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:

- an array of 20 random values
- a function to update the value at a position in the array.
- a function to undo the updating.
- a function to redo the updating.
- 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: Task Scheduling (Queue)

In a task scheduling system, tasks arrive at a processing unit in the order they were created, and they must be processed in the same order (First In, First Out). Some tasks take longer than others to complete, and occasionally, new high-priority tasks arrive that need to be processed before regular tasks. High-priority tasks are always processed immediately, but regular tasks continue in the original order after the high-priority tasks are handled.
Your task is to simulate this task scheduling system using a queue and priority queue.
Problem Description:
You need to implement a task scheduling system with the following operations:

1. add_task(task_name, is_priority): Add a task to the queue. If is_priority is True, it's a high-priority task and should be processed before regular tasks.
2. process_task(): Process the next task in the queue (priority tasks first, followed by regular tasks). Output the task being processed.

Input:

- A series of operations (e.g., add_task("task1", False), process_task()).

Output:

- The task that is processed after each process_task() operation.

Example:

Input:
add_task("task1", False)
add_task("task2", False)
add_task("urgent_task", True)
process_task()
process_task()
process_task()

Output:
Process urgent_task
Process task1
Process task2

Key Challenges:

1. **Queue Operations:** Implementing task scheduling with a regular queue for normal tasks and a priority queue for urgent tasks.
2. **Priority Management:** Students must handle two different types of tasks and process them in the right order.
3. **Edge Cases:** Consider cases where all tasks are high-priority or no tasks are available to process.