CSC 591: Software Testing | HW: 1

a. List all of the input parameters, including the state variables.

Solution:

Input Parameters:

1. `capacity` (int): The maximum number of elements that the queue can hold.
2. `X` (Object): The element to be enqueued.

State Variables:

1. `queue` (Object[]): The internal storage array for the queue.
2. `size` (int): The current number of elements in the queue.
3. `front` (int): The index of the front element in the queue.
4. `rear` (int): The index where the next element will be enqueued.

b. Define characteristics for the input parameters. Make sure you cover all input parameters.

Solution:

1. `capacity`
   - Positive integer: valid capacity
   - Zero or negative integer: invalid capacity
2. `X`
   - Any valid object (e.g., Integer, String)
   - Null object (to check if `null` is handled properly)
3. **State Variables (for the queue operations)**
   - Queue is empty
   - Queue is partially filled
   - Queue is full

c. Partition the characteristics into blocks. Choose one block per partition as the "Base" block.

Solution:

1. **Partitions for `capacity`:**

- **Valid Capacity Block (Base Block)**: Positive integers.
- **Invalid Capacity Block**: Zero or negative integers.

CSC 591: Software Testing | HW: 1

2. **Partitions for `X`:**
- **Valid Object Block (Base Block)**: Non-null objects.
- **Null Object Block**: Null value.

3. **Partitions for Queue State:**
- **Empty Queue Block (Base Block)**
- **Partially Filled Queue Block**
- **Full Queue Block**

d. Define values for each block.

Solution:

1. **Values for `capacity`:**
- **Valid Capacity Block**: 1, 5, 10
- **Invalid Capacity Block**: 0, -1

2. **Values for `X`:**
- **Valid Object Block**: 1 (Integer), "Hello" (String)
- **Null Object Block**: `null`

3. **Values for Queue State:**
- **Empty Queue Block**: When `size` is 0.
- **Partially Filled Queue Block**: When `size` is less than `capacity`.
- **Full Queue Block**: When `size` equals `capacity`.

e. Define a set of Test cases. That is, write test inputs for test cases, together with the expected output of those inputs on the enQueue method. You are not required to write JUnit tests for this problem.

Solution:

CSC 591: Software Testing | HW: 1

Test Case 1: Valid Capacity and Non-null Object

- **Inputs:**
  - `capacity`: 5
  - `X`: 10 (Integer)
- **Expected Outcome:**
  - The `enQueue` method should successfully add `10` to the queue.
  - After `enQueue(10)`, `size` should be 1 and `isEmpty` should be `false` if the queue was initially empty.

Test Case 2: Valid Capacity and Null Object

- **Inputs:**
  - `capacity`: 5
  - `X`: `null`
- **Expected Outcome:**
  - The `enQueue` method should handle `null` correctly, for example throw a null exception.

Test Case 3: Invalid Capacity (0)

- **Inputs:**
  - `capacity`: 0
  - `X`: 10 (Integer)
- **Expected Outcome:**
  - The queue should not be created (or throw an exception if capacity is invalid).
  - Attempting `enQueue(10)` should result in an error or exception indicating that the capacity is invalid.

Test Case 4: Full Queue Scenario

- **Inputs:**
  - `capacity`: 3
  - `X`: 10 (Integer)
- Steps:
  - Create `BoundedQueue` with capacity 3.
  - `enQueue(10)` three times.
- **Expected Outcome:**
  - After 3 `enQueue` operations, the queue should be full.
  - Additional `enQueue` operations should either fail or be ignored depending on the implementation.

Test Case 5: DeQueue on Empty Queue

- **Inputs:**
  - `capacity`: 5
  - `X`: 10 (Integer)
- Steps:
  - Create `BoundedQueue` with capacity 5.
  - Attempt `deQueue()` without enqueuing any elements.
- **Expected Outcome:**
  - The `deQueue` method should return `null` or throw an exception indicating that the queue is empty.

Test Case 6: Enqueuing an almost full queue.

- **Inputs:**
  - `capacity`: 1
  - `X`: 10 (Integer)
- Steps:
  - Create `BoundedQueue` with capacity 1.
  - `enQueue(10)`.
- **Expected Outcome:**
  - After `enQueue(10)` `isFull` should return `true`.