Контрольна робота. Второва Яна Вячеславівна КС-31.

Варіант 3

Мови Прикладного Програмування

1.

DSL, або мова, специфічна для області, - це мова програмування або набір мовних конструкцій, призначений для вирішення конкретних завдань або задач у конкретній області діяльності. Існують два основних види DSL:

Екстерна DSL (зовнішня): Це власне мова програмування, яка існує окремо від загальної мови програмування. Її створюють або адаптовують для використання у конкретній галузі.

Інтерна DSL (вбудована): Це використання конструкцій загальної мови програмування для створення спеціалізованого мовного інтерфейсу для конкретного домену.

DSL дозволяє програмістам виражати ідеї і концепції в мові, близькій до термінів та парадигм, характерних для конкретної галузі. Це полегшує взаємодію між розробниками та фахівцями в галузі, сприяє зрозумінню коду та робить його більш читабельним та зорієнтованим на вирішення конкретних завдань.

2. include та extend використовуються в Ruby для роботи з модулями, але вони використовуються в різних контекстах та мають різний ефект.

include:

Контекст: Зазвичай використовується всередині визначення класу.

Ефект: Додає методи модуля як методи екземпляра класу. Це означає, що методи стають доступними для екземплярів класу.

Приклад:

```
module MyModule
  def my_method
    puts "Це метод з MyModule"
  end
end

class MyClass
  include MyModule
end

obj = MyClass.new
obj.my_method  # Це працює через 'include'
```

extend:

Контекст: Зазвичай використовується на рівні класу чи всередині екземпляра об'єкта.

Ефект: Додає методи модуля як методи класу для класу або як методи екземпляра для об'єкта (залежно від контексту використання).

Приклад:

```
module MyModule
  def my_method
    puts "Це метод з MyModule"
  end
end

class MyClass
  extend MyModule
end
```

MyClass.my_method # Це працює через 'extend'

Отже, include використовується для зроблення методів модуля доступними як методи екземпляра у класі, тоді як extend використовується для зроблення методів модуля доступними як методи класу у класі чи як методи екземпляра для об'єкта.

Практична Частина:

Код:

```
mutex = Mutex.new
turn = 1

thread1 = Thread.new do
  (1..10).each do |i|
    mutex.synchronize do
      while turn != 1
        sleep 0.1
      end

      puts "Thread 1: #{i}"
      turn = 2
    end
  end
end

thread2 = Thread.new do
  (1..10).each do |i|
    mutex.synchronize do
      while turn != 2
        sleep 0.1
      end
```

```
      puts "Thread 2: #{i}"
      turn = 1
    end
  end
end

thread1.join
thread2.join
```