```java
package poo_unidade_11_ex03_buffersincronizado;

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

/**
 * @author Edwar Saliba Júnior
 */
public class BufferSincronizado implements Buffer{
    // Bloqueio 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; // Compartilhado pelas threads Produtor/Consumidor.
    private boolean occupied = false; // Buffer ocupado ou não.


    @Override
    public void set(int value) {
        accessLock.lock(); // Tenta bloquear o objeto.
        try{
            // Produtor aguarda até que o buffer esteja vazio e liberado.
            while(occupied){
                System.out.println("\nProdutor tentou escrever.");
                System.out.println("\nBuffer cheio. Produtor aguardando.");
                // Espera até que seja emitido o sinal de buffer vazio.
                canWrite.await();
            }

            buffer = value;

            // Indica que o buffer está cheio, ou seja, não poderá
            // armazenar outro valor até que o consumidor copie este
            // que está no buffer.
            occupied = true;

            displayState("Produtor escreveu   : " + buffer);

            // Sinaliza para a thread que está esperando para ler o buffer.
            canRead.signal();
        }
        catch(InterruptedException e){
            e.printStackTrace();
        }
        finally{
            // Libera o bloqueio sobre os dados compartilhados.
            // Desbloqueia este objeto.
            accessLock.unlock();
        }
    }

    @Override
    public int get() {
```

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

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

        try{
            // Consumidor aguarda até que o buffer esteja cheio e liberado.
            while(!occupied){
                System.out.println("\nConsumidor está tentando ler o buffer.");
                System.out.println("\nBuffer vazio. Consumidor aguardando.");
                // Espera até que seja emitido o sinal de buffer cheio.
                canRead.await();
            }

            readValue = buffer;

            // Indica que o Produtor pode escrver outro valor no buffer,
            // pois o Consumidor acabou de copiar o último valor armaze-
            // nado.
            occupied = false;

            displayState("Consumidor leu      : " + readValue);

            // Sinaliza para thread que está esperando o buffer ficar vazio.
            canWrite.signal();
        }
        catch(InterruptedException e){
            e.printStackTrace();
        }
        finally{
            // Libera o bloqueio sobre os dados compartilhados.
            // Desbloqueia este objeto.
            accessLock.unlock();
        }

        return readValue;
    }

    public void displayState(String operation){
        System.out.printf("\n%-40s%d\t\t%b\n\n", operation, buffer, occupied);
    }
}
```