

Exclusão Mútua - locks explícitos -

Sistemas Distribuídos

Exclusão mútua em JAVA

● com keyword synchronized

Método:

```
public synchronized void metodo() {  
    x++;  
    i++;  
}
```

Bloco:

```
public void metodo() {  
    x++;  
    synchronized (objecto){  
        i++;  
    }  
}
```

Exclusão mútua em JAVA

● com keyword synchronized

Método:

```
public synchronized void metodo() {  
    x++;  
    i++;  
}
```

Bloco:

```
public void metodo() {  
    x++;  
    synchronized (objecto){  
        i++;  
    }  
}
```

● com lock explícito — classe ReentrantLock

```
private ReentrantLock lockA = new  
ReentrantLock();  
private ReentrantLock lockB = new  
ReentrantLock();
```

```
public void metodo() {  
    lockA.lock();  
    x++;  
    lockB.lock();  
    if(x == 0){  
        i++;  
        lockB.unlock();  
    }  
    else{  
        lockB.unlock();  
        i--;  
    }  
    lockA.unlock();  
}
```

Exercícios

- 1) Modifique o exercício do banco de modo a:
 - ser possível criar e fechar contas; criar uma conta devolve um identificador de conta para ser usado noutras operações, e fechar uma conta devolve o saldo desta.
 - ser possível obter a soma do saldo de um conjunto de contas.
 - lançar exceção `ContaInvalida` se identificador de conta não existir e `SaldoInsuficiente` se não houver saldo suficiente para a operação.
 - permitir concorrência (usando `ReentrantLock`), mas garantindo a atomicidade das operações (e.g. ao somar os saldos de um conjunto de contas, não permita que sejam usados montantes a meio de uma transferência)

Banco

```
private HashMap<Integer, Conta> contas;  
private ReentrantLock lockBanco;
```

```
Banco()  
int criarConta(double saldoInicial) ← retorna id da nova conta criada  
double fecharConta(int id) ← retorna saldo da conta fechada  
    throws ContaInvalida  
void transferir(int c_origem, int c_destino, double valor)  
    throws ContaInvalida, SaldoInsuficiente;  
double consultarTotal(int contas[]);  
double consultar(int id)  
    throws ContaInvalida;  
void levantar(int id, double valor)  
    throws ContaInvalida, SaldoInsuficiente;  
void depositar(int id, double valor)  
    throws ContaInvalida;
```

Conta

```
private double saldo;  
private ReentrantLock lockConta;
```

```
Conta(double saldo)  
double consultar()  
void depositar(double valor)  
void levantar(double valor)
```

Main

```
//criar objecto Banco  
//criar duas contas no Banco c/ saldo 10€  
//criar e iniciar Cliente1 e Cliente2
```

Cliente1 ← implements Runnable → Cliente2

```
private Banco banco;  
run()  
    banco.criarConta(0)  
    banco.transferir(0, 2, 5)  
    banco.consultarTotal([0,1,2])
```

```
private Banco banco;  
run()  
    banco.transferir(0, 1, 10)  
    banco.fecharConta(1)  
    banco.consultar(0)
```

Exercício do Banco (Guião 3)

- Método Banco.criarConta
- Abordagem: Usar lockBanco para proteger alterações ao HashMap contas

```
// Usado para gerar IDs de Conta
private int lastId = 0;

public int criarConta(double saldo){
    this.lockBanco.lock();
    int id = lastId++;
    Conta c = new Conta(id, saldo);
    this.contas.put(id, c);
    this.lockBanco.unlock();

    return id;
}
```

Exercício do Banco (Guião 3)

- Método Banco.criarConta
- Abordagem: Usar lockBanco para proteger alterações ao HashMap contas

novos id tem de ser obtido dentro da secção crítica, senão duas threads podem criar 2x a mesma conta

```
public int criarConta(double saldo){  
    this.lockBanco.lock();  
    → int id = lastId++;  
    Conta c = new Conta(id, saldo);  
    this.contas.put(id, c);  
    this.lockBanco.unlock();  
  
    return id;  
}
```

Exercício do Banco (Guião 3)

- Método Banco.fecharConta
- Abordagem 1: Verificar se conta existe e depois obter lockBanco para proteger alterações ao HashMap contas

```
public double fecharConta(int id) throws ContaInvalida{
    if(contas.containsKey(id)){
        this.lockBanco.lock();
        Conta c = this.contas.get(id);
        double saldo = c.consultar();
        this.contas.remove(id);
        this.lockBanco.unlock();
        return saldo;
    } else
        throw new ContaInvalida(id);
}
```


Exercício do Banco (Guião 3)

- Método Banco.fecharConta
- Abordagem 1: Verificar se conta existe e depois obter lockBanco para proteger alterações ao HashMap contas

Cliente1: banco.fechar(0)

Cliente2: banco.fechar(0)

Cliente1

```
if(contas.containsKey(id)){ True
```

```
this.lockBanco.lock();  
Conta c = this.contas.get(id);  
double saldo = c.consultar();  
this.contas.remove(id);  
this.lockBanco.unlock();
```

Cliente2

```
if(contas.containsKey(id)){ True
```

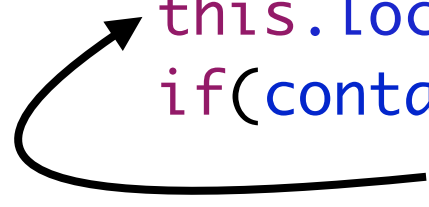
```
this.lockBanco.lock();  
Conta c = this.contas.get(id);  
double saldo = c.consultar();  
this.contas.remove(id);  
this.lockBanco.unlock();
```

**NullPointerException:
Conta 0 já não existe!**

Exercício do Banco (Guião 3)

- Método Banco.fecharConta
- Abordagem 2: Obter lockBanco para verificar se a conta existe e para proteger alterações ao HashMap contas

```
public double fecharConta(int id) throws ContaInvalida{
    this.lockBanco.lock();
    if(contas.containsKey(id)){
        this.lockBanco.lock();
        Conta c = this.contas.get(id);
        double saldo = c.consultar();
        this.contas.remove(id);
        this.lockBanco.unlock();
        return saldo;
    } else {
        this.lockBanco.unlock();
        throw new ContaInvalida(id);
    }
}
```



Exercício do Banco (Guião 3)

- Método Banco.fecharConta
- Abordagem 2: Obter lockBanco para verificar se a conta existe e para proteger alterações ao HashMap contas

Cliente1: banco.fechar(0)

Cliente2: banco.fechar(0)

Cliente1

```
this.lockBanco.lock();  
if(contas.containsKey(id)){ True  
Conta c = this.contas.get(id);  
double saldo = c.consultar();  
this.contas.remove(id);  
this.lockBanco.unlock();
```

Cliente2

```
this.lockBanco.lock();  
if(contas.containsKey(id)){ False  
this.lockBanco.unlock();  
throw new ContaInvalida(id);
```

Exercício do Banco (Guião 3)

- Método Banco.fecharConta
- Abordagem 2: Obter lockBanco para verificar se a conta existe e para proteger alterações ao HashMap contas

Cliente1: banco.fechar(0)

Cliente2: banco.depositar(0,10)

Cliente1

```
this.lockBanco.lock();
if(contas.containsKey(id)){ True
Conta c = this.contas.get(id);
double saldo = c.consultar();
this.contas.remove(id);
this.lockBanco.unlock();
```

Cliente2

```
this.lockBanco.lock();
if(contas.containsKey(id)){ True
this.contas.get(id).lock();
this.lockBanco.unlock();
```

**NullPointerException:
Conta 0 já não existe!**



```
this.contas.get(id).depositar(10);
this.contas.get(id).unlock();
```

Exercício do Banco (Guião 3)

- Método Banco.fecharConta
- Abordagem 3: Obter lockBanco para verificar se a conta existe e para proteger alterações ao HashMap contas. Obter lockConta para impedir que se remova uma conta quando outra thread está a meio de uma operação sobre essa conta.

```
public double fecharConta(int id) throws ContaInvalida{
    this.lockBanco.lock();
    if(contas.containsKey(id)){
        Conta c = this.contas.get(id);
        c.lock();
        double saldo = c.consultar();
        this.contas.remove(id);
        c.unlock();
        this.lockBanco.unlock();
        return saldo;
    } else {
        this.lockBanco.unlock();
        throw new ContaInvalida(id);
    }
}
```

Exercício do Banco (Guião 3)

- Método Banco.fecharConta
- Abordagem 3: Obter lockBanco para verificar se a conta existe e para proteger alterações ao HashMap contas. Obter lockConta para impedir que se remova uma conta quando outra thread está a meio de uma operação sobre essa conta.

Cliente1: banco.fechar(0)

Cliente2: banco.depositar(0,10)

Cliente1

```
this.lockBanco.lock();  
if(contas.containsKey(id)){  
Conta c = this.contas.get(id); True
```

```
c.lock();  
double saldo = c.consultar();  
this.contas.remove(id);  
c.unlock();  
this.lockBanco.unlock();
```

Cliente2

```
this.lockBanco.lock();  
if(contas.containsKey(id)){ True  
this.contas.get(id).lock();  
this.lockBanco.unlock();
```

```
this.contas.get(id).depositar(10);  
this.contas.get(id).unlock();
```

Exercício do Banco (Guião 3)

- Método Banco.consultarTotal
- Abordagem 1: Para cada conta, obter lockBanco para verificar se a conta existe e obter lockConta para impedir que outra thread remova a conta durante a consulta do saldo.

```
public double consultarTotal(int[] contasInput) {  
    double saldoTotal = 0;  
    for(int i = 0; i < contasInput.length; i++){  
        int id = contasInput[i];  
        this.lockBanco.lock();  
        if(contas.containsKey(id)){  
            contas.get(id).lock();  
            this.lockBanco.unlock();  
            saldoTotal += contas.get(id).consultar();  
            contas.get(id).unlock();  
        }  
        else this.lockBanco.unlock();  
    }  
    return saldoTotal;  
}
```

Exercício do Banco (Guião 3)

- Método Banco.consultarTotal
- Abordagem 1: Para cada conta, obter lockBanco para verificar se a conta existe e obter lockConta para impedir que outra thread remova a conta durante a consulta do saldo.

Cenário: conta0 = conta1 = conta2 = 10

Cliente1: banco.consultarTotal([0,1,2])

Cliente2: banco.transferir(0,2,10)

Cliente1

```
this.contas.get(0).lock();  
saldoTotal += this.contas.get(0).consultar(); saldoTotal = 10  
this.contas.get(0).unlock();
```

```
this.contas.get(1).lock();  
saldoTotal += this.contas.get(1).consultar(); saldoTotal = 20  
this.contas.get(1).unlock();  
this.contas.get(2).lock();  
saldoTotal += this.contas.get(2).consultar(); saldoTotal = 40  
this.contas.get(2).unlock();
```

Cliente2

```
banco.transferir(0,2,10)
```

*obtem locks das
contas 0 e 2 e
faz a transferência*

Banco só tem 30€!

Exercício do Banco (Guião 3)

- Método Banco.consultarTotal
- Abordagem 2: Obter lockBanco para obter todos os lockConta das contas válidas desejadas. Consultar saldo total só após ter todos os locks das contas.

```
public double consultarTotal(int[] contasInput) {  
    double saldoTotal = 0;  
    ArrayList<Integer> contasLocked = new ArrayList(contasInput.length);  
    this.lockBanco.lock();  
    for(int i = 0; i < contasInput.length; i++){  
        int id = contasInput[i];  
        if(contas.containsKey(id)){  
            this.contas.get(id).lock();  
            contasLocked.add(id);  
        }  
    }  
    this.lockBanco.unlock();  
    for(int id : contasLocked){  
        saldoTotal += contas.get(id).consultar();  
        contas.get(id).unlock();  
    }  
    return saldoTotal;  
}
```

obter os locks das contas

consultar o saldo total

Exercício do Banco (Guião 3)

- Método Banco.consultarTotal
- Abordagem 2: Obter lockBanco para obter todos os lockConta das contas válidas desejadas. Consultar saldo total só após ter todos os locks das contas.

Cenário: conta0 = conta1 = conta2 = 10

Cliente1: banco.consultarTotal([0,1,2])

Cliente2: banco.transferir(0,2,10)

Cliente1

```
this.contas.get(0).lock();  
this.contas.get(1).lock();  
this.contas.get(2).lock();  
(...)  
saldoTotal += this.contas.get(0).consultar(); saldoTotal = 10  
this.contas.get(0).unlock();  
saldoTotal += this.contas.get(1).consultar(); saldoTotal = 20  
this.contas.get(1).unlock();  
saldoTotal += this.contas.get(2).consultar(); saldoTotal = 30  
this.contas.get(2).unlock();
```

Cliente2

```
banco.transferir(0,2,10)
```

*obtem locks das
contas 0 e 2 e
faz a transferência*