```java
package poo_unidade_11_ex04_buffercircular;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 * @author Edwar Saliba Júnior
 */
public class BufferCircular implements Buffer {
    // Bloquei para controlar sincronização com este buffer.
    private Lock accessLock = new ReentrantLock();

    // Condições para controlar gravação e leitura.
    private Condition canWrite = accessLock.newCondition();
    private Condition canRead = accessLock.newCondition();

    private int [] buffer = {-1, -1, -1};

    private int occupiedBuffers = 0;  // Conta número de buffers utilizados.
    private int writeIndex = 0;  // Índice para escrever o próximo valor.
    private int readIndex = 0;   // Índice para ler o próxim valor.


    @Override
    public void set(int value) {
        accessLock.lock(); // Bloqueia o objeto.
        try{
            // Enquanto não houver posições vazias põe a thread no estado de
            // espera.
            while(occupiedBuffers == buffer.length){
                System.out.println("Buffer cheio. Produtor esperando.");
                canWrite.await();
            }

            buffer[writeIndex] = value;

            // Atualiza índice de gravação circular.
            writeIndex = (writeIndex + 1) % buffer.length;

            occupiedBuffers++;  // Outra posição do buffer está cheia.

            displayState("Produtor escreveu: " + value);

            // Sinalização para a thread que está esperando par ler o buffer.
            canRead.signal();
        }
        catch(InterruptedException e){
            e.printStackTrace();
        }
        finally{
            accessLock.unlock();
        }
    }

    @Override
    public int get() {
```

```
    public int get() {
        int readValue = 0;

        // Bloqueia este objeto.
        accessLock.lock();

        try{
            while(occupiedBuffers == 0){
                System.out.println("Buffer vazio. Consumidor esperando.");
                canRead.await();
            }

            // Lê valor do buffer na posição apontada pelo índice.
            readValue = buffer[readIndex];

            // Atualiza o índice de leitura Circular
            readIndex = (++readIndex) % buffer.length;

            occupiedBuffers--;  // Mais uma posição do buffer está vazia.

            displayState("Consumidor leu:    " + readValue);

            canWrite.signal();
        }
        catch(InterruptedException e){
            e.printStackTrace();
        }
        finally{
            accessLock.unlock();
        }

        return readValue;
    }

    public void displayState(String operation){
        System.out.printf("%s%s%d)\n%s", operation,
           "(posições ocupadas no buffer: ", occupiedBuffers, "buffers:  ");

        for(int value: buffer)
            System.out.printf(" %2d", value);

        System.out.print("\n         ");
        for(int i = 0; i < buffer.length; i++){
            System.out.print("---- ");
        }

        System.out.print("\n         ");

        for(int i = 0; i < buffer.length; i++){
            if(i == writeIndex && i == readIndex)
                System.out.print(" WR");
            else
                if(i == writeIndex)
                    System.out.print(" W   ");
                else
                    if(i == readIndex)
                        System.out.print("  R  ");
```

```
                    else
                        System.out.print("     ");
            }

            System.out.println("\n");
        }
    }
```