```kotlin
package net.zeevox.nearow.data

import java.util.concurrent.atomic.AtomicInteger

/**
* Initialise a new circular buffer (ring buffer / circular array) for storing [Double] values.
* @param _size integer specifying the capacity of the array
*/
class CircularDoubleBuffer(private val _size: Int) : Collection<Double> {

    /**
     * Secondary constructor that supports initialising the circular buffer with data given by a
     * user-provided function mapping indices to values.
     */
    constructor(_size: Int, function: (Int) -> Double) : this(_size) {
        (0 until size).forEach { index -> addLast(function(index)) }
    }

    /** Returns the size of the collection. */
    override val size: Int
        get() = _size

    private val buffer = DoubleArray(size)
    private var pointer = 0

    /**
     * Add a single [Double] value to the end of the list, displacing the oldest recorded value.
     * @param value is a [Double] value to save to the tail of the ring buffer
     */
    fun addLast(value: Double) {
        // record given reading
        buffer[pointer] = value

        // update circular buffer pointer
        pointer = (pointer + 1).mod(size)
    }

    /** Get the most recently added value */
    val head: Double
        get() = this[-1]

    /** Get the oldest value in the buffer */
    val tail: Double
        get() = this[0]

    /**
     * Returns a string representation of the circular array, with the newest value printed last.
     */
    override fun toString(): String =
        "[${
            (buffer.sliceArray(pointer until size)
                    + buffer.sliceArray(0 until pointer))
                .joinToString(", ")
        }]"

    operator fun get(index: Int): Double = buffer[(pointer + index).mod(size)]

    /** Checks if the specified element is contained in this collection. */
    override fun contains(element: Double): Boolean = buffer.any { it == element }

    /** Checks if all elements in the specified collection are contained in this collection. */
    override fun containsAll(elements: Collection<Double>): Boolean =
        elements.all { element -> buffer.any { it == element } }

    override fun iterator(): Iterator<Double> =
        object : Iterator<Double> {
            private val index: AtomicInteger = AtomicInteger(0)

            /** Returns `true` if the iteration has more elements. */
            override fun hasNext(): Boolean = index.get() < size

            /** Returns the next element in the iteration. */
            override fun next(): Double = get(index.getAndIncrement())
        }

    /** Returns `true` if the collection is empty (contains no elements), `false` otherwise. */
    override fun isEmpty(): Boolean = size > 0
}
```