dog
  def sound
    puts "Woof"
  end
end

cat = Cat.new
dog = Dog.new

cat.sound  # Виведе "Meow"
dog.sound  # Виведе "Woof"
```

Ruby підтримує **спадкування класів**, що дозволяє одному класу успадковувати властивості та методи іншого класу.

```
class Animal
  def speak
    puts "Animal speaks"
  end
end
```

```
class Dog < Animal
  def speak
    puts "Dog barks"
  end
end

animal = Dog.new
animal.speak  # Виведе "Dog barks"
```

Інкапсуляція в Ruby дозволяє обмежувати доступ до властивостей та методів класу. Використовуються ключові слова private та protected.

```
class MyClass
  def public_method
    puts "This is a public method"
  end

  private

  def private_method
    puts "This is a private method"
  end
end

obj = MyClass.new
obj.public_method  # Доступно
obj.private_method # Недоступно, викличе помилку
```

4. **(Практичне завдання) Структури даних та алгоритми - Купа (20 балів)**

Купа – це структура даних, що по суті є більш вузькоспеціалізованим деревом. Існує багато типів куп, таких як біноміальна, фібоначі, двійкова, н-арна. На цій структурі часто засновується дуже ефективні алгоритми, як пошук найбільших/найменших значень або пірамідальне сортування(heap sort).
Купу можна відобразити мінімум двома способами: масивом та вузлами

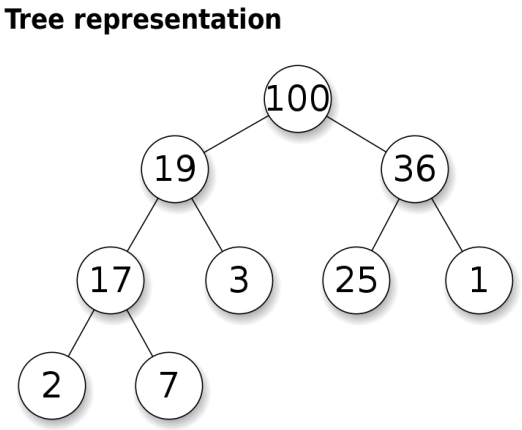

Рис. 2 – Дерево у виді графа

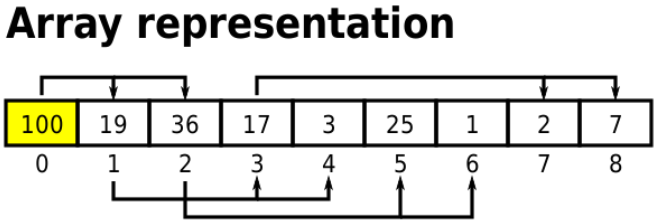

Рис. 3 – Дерево у виді масива

Я написав програму, що створює таку купу(MAX heap) для масиву чисел.

```
class HeapNode
  attr_accessor :value
  attr_accessor :inheritor_a
  attr_accessor :inheritor_b
  def initialize(value, heir,heir2)
    @value = value
    @inheritor_a = heir
    @inheritor_b = heir2
  end
end
```

Рис. 4 – Вузол купи

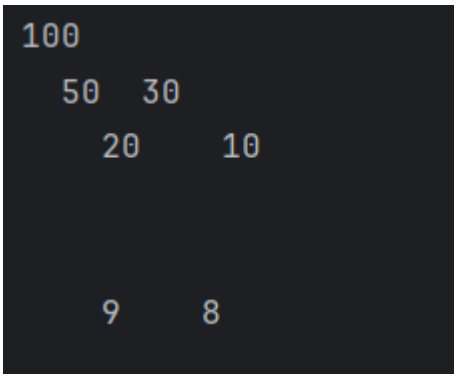

Рис. 5 – Купа

Зсув по X визначає глибину дерева.

100 – 50 (20,10) – 30(9,8)