Ruby Programming Assessment 3.2

21MIS1021 VIMAL KUMAR S

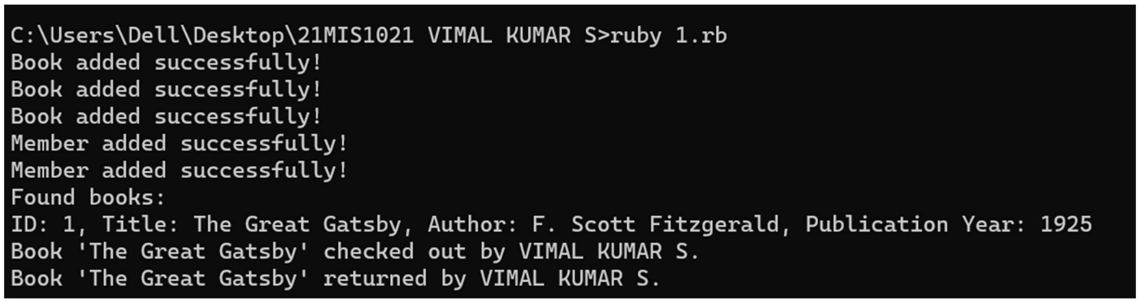
1. Ruby code for Develop a ruby program for a library management system using class and methods. The system needs to handle multiple functionalities such as adding books, searching for books, checking out books, and managing member records. Design and implement a solution that satisfies the following requirements: • Each book should have a unique identifier, title, author, and publication year and further the program should also handle member records. • Each member should have a unique identifier, name, contact information, and a list of books they have checked out.
   CODE:

```ruby
class Book
  attr_reader :id, :title, :author, :publication_year

  def initialize(id, title, author, publication_year)
    @id = id
    @title = title
    @author = author
    @publication_year = publication_year
  end
end

class Member
  attr_reader :id, :name, :contact_info, :checked_out_books

  def initialize(id, name, contact_info)
    @id = id
    @name = name
    @contact_info = contact_info
    @checked_out_books = []
  end

  def check_out_book(book)
    @checked_out_books << book
  end

  def return_book(book)
    @checked_out_books.delete(book)
  end
end

class Library
```

```
  attr_reader :books, :members

  def initialize
    @books = []
    @members = []
  end

  def add_book(id, title, author, publication_year)
    book = Book.new(id, title, author, publication_year)
    @books << book
    puts "Book added successfully!"
  end

  def search_book(title)
    found_books = @books.select { |book| book.title.downcase.include?(title.downcase) }
    if found_books.empty?
      puts "No books found with the given title."
    else
      puts "Found books:"
      found_books.each do |book|
        puts "ID: #{book.id}, Title: #{book.title}, Author: #{book.author}, Publication Year:
#{book.publication_year}"
      end
    end
  end

  def add_member(id, name, contact_info)
    member = Member.new(id, name, contact_info)
    @members << member
    puts "Member added successfully!"
  end

  def check_out_book(member_id, book_id)
    member = find_member(member_id)
    book = find_book(book_id)

    if member.nil?
      puts "Member with ID #{member_id} not found."
    elsif book.nil?
      puts "Book with ID #{book_id} not found."
    else
      member.check_out_book(book)
      puts "Book '#{book.title}' checked out by #{member.name}."
    end
  end

  def return_book(member_id, book_id)
```

```
    member = find_member(member_id)
    book = find_book(book_id)

    if member.nil?
      puts "Member with ID #{member_id} not found."
    elsif book.nil?
      puts "Book with ID #{book_id} not found."
    elsif !member.checked_out_books.include?(book)
      puts "This book is not checked out by #{member.name}."
    else
      member.return_book(book)
      puts "Book '#{book.title}' returned by #{member.name}."
    end
  end

  private

  def find_member(member_id)
    @members.find { |member| member.id == member_id }
  end

  def find_book(book_id)
    @books.find { |book| book.id == book_id }
  end
end

# Example usage of the Library management system

library = Library.new

# Adding books
library.add_book(1, "The Great Gatsby", "F. Scott Fitzgerald", 1925)
library.add_book(2, "To Kill a Mockingbird", "Harper Lee", 1960)
library.add_book(3, "1984", "George Orwell", 1949)

# Adding members
library.add_member(1, "VIMAL KUMAR S", "vimal.s@gmail.com")
library.add_member(2, "PRIYANKA", "priyanka@gmai.com")

# Searching for a book
library.search_book("gatsby")

# Checking out a book
library.check_out_book(1, 1)

# Returning a book
library.return_book(1, 1)
```

OUTPUT:



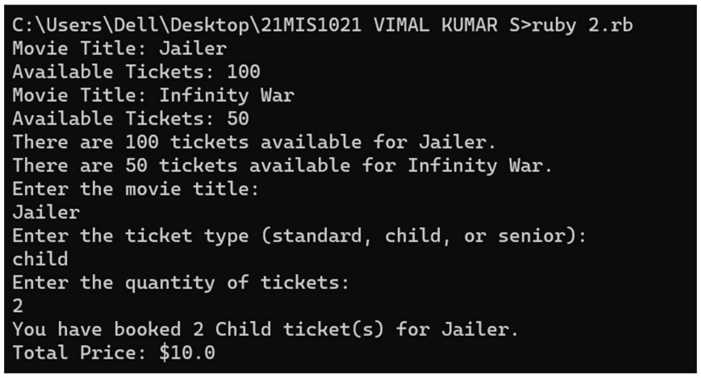
```
C:\Users\Dell\Desktop\21MIS1021 VIMAL KUMAR S>ruby 1.rb
Book added successfully!
Book added successfully!
Book added successfully!
Member added successfully!
Member added successfully!
Found books:
ID: 1, Title: The Great Gatsby, Author: F. Scott Fitzgerald, Publication Year: 1925
Book 'The Great Gatsby' checked out by VIMAL KUMAR S.
Book 'The Great Gatsby' returned by VIMAL KUMAR S.
```

2. Ruby code for You are developing a ticketing system for a cinema. Create a ruby program that models a movie class. The class should have methods to display movie details, check ticket availability, book tickets, and calculate the total ticket price based on different ticket types. Implement proper error handling for scenarios such as sold-out shows or invalid ticket selections.

CODE:

```
class Movie
  TICKET_PRICES = {
    standard: 10.0,
    child: 5.0,
    senior: 7.5
  }.freeze

  attr_reader :title, :available_tickets

  def initialize(title, total_tickets)
    @title = title
    @total_tickets = total_tickets
    @available_tickets = total_tickets
  end

  def display_movie_details
    puts "Movie Title: #{@title}"
    puts "Available Tickets: #{@available_tickets}"
  end

  def check_ticket_availability
    if @available_tickets > 0
      puts "There are #{@available_tickets} tickets available for #{@title}."
    else
      puts "Sorry, #{@title} is sold out."
    end
  end

  def book_tickets(ticket_type, quantity)
    if @available_tickets >= quantity
      price = calculate_ticket_price(ticket_type, quantity)
      @available_tickets -= quantity
      puts "You have booked #{quantity} #{ticket_type.capitalize} ticket(s) for #{@title}."
      puts "Total Price: $#{price}"
    else
      puts "Sorry, there are not enough tickets available for #{@title}."
    end
  end

  private

  def calculate_ticket_price(ticket_type, quantity)
    price_per_ticket = TICKET_PRICES[ticket_type.to_sym]
    price_per_ticket * quantity
  end
end

# Example usage of the Movie class

# Creating movie objects
movie1 = Movie.new("The Avengers", 100)
movie2 = Movie.new("Toy Story 4", 50)

# Display movie details
movie1.display_movie_details
movie2.display_movie_details

# Check ticket availability
movie1.check_ticket_availability
movie2.check_ticket_availability

# Booking tickets
puts "Enter the movie title:"
title = gets.chomp

puts "Enter the ticket type (standard, child, or senior):"
ticket_type = gets.chomp

puts "Enter the quantity of tickets:"
quantity = gets.chomp.to_i

if title.downcase == movie1.title.downcase
  movie1.book_tickets(ticket_type, quantity)
elsif title.downcase == movie2.title.downcase
  movie2.book_tickets(ticket_type, quantity)
else
  puts "Movie not found."
end
```

OUTPUT:



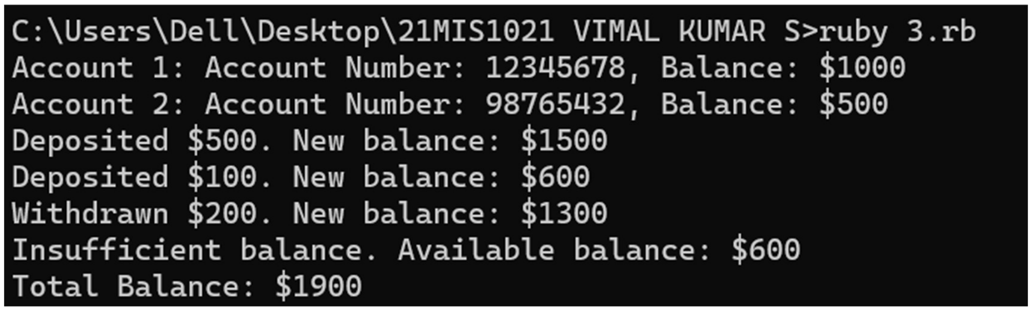
```
C:\Users\Dell\Desktop\21MIS1021 VIMAL KUMAR S>ruby 2.rb
Movie Title: Jailer
Available Tickets: 100
Movie Title: Infinity War
Available Tickets: 50
There are 100 tickets available for Jailer.
There are 50 tickets available for Infinity War.
Enter the movie title:
Jailer
Enter the ticket type (standard, child, or senior):
child
Enter the quantity of tickets:
2
You have booked 2 Child ticket(s) for Jailer.
Total Price: $10.0
```

3. Create a ruby program that simulates a bank account management system. The program should have a Bankaccount class with the following features: • The Bankaccount class should have instance variables for account_number and balance. These variables should be accessible through accessor methods. • Implement an instance method called deposit that takes an amount as input and adds it to the account's balance. • Implement an instance method called withdraw that takes an amount as input and subtracts it from the account's balance, if the balance is sufficient. If the balance is not sufficient, display an appropriate error message. • Implement a class method called total_balance that calculates and returns the total balance of bank account.

CODE:

```
class BankAccount
  attr_accessor :account_number, :balance

  @@total_balance = 0

  def initialize(account_number, balance = 0)
    @account_number = account_number
    @balance = balance
    @@total_balance += balance
  end

  def deposit(amount)
    @balance += amount
    @@total_balance += amount
    puts "Deposited $#{amount}. New balance: $#{@balance}"
  end

  def withdraw(amount)
    if @balance >= amount
      @balance -= amount
      @@total_balance -= amount
      puts "Withdrawn $#{amount}. New balance: $#{@balance}"
    else
      puts "Insufficient balance. Available balance: $#{@balance}"
    end
  end

  def self.total_balance
    @@total_balance
  end
end

# Example usage of the BankAccount class

# Creating bank account objects
account1 = BankAccount.new("12345678", 1000)
account2 = BankAccount.new("98765432", 500)

# Accessing account details
puts "Account 1: Account Number: #{account1.account_number}, Balance:
$#{account1.balance}"
puts "Account 2: Account Number: #{account2.account_number}, Balance:
$#{account2.balance}"

# Depositing into accounts
account1.deposit(500)
account2.deposit(100)

# Withdrawing from accounts
account1.withdraw(200)
account2.withdraw(700)

# Checking total balance
puts "Total Balance: $#{BankAccount.total_balance}"
```

OUTPUT:



```
C:\Users\Dell\Desktop\21MIS1021 VIMAL KUMAR S>ruby 3.rb
Account 1: Account Number: 12345678, Balance: $1000
Account 2: Account Number: 98765432, Balance: $500
Deposited $500. New balance: $1500
Deposited $100. New balance: $600
Withdrawn $200. New balance: $1300
Insufficient balance. Available balance: $600
Total Balance: $1900
```

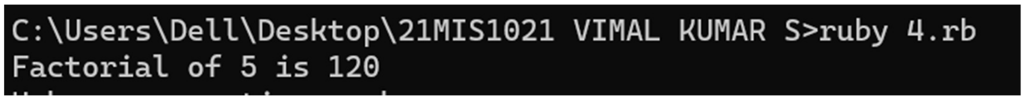
4. Create a ruby program that implements a dynamic method dispatcher using the method_missing method. The program should have a class called DynamicDispatcher that allows the user to define and call methods dynamically. The DynamicDispatcher class should have the following functionalities: • When an undefined method is called on an instance of DynamicDispatcher, the method_missing method should be invoked. • Inside the method_missing method, check if the method name starts with "calculate_". • If the method name starts with "calculate_", extract the operation name from the method name (e.g., if

the method name is "calculate_factorial", the operation name is "factorial"). • Perform the corresponding calculation based on the operation name (factorial). • Display the result of the calculation.

CODE:

```
class DynamicDispatcher
  def method_missing(method_name, *args)
    if method_name.to_s.start_with?("calculate_")
      operation_name = method_name.to_s.split("_", 2).last
      calculate(operation_name, *args)
    else
      super
    end
  end

  private

  def calculate(operation_name, *args)
    case operation_name
    when "factorial"
      calculate_factorial(*args)
    else
      puts "Unknown operation: #{operation_name}"
    end
  end

  def calculate_factorial(n)
    result = (1..n).inject(:*) || 1
    puts "Factorial of #{n} is #{result}"
  end
end

# Example usage of the DynamicDispatcher class

dispatcher = DynamicDispatcher.new

dispatcher.calculate_factorial(5)  # Factorial of 5 is 120
dispatcher.calculate_unknown(10)   # Unknown operation: unknown
dispatcher.unknown_method(3)       # NoMethodError
```

OUTPUT:



```
C:\Users\Dell\Desktop\21MIS1021 VIMAL KUMAR S>ruby 4.rb
Factorial of 5 is 120
```

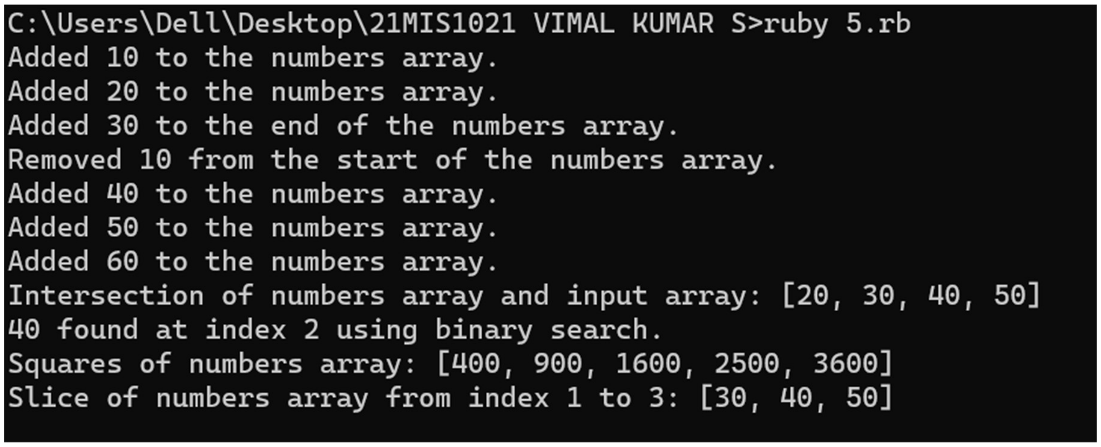
5. Write a ruby program that demonstrates the usage of various array methods. Implement a class called ArrayOperations with the following features: • The ArrayOperations class should have an instance variable called numbers which is an array initially empty. • Implement a method called add_number that takes an integer as input and adds it to the numbers array.

• Implement a method called add_to_end that takes an integer as input and adds it to the end of the numbers array. • Implement a method called remove_from_start that removes the first element from the numbers array and returns it. • Implement a method called get_intersection that takes an array as input and returns a new array containing the intersection of the numbers array and the input array (common elements). • Implement a method called binary_search that takes an integer as input and performs a binary search on the numbers array. If the integer is found, return its index; otherwise, return nil. • Implement a method called collect_squares that returns a new array containing the squares of each element in the numbers array. • Implement a method called get_slice that takes two indices as input and returns a new array containing the elements from the numbers array between the specified indices. Create an instance of the ArrayOperations class and demonstrate the usage of all the implemented methods by performing various operations on the numbers array.

CODE:

```
class ArrayOperations
  attr_reader :numbers

  def initialize
    @numbers = []
  end

  def add_number(number)
    @numbers << number
    puts "Added #{number} to the numbers array."
  end

  def add_to_end(number)
    @numbers.push(number)
    puts "Added #{number} to the end of the numbers array."
  end

  def remove_from_start
    number = @numbers.shift
    puts "Removed #{number} from the start of the numbers array."
    number
  end

  def get_intersection(array)
    intersection = @numbers & array
    puts "Intersection of numbers array and input array: #{intersection}"
    intersection
  end

  def binary_search(number)
    sorted_numbers = @numbers.sort
    low = 0
    high = sorted_numbers.length - 1
    while low <= high
      mid = (low + high) / 2
      if sorted_numbers[mid] == number
        puts "#{number} found at index #{mid} using binary search."
        return mid
      elsif sorted_numbers[mid] < number
        low = mid + 1
      else
        high = mid - 1
      end
    end

    puts "#{number} not found using binary search."
    nil
  end

  def collect_squares
    squares = @numbers.map { |number| number**2 }
    puts "Squares of numbers array: #{squares}"
    squares
  end

  def get_slice(start_index, end_index)
    slice = @numbers[start_index..end_index]
    puts "Slice of numbers array from index #{start_index} to #{end_index}: #{slice}"
    slice
  end
end

# Example usage of the ArrayOperations class

array_ops = ArrayOperations.new

array_ops.add_number(10)
array_ops.add_number(20)
array_ops.add_to_end(30)
array_ops.remove_from_start

array_ops.add_number(40)
array_ops.add_number(50)
array_ops.add_number(60)

array_ops.get_intersection([20, 30, 40, 50])
array_ops.binary_search(40)

array_ops.collect_squares
array_ops.get_slice(1, 3)
```

OUTPUT:



```
C:\Users\Dell\Desktop\21MIS1021 VIMAL KUMAR S>ruby 5.rb
Added 10 to the numbers array.
Added 20 to the numbers array.
Added 30 to the end of the numbers array.
Removed 10 from the start of the numbers array.
Added 40 to the numbers array.
Added 50 to the numbers array.
Added 60 to the numbers array.
Intersection of numbers array and input array: [20, 30, 40, 50]
40 found at index 2 using binary search.
Squares of numbers array: [400, 900, 1600, 2500, 3600]
Slice of numbers array from index 1 to 3: [30, 40, 50]
```