UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO – UNIVASF
COLEGIADO DE ENGENHARIA DA COMPUTAÇÃO - CCOMP
CCMP0030 – SISTEMAS OPERACIONAIS

Professora Andreza Leite
Instrutor Renan Costa Alencar

EXERCÍCIO DE FIXAÇÃO – MULTITHREADING EM JAVA

Leia o problema da classe *BankAccount* no livro Big Java 4ª Edição, paginas 809 a 824. Adicione uma condição para que o método *deposit* da classe *BankAccount* restrinja depósitos quando o saldo total da conta for de R$70.000,00 (limite segurado pelo governo brasileiro para poupanças). O método deve bloquear até que a quantia necessária seja sacada por outro thread. Teste o seu programa com um numero grande de threads para deposito.

Preencha os espaços em branco.

*Arquivo **BankAccountThreadRunner**.java*

```
/**
Este programa executa threads que depositam e sacam
dinheiro da mesma conta bancária.
*/
public class BankAccountThreadRunner
{
public static void main(String[] args)
{
BankAccount account = new BankAccount();
final double AMOUNT = 10000;
final int REPETITIONS = 10;
final int DEPOSIT_THREADS = 10;
final int WITHDRAW_THREADS = 2;
for (int i = 0; i < DEPOSIT_THREADS; i++)
{
DepositRunnable d = new DepositRunnable(account, AMOUNT, REPETITIONS);
Thread t = _____________________;
_____________________________;
}
for (int i = 0; i < WITHDRAW_THREADS; i++)
{
WithdrawRunnable d = new WithdrawRunnable(account, AMOUNT, REPETITIONS *
DEPOSIT_THREADS /
WITHDRAW_THREADS);
Thread t _____________________;
_____________________________;
}
}
}
```

*Arquivo **BankAccount**.java*

```
public class BankAccount
{
public static final double MAX_BALANCE = 70000;
private double balance;
private Lock balanceChangeLock;
private Condition sufficientFundsCondition;
```

```
private Condition lessThanMaxBalanceCondition;
/**
Constroi uma conta bancária com saldo zero.
*/
public BankAccount()
{
balance = 0;
balanceChangeLock = _____________________________________;
sufficientFundsCondition = balanceChangeLock.newCondition();
lessThanMaxBalanceCondition = _____________________________;
}
/**
Deposita dinheiro na conta bancária.
@param amount a quantia a ser depositada
*/
public void deposit(double amount)throws InterruptedException
{
_____________________________________________________;
try
{
while (balance + amount > MAX_BALANCE)
lessThanMaxBalanceCondition.await();
System.out.print("Depositando " + amount);
double newBalance = balance + amount;
System.out.println(", o novo saldo é " + newBalance);
balance = newBalance;
_____________________________________________________;
}
finally
{
balanceChangeLock.unlock();
}
}
/**
Saca dinheiro da conta bancária.
@param amount a quantia a ser sacada.
*/
public void withdraw(double amount)
throws InterruptedException
{
_____________________________________________________;
try
{
while (balance < amount)
_____________________________________________________;
System.out.print("Sacando " + amount);
double newBalance = balance - amount;
System.out.println(", o novo saldo é " + newBalance);
balance = newBalance;
lessThanMaxBalanceCondition.signalAll();
}
finally
{
balanceChangeLock.unlock();
}
}
/**
Retorna o saldo atual da conta bancária.
@return o saldo alual
*/
public double getBalance()
{
```

```
return balance;
}
}
```

*Arquivo **DepositRunnable**.java*

```
/**
Um objeto runnable para deposito faz depositos periódicos para conta
bancária.
*/
public class DepositRunnable _____________________________
{
private static final int DELAY = 1;
private BankAccount account;
private double amount;
private int count;
/**
Constrói um objeto runnable para depositos.
@param anAccount a conta na qual é depositado o dinheiro
@param anAmount a quantia a ser depositada em cada repetição
@param aCount o numero de repetições
*/
public DepositRunnable(BankAccount anAccount, double anAmount,
int aCount)
{
account = anAccount;
amount = anAmount;
count = aCount;
}
public void run()
{
try
{
for (int i = 1; i <= count; i++)
{
_____________.deposit(amount);
_____________________________;
}
}
catch (InterruptedException exception) {}
}
}
```

*Arquivo **WithdrawRunnable**.java*

```
/**
Um objeto runnable para saques que faz retiradas periódicas de uma conta
bancária.
*/
public class WithdrawRunnable implements Runnable
{
private static final int DELAY = 1;
private BankAccount account;
private double amount;
private int count;
/**
Constrói um objeto runnable para saques.
@param anAccount a conta na qual é sacado o dinheiro
@param anAmount a quantia a ser retirada em cada repetição
@param aCount o numero de repetições
*/
```

```
public WithdrawRunnable(BankAccount anAccount, double anAmount,
int aCount)
{
account = anAccount;
amount = anAmount;
count = aCount;
}
public void run()
{
try
{
for (int i = 1; i <= count; i++)
{
____________________.withdraw(amount);
_____________________________________;
}
}
catch (InterruptedException exception) {}
}
}
```