## QUESTION 4

(a)

```
// Abstract state: queue = [int]   // use of Haskell notations throughout the trait
                                   // = a set of elements (possibly with repetitions)
// Init: queue = ∅

trait DoubleEndedQueue {

  /** tests whether the queue is empty */
  // Post: return (length queue)
  def isEmpty: Boolean

  /** adds the integer x to the start of the queue */
  // Post: queue = x : queue_0
  def addLeft (x: Int): Unit

  /** removes the element at the start of the queue and returns it */
  // Pre: length queue > 0
  // Post: queue = tail queue_0 && return (head queue_0)
  def getLeft: Int

  /** adds the integer x to the end of the queue */
  // Post: queue = queue_0 ++ [x]
  def addRight (x: Int): Unit

  /** removes the element at the end of the queue and returns it */
  // Pre: length queue > 0
  // Post: queue = init queue_0 && return (last queue_0)
  def getRight: Int

}
```

(b) If we use a circular array that can contain at most MAX elements in it, we need to add the precondition: length queue < MAX to addLeft and addRight.

```
class ArrayDoubleEndedQueue (val MAX: Int) extends DoubleEndedQueue {
  private var a = new Array [Int](MAX) // the circular array
  private var left = 0 // the head of the queue
  private var right = 0 // the end of the queue
```

```
    private var size = 0  // the number of elements in the array
  // Abstraction function: queue = { a(i) | if (left < right) i ∈ [left .. right)
  //                                        else if (left > right) i ∈ [0 .. right) ∪ [left .. MAX)
  //                                        else if ((left == right) && (size == MAX)) i ∈ [0 .. MAX)
  //                                        else i ∈ ∅ }
  // RTi: 0 ≤ left, right < MAX && size = (right - left) % MAX if right != left && 0 ≤ size ≤ MAX
```

(c)

```
  def isEmpty : Boolean = (size == 0)
  def addLeft (x: Int) : Unit = {
      if (left == 0) left = MAX-1
      else left -= 1
      a(left) = x
      size += 1
  }
  def getLeft : Int = {
      var res = a(left)
      if (left == MAX-1) left = 0
      else left += 1
      size -= 1
      return res
  }
  def addRight (x: Int) : Unit = {
      a(right) = x
      if (right == MAX-1) right = 0
      else right += 1
      size += 1
  }
  def getRight : Int = {
      if (right == 0) right = MAX-1
      else right -= 1
      var res = a(right)
      size -= 1
      return res
  }
```

(d) (i)

```
/** prints a representation of the current abstract datatype to the screen */
// Post: print every element of the array a from a(left) until a(left + size) (wrapping around)
def display : Unit = {
    for (i <- left until left + size) print(a(i % MAX) + " ")
    println()
}
```

(ii) The code fragment from the task will print:

```
20
10
10
```

Queue status: 30 40 15