ASSIGNMENT 3 PROGRAMMING
Name : Ali Sher
Student ID : 40255236

CLASS Input<K extends Comparable<K>, V> IMPLEMENTS Comparable<Input<K, V>>:

```
PRIVATE VARIABLE key OF TYPE K
PRIVATE VARIABLE value OF TYPE V
PRIVATE VARIABLE index OF TYPE int

METHOD getIndex():
   RETURN index

METHOD setIndex(newIndex):
   SET index TO newIndex

METHOD Input(newKValue, newVValue):
   SET key TO newKValue
   SET value TO newVValue

METHOD getKey():
   RETURN key

METHOD getValue():
   RETURN value

METHOD setKey(newKValue):
   SET key TO newKValue

METHOD setValue(newValue):
   SET value TO newValue

METHOD equals(obj):
   IF obj IS THIS:
      RETURN true
   IF obj IS NULL OR obj IS NOT INSTANCE OF Input:
      RETURN false
   CAST obj TO Input<?, ?>
   RETURN (key EQUALS obj.key) AND (value EQUALS obj.value)
```

```
    METHOD print():
        RETURN STRING "(" + key + "," + value + ")"

    METHOD compareTo(other):
        RETURN key.compareTo(other.key)



CLASS ExpandableArray<K extends Comparable<K>, V>:
    PRIVATE ARRAY of Input<K, V>
    PRIVATE VARIABLE size OF TYPE int
    PRIVATE CONSTANT DefaultCapacity = 10

    CONSTRUCTOR ExpandableArray():
        CALL ExpandableArray(DefaultCapacity)

    CONSTRUCTOR ExpandableArray(starting_size):
        CREATE Array WITH CAPACITY starting_size
        SET size TO 0
        INITIALIZE all elements of Array TO null

    METHOD size():
        RETURN size

    METHOD length():
        RETURN size

    METHOD Capacity():
        RETURN LENGTH OF Array

    METHOD isEmpty():
        RETURN (size <= 0)

    METHOD clear():
        FOR i FROM 0 TO size:
            SET Array[i] TO null
        SET size TO 0

    METHOD ensureCapacity():
        IF size == Array.length - 1:
            CREATE newQueueArray WITH DOUBLE THE SIZE OF CURRENT Array
            COPY Array TO newQueueArray
            SET Array TO newQueueArray
```

```
METHOD swapIndex(index1, index2):
   IF index1 OR index2 IS OUT OF BOUNDS:
      THROW IndexOutOfBoundsException
   SWAP Array[index1] WITH Array[index2]
   UPDATE INDICES OF BOTH ELEMENTS

METHOD get(index):
   IF index IS OUT OF BOUNDS:
      RETURN null
   RETURN Array[index]

METHOD set(index, arg):
   IF arg IS null OR index IS OUT OF BOUNDS:
      THROW IllegalArgumentException OR RETURN
   IF Array[index] IS null:
      INCREMENT size
   SET Array[index] TO arg
   SET INDEX OF arg TO index

METHOD setNew(index, newK, data):
   IF index IS OUT OF BOUNDS:
      RETURN
   CREATE new Input<K, V> WITH newK AND data
   SET Array[index] TO THIS NEW Input
   INCREMENT size
   UPDATE INDEX OF THE NEW Input

METHOD remove(index):
   IF index IS OUT OF BOUNDS:
      RETURN null
   STORE Array[index] IN temp
   SET Array[index] TO null
   DECREMENT size
   RETURN temp

METHOD printArray():
   IF Array IS EMPTY:
      PRINT "{}"
      RETURN
   PRINT "{ "
   FOR i FROM 0 TO Array.length:
      IF Array[i] IS NOT null:
         PRINT ELEMENT USING print() METHOD
      ELSE:
        PRINT "( , )"
    ADD FORMATTING BASED ON POSITION (FIRST, LAST, OR MIDDLE ELEMENT)
PRINT " }"
```

CLASS SPQ<K extends Comparable<K>, V>:

```
PRIVATE expandingArray OF TYPE ExpandableArray<K, V>
PRIVATE CONSTANT DefaultCapacity = 10
PRIVATE last_added OF TYPE int INITIALIZED TO 0
PRIVATE size OF TYPE int INITIALIZED TO 0
PRIVATE currentHeapType OF TYPE HeapType INITIALIZED TO Max

ENUM HeapType:
    Max, Min

# Constructors
CONSTRUCTOR SPQ():
    CALL SPQ(HeapType.Max, DefaultCapacity)

CONSTRUCTOR SPQ(heapType):
    CALL SPQ(heapType, DefaultCapacity)

CONSTRUCTOR SPQ(startingSize):
    CALL SPQ(HeapType.Max, startingSize)

CONSTRUCTOR SPQ(heapType, startingSize):
    INITIALIZE expandingArray WITH ExpandableArray OF SIZE startingSize
    SET currentHeapType TO heapType

# Utility Methods
METHOD size():
    RETURN expandingArray.size()

METHOD length():
    RETURN expandingArray.size()

METHOD capacity():
    RETURN expandingArray.Capacity()

METHOD isEmpty():
    RETURN expandingArray.isEmpty()
```

```
METHOD clear():
    CALL expandingArray.clear()
    SET last_added TO 0
    SET size TO 0

METHOD ensureCapacity():
    CALL expandingArray.ensureCapacity()

# Navigation Methods
METHOD getParent(index):
    RETURN (index - 1) / 2 IF VALID INDEX, ELSE RETURN -1

METHOD getLeftChild(index):
    RETURN 2 * index + 1 IF VALID INDEX, ELSE RETURN -1

METHOD getRightChild(index):
    RETURN 2 * index + 2 IF VALID INDEX, ELSE RETURN -1

METHOD hasParent(index):
    RETURN getParent(index) >= 0

METHOD hasLeftChild(index):
    RETURN getLeftChild(index) < size

METHOD hasRightChild(index):
    RETURN getRightChild(index) < size

# Search Methods
METHOD linearSearch(target):
    FOR i FROM 0 TO expandingArray.Capacity():
        IF expandingArray.get(i) IS NOT null AND MATCHES target:
            RETURN i
    RETURN -1

# Heap Operations
METHOD toggle():
    SWITCH currentHeapType BETWEEN Max AND Min
    CALL heapSortAndInsert()

METHOD heapSortAndInsert():
    CREATE data ARRAY
    REMOVE ALL ELEMENTS FROM HEAP INTO data
    INSERT ALL ELEMENTS BACK INTO HEAP
```

```
METHOD removeTop():
    IF HEAP IS EMPTY:
        RETURN null
    REMOVE ROOT ELEMENT
    SWAP WITH LAST ELEMENT
    REHEAPIFY USING downHeap(0)
    RETURN REMOVED ELEMENT

METHOD insert(key, value):
    CALL ensureCapacity()
    ADD NEW ELEMENT TO THE HEAP
    CALL upHeap(last_added)
    RETURN NEW ELEMENT

METHOD remove(target):
    FIND INDEX OF target USING linearSearch()
    REMOVE ELEMENT FROM THAT INDEX
    SWAP WITH LAST ELEMENT
    REHEAPIFY USING upHeap() AND downHeap()
    RETURN REMOVED ELEMENT

# Replace Methods
METHOD replaceKey(target, newKey):
    FIND INDEX OF target USING linearSearch()
    UPDATE KEY AT THAT INDEX
    REHEAPIFY USING upHeap() AND downHeap()
    RETURN OLD KEY

METHOD replaceValue(target, newValue):
    FIND INDEX OF target USING linearSearch()
    UPDATE VALUE AT THAT INDEX
    RETURN OLD VALUE

# Heapify Methods
METHOD upHeap(index):
    WHILE index > 0:
        FIND PARENT INDEX
        IF HEAP PROPERTY IS VALID:
            BREAK
        SWAP ELEMENT WITH PARENT
        UPDATE index TO PARENT INDEX

METHOD downHeap(index):
    WHILE index IS WITHIN BOUNDS:
        FIND LEFT AND RIGHT CHILD INDICES
        DETERMINE child BASED ON HEAP TYPE AND PRIORITY
        IF HEAP PROPERTY IS VALID:
            BREAK
        SWAP ELEMENT WITH child
        UPDATE index TO child

    # Other Methods
    METHOD top():
        RETURN ROOT ELEMENT OF THE HEAP

    METHOD printPriorityQueue():
        CALL expandingArray.printArray()
```

COMPLEXITIES OF THE METHOD SPQ CLASS

1. Initialization
   - Time Complexity: O(1)
   - Space Complexity: O(1)

2. Top()
   - Time Complexity: O(1)
   - Space Complexity: O(1)

3. remove(K e) / remove(Input<K, V> e)
   - Time Complexity: O(log n)
   - Space Complexity: O(1)

4. replaceKey(Input<K, V>, K)
   - Expected Time Complexity: O(log n)
   - Worst-Case Time Complexity: O(n)
   - Space Complexity: O(1)

5. replaceValue(Input<K, V>, V)
   - Expected Time Complexity: O(1)
   - Worst-Case Time Complexity: O(n)
   - Space Complexity: O(1)

6. state()
   - Time Complexity: O(1)
   - Space Complexity: O(1)

7. toggle()

- Time Complexity: O(n log n)
- Space Complexity: O(n)

8. heapSortAndInsert()
   - Time Complexity: O(n log n)
   - Space Complexity: O(n)

9. upHeap() / downHeap()
   - Time Complexity: O(log n)
   - Space Complexity: O(1)

10. insert(K key, V value)
    - Time Complexity: O(log n)
    - Space Complexity: O(1)

11. linearSearch()
    - Time Complexity: O(n)
    - Space Complexity: O(1)