Travas explícitas

- Mais flexíveis
  - Não apenas travamento com escopo
  - Travamento não-bloqueante
  - Multiplas `Conditions`
- Mais complexas
- Interface `Lock`, classe `ReentrantLock`
  - Outras implementações estão disponíveis

```
// padrão geral de uso
Lock trava = ... ;// cria a trava
trava.lock();
try {
   ... //acesso aos recursos compartilhados
} finally {
   trava.unlock();
}
```

```
public class Conta {
  private Lock l = new ReentrantLock();
  ...
  // Deve ser feita atomicamente
  void transferir (Conta dest,double money){
    boolean t1 = l.tryLock();
    boolean t2 = dest.l.tryLock();
    try {
      while (!(t1 && t2)) {
        if (t1) l.unlock();
        if (t2) dest.l.unlock();
        t1 = l.tryLock(); t2 = dest.l.tryLock();
      }
      this.saldo -= money;
      dest.saldo += money;
    } finally {
      if (t1) l.unlock();
      if (t2) dest.l.unlock();
    }
  }
}
```

Exercício

- Reimplemente o programa do exercício anterior (árvore de busca) usando travas explícitas. Compare o desempenho dessa abordagem com o da anterior.

Exercício

- Modifique a sua implementação de árvore para que também inclua uma operação de remoção de nós. Faça com que as threads que você cria, que antes faziam apenas operações de inserção, agora também façam operações de remoção. Meça o desempenho.
- Modifique sua abordagem de travamento para garantir que, se duas threads estão tentando percorrer, inserir ou remover nós em sub-árvores diferentes, elas **podem fazer isso em paralelo** (caso o hardware permita). Em outras palavras, não pode mais usar travamento global!!!

Exercício (1/2)

Implemente um vetor (tamanho fixo) de inteiros seguro para threads com operações para ler e escrever em determinado índice e para trocar os valores armazenados em dois índices distintos.

Exercício (2/2)

Sua solução deve satisfazer as seguintes propriedades:

Safety: duas threads nunca obtém acesso a uma posição do vetor ao mesmo tempo.

Safety: a operação de troca (*swap*) dos valores em posições distintas deve ser atômica

Liveness: para um vetor de tamanho *N*, até *K* <= *N* threads podem manipular elementos do vetor simultaneamente (considere que *N* é múltiplo de *K*).

Liveness: o programa não entra em deadlock