3. Lab 3: Stacks & Queues

3.1. Objectives

- Know how to use the data structure Stack for solving real problems.

3.2. Problem 1: Simple stack application

Write a program to

- Convert a decimal number and convert it to octal form.
- Concatenate two stacks.
- Determine if the contents of one stack are identical to that of another.

3.3. Problem 3: Undo and redo

Write a class *SpecialArray* that has:

- (8 points) an array of 20 random values
- (8 points) a function to update the value at a position in the array.
- (8 points) a function to undo the updating.
- (8 points) a function to redo the updating.
- (8 points) a function to display content of the array.

Hint: use two stacks to store the array after each operation

3.4. QueueApp.java

- Write a method to display the queue array and the front and rear indices. Explain how wraparound works.
- Write a method to display the queue (loop from 1 to nItems and use a temporary front for wraparound).
- Display the aray, the queue, and the front and rear indices.
- Insert fewer items or remove fewer items and investigate what happens when the queue is empty or full.
- Extend the insert and remove methods to deal with a full and empty queue.
- Add processing time to the queue. Create a new remove method that removes item N after N calls to the method.
- Simulate a queue of customers each one served for a random amount of time. Investigate how simulation is affected by:
    - the size of the queue
    - the range of time for wich each customer is served
    - the rate at which customers arrive at the queue

3.5. ReverseApp.java

- Create a stack of objects of class Person and use to reverse a list of persons.

3.6. PriorityQApp.java

- Write a method to display the queue and use to trace the queue operation.
- Modify the insert method to insert the new item at the rear. Compare this queue with QueueApp.java. Which one is more efficient?
- Use a priority queue instead of an ordinary one in the simulation experiments described above.

3.7. Exercise: Customer Service Center (Queue)

In a customer service center, requests are processed in the order they are received. However, VIP customers should always be served before regular customers. You need to manage two types of customers—regular and VIP—using a queue system. Explain the data structure, algorithm, and time complexity of your solution.

Problem Description:
You need to implement a customer service queue with the following operations:

1. enqueue(customer_name, is_vip): Add a customer to the queue. VIP customers are always served first, followed by regular customers.
2. serve_customer(): Serve the next customer in line (VIP customers first, then regular customers).
3.

Input:
A series of operations (e.g., enqueue("Alice", False), serve_customer()).

Output:

The customer that is served after each serve_customer() operation.

Example:

Input
enqueue("Alice", False)
enqueue("Bob", True)
enqueue("Charlie", False)
serve_customer()
serve_customer()
serve_customer()
Output:
Serve Bob
Serve Alice
Serve Charlie

Key Challenges:

1. Queue Operations: Students need to implement two queues (one for regular customers, one for VIPs) and manage the order of service.
2. Priority Handling: VIP customers must always be served before regular customers, even if regular customers have been waiting longer.
3. Efficient Queue Management: Ensuring that operations such as enqueueing and serving customers are efficient, especially with a mix of VIP and regular customers.