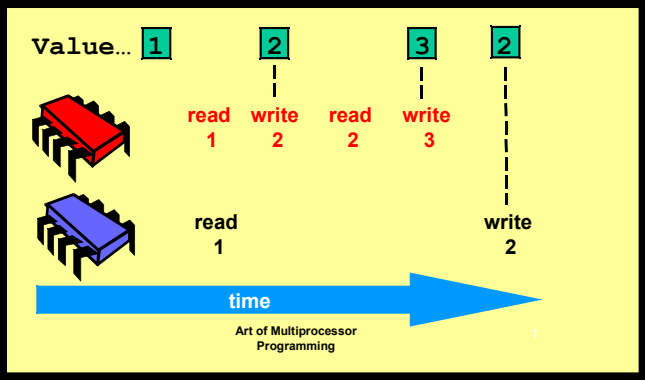
Relembrando: por que exclusão mútua?




```
class GerContas extends Thread {
  Conta[] contas=null;
  public GerContas(Conta[] accs) {
    contas = accs;
  }
  public void run(){
    contas[0].depositar(100);
    contas[3].sacar(50);
    contas[2].depositar(100);
    contas[1].sacar(50);
  }
}
```

Região crítica

...e o método `main()` cria várias contas e passa elas para dois `GerContas`.

Monitores

Objetos que garantem a exclusão mútua

- na execução dos procedimentos associados a eles.
- apenas **um procedimento associado ao monitor** pode ser executado de cada vez
- Atomicidade

Em Java todo objeto possui um monitor associado.

```
public class Conta {
  double saldo;
  String  nome;
  public Conta(String nm,double amnt) {
        saldo=amnt;
        nome=nm;
  }
  synchronized void depositar(double money){
      saldo+=money;
  }
  synchronized void sacar(double money){
      saldo-=money;
  }
...
```

```
public class Conta {
  double saldo;
  String  nome;
  public Conta(String nm,double amnt) {
        saldo=amnt;
        nome=nm;
  }
  void depositar(double money){
      saldo+=money;
  }
  void sacar(double money){
      saldo-=money;
  }
...
```

Métodos sincronizados

Estratégia simples de prevenção de interferência entre `threads`

Previnem também:

- Inicialização parcial
- Reordenamento de instruções

Podem ser caros, dependendo da aplicação

Métodos sincronizados 2

Para executá-los, é necessário adquirir o **monitor do objeto**

- Se for `static`, **monitor da classe**

Mesmo monitor, **`this`**, para todos os métodos sincronizados de uma classe

- Reiterando: monitor é o **objeto**, não a classe

Exemplo:

`Parenteses.java`,
`MinhaThread.java`,
`Demo.java`

Exercício

Defina uma classe ArvoreBusca que implementa uma árvore de busca onde é possível realizar inserções de elementos. Essa estrutura de dados deve funcionar com várias threads. Faça o que é pedido:

- Implemente um método `main()` que cria 50 threads onde cada uma insere 2000 números aleatórios nessa árvore.
- Seu programa deve informar a quantidade de nós total da árvore após todas as inserções
- Meça o tempo de execução do seu programa, comparando-o com o de uma execução puramente sequencial.
- O que significa "**funcionar com várias threads**"?

Aninhamento de blocos `synchronized`

```
public class Conta {
  ...
  // Deve ser feita atomicamente
  void transferir (Conta dest,double money){
      synchronized(this) {
          synchronized(dest) {
              this.saldo -= money;
              dest.saldo += money;
          }
      }

...
```

Blocos sincronizados

Reduzem o custo de sincronização associado ao método

- Afetam apenas parte do método

Monitor fica explícito

- Aninhamento de monitores também

Aninhamento de blocos `synchronized`

```
public class Conta {
```

Pode entrar em deadlock!

```
              dest.saldo += money;
          }
      }

...
```

Exercício

Modifique o programa do exercício anterior para tornar mais fina a granularidade do travamento. Em outras palavras, faça com que o travamento seja feito por nó, ao invés de afetar a árvore inteira.

- Compare o desempenho desta versão com o da sequencial e o da que usa apenas uma trava