Assignment B: Stacks, Queues & Priority (Kotlin)

Due: Wednesday, Oct 15, 10:00 pm

Overview

You'll implement a **Stack** (LIFO), a **Linked Queue** (FIFO), and a **Priority Queue** using a binary *min-heap* to simulate different layoff policies. You'll also compute a **performance score** per employee, then reflect briefly on fairness/impact.

Given: `Employee.kt`, `Main.kt`, all unit tests, and `pom.xml`
You complete TODOs in:

- `Stack.kt`
- `LinkedQueue.kt`
- `Performance.kt` (implement one function)
- `PriorityQueue.kt` (heap + sort)

Data file: `data/assignment_b1.csv` (used by `Main.kt`)

Part 1 — Stack (LIFO)

Implement a singly-linked stack with the usual operations:

```kotlin
class Stack<T> {
  private class Node<E>(val v: E, var next: Node<E>? = null)
  private var head: Node<T>? = null
  private var n = 0

  fun push(item: T) { /* TODO */ }
  fun pop(): T { /* TODO */ }            // remove head; throw if
empty
  fun peek(): T? { /* TODO */ }          // null if empty
  fun isEmpty(): Boolean { /* TODO */ }
  fun size(): Int { /* TODO */ }
}
```

Notes:

- **push** adds at head; **pop** removes head and decrements size.
- Keep size (`n`) accurate; `peek()` returns `null` for empty.

Part 2 — Linked Queue (FIFO)

Implement a singly-linked queue with head/tail pointers:

```kotlin
class LinkedQueue<T> {
  private class Node<E>(val v: E, var next: Node<E>? = null)
  private var head: Node<T>? = null
  private var tail: Node<T>? = null
  private var n = 0

  fun enqueue(x: T) { /* TODO */ }       // add at tail
  fun dequeue(): T { /* TODO */ }        // remove head; throw if
empty
  fun peek(): T? { /* TODO */ }          // null if empty
  fun isEmpty(): Boolean { /* TODO */ }
  fun size(): Int { /* TODO */ }
}
```

Notes:

- When the queue becomes empty after `dequeue()`, set both `head` and `tail` to `null`.

Part 3 — Performance Score (exact rules)

Compute a 1..5 integer score (or `null` if insufficient signals).

Reviews (R): mean of present values among

- `selfEvaluation` (1..5)
- `peer360Feedback` (1..5)
- `managerFeedback` (1..5)
- `okr / 20` (maps 0..100 → 0..5; then used as-is in the mean)

Behavior (B):

- Productivity = `problemsFixed / problemsAssigned`.
  Skip if `problemsAssigned == 0` or missing. Value is in [0,1].
- `punctualityRate` is already in [0,1].
- Map any [0,1] signal to [1,5] via `1 + 4*v`.
- B is the average of available mapped signals (1–5 scale).

Combine:

- If both R and B exist: `0.7*R + 0.3*B`
- If only one exists: use it
- Round to nearest integer, then clamp to [1..5]
- If both missing, return `null`
  (Missing values are skipped; do **not** default them to 0.)

Skeleton:

```kotlin
fun computePerformanceScore(e: Employee): Int? {
  // Implement per rules above
}
```

Part 4 — Priority Layoffs (Min-Heap)

Build a min-heap and sort with this two-key policy:

1. **Lower performance first.** Real scores 1..5 come before `null` (insufficient data).
2. **Higher cost next.** Missing/zero cost should not boost priority.

No extra tie-breakers; exact ties may appear in any order.

Keys:

```kotlin
private fun perfKey(e: Employee): Int = e.performanceScore ?:
Int.MAX_VALUE

private fun costKey(e: Employee): Int {
  val c = e.costToCompany ?: 0
  return if (c > 0) -c else Int.MAX_VALUE   // invert cost for
min-heap; missing/zero = low priority
}
```

Comparator idea for your heap:

```kotlin
val heap = MinHeap<Employee> { a, b ->
  val ap = perfKey(a); val bp = perfKey(b)
  if (ap != bp) ap < bp else {
    val ac = costKey(a); val bc = costKey(b)
    if (ac != bc) ac < bc else false
  }
}
```

CLI: `mvn exec:java -Dexec.args=priority` prints the ordered layoff list using your heap.

How to Run & Test

Run all tests:

```shell
mvn clean test
```

Run one test class:

```shell
mvn -Dtest=com.carleton.comps.StackTest test
mvn -Dtest=com.carleton.comps.QueueTest test
mvn -Dtest=com.carleton.comps.PerformanceScoreTest test
mvn -Dtest=com.carleton.comps.PriorityQueueTest test
```

Run the app (from repo root):

```shell
mvn exec:java -Dexec.args=lifo
mvn exec:java -Dexec.args=fifo
mvn exec:java -Dexec.args=priority
mvn exec:java -Dexec.args=dump
mvn exec:java -Dexec.args=all
```

Note: Ensure Java 17 is active (e.g., `mvn -v` shows Java 17).

Submission

- Complete TODOs in `Stack.kt`, `LinkedQueue.kt`, `Performance.kt`, `PriorityQueue.kt`.

- Include a short `REFLECTION.md` (2–5 sentences) on the fairness/impact of the three policies.