Descrição do comportamento observado

O código base não constrola o comportamento das threads se interrelacionam e aplica lock no laço. Os seguintes comportamentos foram observados:

1) Quando o Consumer é iniciado primeiro, o laço for é consumido totalmente com valor igual a zero para apenas então liberar o lock para o Producer. O Producer, então, adiciona todos os demais valores. 2) Quando o Producer é iniciado primeiro, são adicionados todos os valores primeiro para que sejam consumidos apenas posteriormente. 3) Situações de concorrência de acesso à variável também ocorreram, não sendo possível prever o comportamento de soma e subtração.

Problemas identificados:

O comportamento adotado pela solução é a estrutura pilha, o tempo tomado para processar itens mais velhos é maior que itens mais novos pela situação única de ordenação de acesso. Nesse caso, o comportamento de fila encadeada resolveria a situação de consumo de itens mais novos antes de itens mais velhos, mas aumentaria o tempo de processamento. Outro problema identificado é que a complexidade da solução é aumentada, uma vez que será necessário gerir memória e será empregada uma estrutura de dados que não existe atualmente na implementação.

Assim, outra solução que utiliza o conceito de fila encadeada é a utilização e Object#notify e Object#wait nos elementos, Assim, embora o tempo e processamento seja potenciamente maior, nenhum dado novo é consumido ou gerado antes que o anterior tenha sido processado.

Referência:

```
public class Main {
    public volatile static int produtos = 0;

    public static void main(String[] args) {
        Produtor p1 = new Produtor(1);
        Consumidor c1 = new Consumidor(2);
        p1.start();
        c1.start();
    }
}

class Produtor extends Thread {
    int nProdutor = 0;
    private static volatile Object lock = new Object();
    Produtor(int num){
        this.nProdutor = num;
    }
    public void run() {
        for(int i = 0; i <= 20; i++) {
            synchronized( lock )
            {
                if (Main.produtos < 100)
                    Main.produtos = Main.produtos + 1;
```

```
            }
            System.out.println("Produtor: " + nProdutor + ": " + Main.produtos);
        }
    }
}

class Consumidor extends Thread {
    int nConsumidor = 0;
    private static volatile Object lock = new Object();
    Consumidor(int num){
        this.nConsumidor = num;
    }
    public void run() {
        for(int i = 0; i <= 20; i++) {
            synchronized( lock )
            {
                if ( Main.produtos > 0)
                    Main.produtos = Main.produtos - 1;
            }
            System.out.println("Consumidor: " + nConsumidor + ": " + Main.produtos);
        }
    }
}
```