Event Attention Buffer Specification

Overview: The *EventAttentionBuffer* represents a bounded priority queue with a fixed capacity. It stores items of a generic type *T*, each associated with a real-valued priority. The buffer supports insertion of items with priority-based eviction when the capacity is exceeded. It allows removal and inspection of items with either the maximum or minimum priority. The class ensures that the highest-priority items are retained when capacity constraints necessitate eviction.

Mathematical Definitions of Parameters and Sets

Capacity: $C \in N, C > 0$

The maximum number of items the buffer can hold.

Priority Space: $P = \{\mathrm{x} \in \mathrm{R} \mid 0 \leq \mathrm{x} \leq 1\}$

The set of real numbers in $[0, 1]$ representing item priorities.

Item Space: $T = All\, possible\ items\ of\ type\ T$

Buffer State: At any given time, the buffer B is a finite multiset of items with associated priorities:

$$B \subseteq P \times T, \mid B \mid \leq C$$

Each element (p, t) in B consists of a priority $p \in P$ and an item $t \in T$.

Ordering

Define the standard ordering on priorities:

$$For\ p1, p2 \in P, p1 \leq p2\ if\ and\ only\ if\ p1\ is\ less\ than\ or\ equal\ to\ p2$$

Specification EventAttentionBuffer(C)

Purpose: Initializes the buffer with a specified capacity.

Preconditions: $C > 0$

Postconditions: The buffer B is initialized to the empty set: $B = \emptyset$

The capacity of the buffer is set to C.

Methods

1. **Insert(**p**,** t**)**

Inputs:

$p \in P$: The priority of the item.

$t \in T$: The item to be inserted.

Behaviour:

```
**Case 1**: $If \ |B| < C$
    The item $(p, t)$ is added to the buffer: $B \leftarrow B \cup \{(p, t)\}$
**Case 2**: $If \ |B| = C$
    $Let \ p_{min} = min\ \{p' \mid (p', t') \in B\}$.
    **If** $p \leq p_{min}$
        $The\ item\ (p, t)\ is\ not\ added\ to\ the\ buffer.$
        $The\ buffer\ remains\ unchanged:$
        $B \leftarrow B$
    **Else** $(i.e.,\ p > pmin\ )$
        $Remove\ an\ item\ (p_{min}\ , t_{min}\ ) \in B\ with\ the\ minimum\ priority$
        $B \leftarrow B \setminus \{(p_{min}, t_{min}\ )\}$
    $Add\ the\ new\ item\ (p, t)\ to\ the\ buffer:$
    $B \leftarrow B \cup \{(p, t)\}$
```

Postconditions: The buffer size does not exceed C.

If the buffer was at capacity, the item with the lowest priority is evicted in favour of the new item only if the new item's priority is higher.

2. **RemoveMax()**

Outputs: Returns $(t_{max}\ , p_{max})$ where $t_{max} \in T\ and\ p_{max} \in P$, or null if the buffer is empty.

Behaviour:

```
$\boldsymbol{If}\ B = \emptyset$
        $Return\ null$
$\boldsymbol{Else}$
        $Let\ p_{max} \ = max\ \{p \mid (p, t) \in B\}$
        $Select\ an\ item\ (p_{max}\ , t_{max}\ ) \in B$
        $Remove\ (p_{max}\ , t_{maax}\ )\ from\ B$
        $B \leftarrow B \setminus \{(p_{max}\ , t_{max}\ )\}$
        $Return\ (t_{max}\ , p_{max}\ )$
```

Postconditions: The buffer size decreases by one if it was not empty.

3. RemoveMin()

Outputs: Returns (t_{min}, p_{min}) where $t_{min} \ \in T$ and $p_{min} \ \in P$, or null if the buffer is empty.

Behaviour:

```
$\boldsymbol{If}\ B = \emptyset$
        $Return\ null$
$\boldsymbol{Else}$
        $Let\ p_{min} \ = min\ \{p \mid (p, t) \in B\}$
        $Select\ an\ item\ (p_{min}, t_{min}) \in B$
        $Remove\ (p_{min}, t_{min})\ from\ B$
        $B \leftarrow B \setminus \{(p_{min}, t_{min})\}$
        $Return\ (t_{min}, p_{min})$
```

Postconditions: The buffer size decreases by one if it was not empty.

4. **PeekMax()**

Outputs: Returns (t_{max}, p_{max}) without modifying the buffer, or null if the buffer is empty.

Behaviour:

```
$\boldsymbol{If}\ B = \emptyset$:
    $Return\ null$
$\boldsymbol{Else}$
    $Let\ p_{max} = max\ \{p \mid (p, t) \in B\}$
    $Select\ an\ item\ (p_{max}, t_{max}) \in B$
    $Return\ (t_{max}, p_{max})$
```

Postconditions: The buffer B remains unchanged.

5. **PeekMin()**

Outputs: Returns (t_{min}, p_{min}) without modifying the buffer, or null if the buffer is empty.

Behaviour:

```
$\boldsymbol{If}\ B = \emptyset$
    $Return\ null$
$\boldsymbol{Else}$:
    $Let\ p_{min} = min\ \{p \mid (p, t) \in B\}$
    $Select\ an\ item\ (p_{min}, t_{min}) \in B$
    $Return\ (t_{min}, p_{min})$
```

Postconditions: The buffer B remains unchanged.

6. Count

Outputs: Returns the integer $\mid B \mid$, representing the number of items currently in the buffer.

Behaviour: Simply compute and return |B|

7. IsEmpty

Outputs: Returns a Boolean value

$True\ if\ B = \emptyset\ False\ otherwise$

Behaviour: Evaluate and return $B = \emptyset$

Invariants: At all times, the following conditions hold:

Capacity Constraint: $0 \leq |B| \leq C$

Item Integrity: Every element $(p, t) \in B$ satisfies $p \in P$ and $t \in T$.

No Exceeding Capacity: Insertion operations ensure that $|B| \leq C$ by evicting items if necessary.

Notes

Priority Ties: If multiple items share the same minimum or maximum priority, any one of them may be selected for removal or inspection.

Eviction Policy: During insertion when capacity is exceeded, the item with the lowest priority is evicted if the new item has a higher priority.

Item Uniqueness: The buffer allows duplicate items and priorities unless additional constraints are imposed externally.

Priority Precision: Since priorities are real numbers, care must be taken in implementation to handle floating-point precision issues.

Error Handling

Constructor Exception: If $C \leq 0$, the constructor raises an *ArgumentException*.

Operations on Empty Buffer:

The *RemoveMax*, *RemoveMin*, *PeekMax*, and *PeekMin* methods return null if the buffer is empty.

Null Items or Priorities: The mathematical specification assumes priorities are real numbers in $[0,1]$ and items are valid elements of T. Handling of null or invalid items is an implementation detail.

Implementation Considerations

Data Structures: An **Interval Heap** is used to efficiently support *FindMin*, *FindMax*, *DeleteMin*, and *DeleteMax* operations in $O(log\ |B|)$ time.

Thread Safety: The specification does not address concurrent access. Synchronization mechanisms should be implemented if thread safety is required.

Generics: The class is generic over type T, accommodating any item type.

Summary: The *EventAttentionBuffer<T>* class is a bounded priority queue that manages a collection of items with associated priorities. It ensures that:

The highest-priority items are retained when capacity constraints require eviction.

Items can be efficiently inserted, removed, and inspected based on their priorities.

The buffer respects capacity limits while maximizing the importance of stored items according to their priorities.

This mathematical specification provides a clear and precise understanding of the behaviour and constraints of the *EventAttentionBuffer<T>* class, suitable for rigorous analysis or implementation verification.