Python Lab 3 – Implementing Queue Data Structures

1. Overview

In this lab, you will implement two versions of a **Queue** data structure in Python.

A **Queue** is a linear data structure that follows the **First In, First Out (FIFO)** principle: - The first element inserted is the first one to be removed. - Real-life example: people standing in a line (queue) at a bank or supermarket — the first person to arrive is served first.

You will start with a simple queue, then build a more advanced, named and size-limited queue that can be saved to and loaded from a file.

---

2. Part 1 – Basic Queue Class

Goal

Create a Python class that represents a simple FIFO queue and supports basic operations.

Requirements

Create a class (e.g. Queue) with the following methods:

1. insert(value)
    - Adds a new element **at the rear** (end) of the queue.
2. pop()
    - Removes and returns the element **at the front** of the queue.
    - If the queue is **empty**, it should:
        - Print a clear warning message (e.g. "Warning: cannot pop from an empty queue.")
        - Return None
3. is_empty()
    - Returns True if the queue has no elements, otherwise returns False.

Notes & Hints

- You may use a Python list internally to store the elements.

- Think about which end of the list represents the “front” and which is the “rear” so that operations remain clear and consistent.

---

3. Part 2 – Advanced Named & Bounded Queue

In this part, you will build a more advanced queue class that:

- Has a **name**.
- Has a **fixed maximum size**.
- Tracks **all created queue instances**.
- Can be **saved to** and **loaded from** a file.

3.1. New Class Requirements

Create another queue class (you may call it e.g. NamedQueue or BoundedQueue) that has the same behavior as the basic queue in Part 1, **plus** the following changes:

A. Constructor with Name and Size

- The constructor should accept:
    - name (string): a unique name for this queue.
    - size (integer): the maximum number of elements the queue can hold.
- Example: python q = NamedQueue(name="students_queue", size=10)

B. Size Limitation and Custom Exception

- If the user tries to insert more elements than the specified size:
    - The class must raise a **custom exception** called QueueOutOfRangeException.
- You must:
    - Define a new exception class:
- class QueueOutOfRangeException(Exception):
- pass
    - Use this exception in the insert method when the queue is full.

C. Tracking All Queue Instances

- The class should keep track of **all instances** created from it.
- Use a **class-level attribute** (e.g. a dictionary) to map from queue name to the queue instance.
    - Example structure:
- _instances = {

  "students_queue": <NamedQueue object>,

  "tickets_queue": <NamedQueue object>,

  ...

  }
- Provide a way to get any created queue by its name (for example, a @classmethod like get_by_name(name) that returns the corresponding queue instance or None if it does not exist).

D. Saving and Loading Queues

The class should support saving and loading all created instances:

1. @classmethod save(cls, filename)
    - Saves **all currently created queue instances** to a file (e.g. JSON file).
    - For each queue, you should save at least:
        - Its name
        - Its size
        - Its current elements (in the correct FIFO order)
2. @classmethod load(cls, filename)
    - Loads queues from the given file and **recreates** the queue instances.
    - After loading:
        - The class-level registry of instances should be updated.
        - You should be able to access the queues again by their names (e.g. using get_by_name).

Suggested File Format

You can use JSON to store the queues. Example JSON structure:

```
{
  "students_queue": {
    "size": 10,
    "items": [1, 2, 3]
  },
  "tickets_queue": {
    "size": 5,
    "items": ["A", "B"]
  }
}
```

(You do **not** need to use exactly this format, but your choice should be clear and consistent.)

---

4. Deliverables

By the end of this lab, you should have:

1. **Part 1** – A working basic Queue class with:
    - insert(value)
    - pop()
    - is_empty()
2. **Part 2** – A working advanced queue class (e.g. NamedQueue) that:
    - Has a name and maximum size (set in the constructor).
    - Raises QueueOutOfRangeException when inserting into a full queue.
    - Tracks all created instances and can retrieve them by name.
    - Can save all queues to a file and load them back using save and load class methods.

Make sure your code is clean, well-commented, and easy to read.