INSTITUTO MAUÁ DE TECNOLOGIA

MAUÁ

ECM225 – Sistemas Operacionais

Projeto

Threads em Java

Prof. Marco Furlan

Maio/2021

Instruções:

- **Esta atividade** tem **peso 5** nas notas de trabalho;
- **Implementar o projeto em Java** seu ambiente de desenvolvimento preferido (VSCode, IntelliJ IDEA, Netbeans, etc);
- **Compactar** a **pasta contendo** os **pacotes** e os **arquivos Java desenvolvidos** e **enviar** no **link** indicado no **OpenLMS**.

Elaborar um programa em Java que **simule** a **situação** em que, em um **banco**, onde uma **conta-corrente** é **compartilhada** por **quatro clientes**. Cada **cliente** pode **realizar** um **depósito** ou uma **retirada** (jogue uma moeda antes de executar e escolha …).

O **banco não aceita** que **retiradas** sejam **executadas** quando a **conta não possui saldo** suficiente (**não** se admite **saldo negativo**). Os **clientes** podem **continuamente** realizar **depósitos** ou **retiradas** e **cabe** ao seu programa **sincronizar** como isso será feito, de modo a **manter** a **consistência** e **integridade** da conta-corrente.

Requisitos do programa:

- A **conta-corrente** deverá ser **modelada** em uma **classe denominada** `Account`, possuir um **campo privado** para armazenar o **saldo** denominado `balance`, um **construtor** que permitirá a **criação** da **conta** com um **saldo inicial**, um **método** para **realizar depósito** denominado `deposit()` e um **método** para **realizar retiradas** denominado `withdraw()`. A **conta é um recurso compartilhado pelos seus clientes**;
- Os **clientes** serão modelados pela **classe** `Client`, que deverá ter um **campo privado** denominado `name` (cadeia de caracteres com o **nome** do **cliente**) e um **campo privado** denominado `account`, para **armazenar** uma **referência** da **conta**. Como **métodos**, a classe deverá possuir um **construtor** com **dois parâmetros**, um para **obter** o **nome** do **cliente** e **outro** para **associar** com uma **conta existente**, que deverá **ser criada** no programa principal e, **por fim**, um **método** denominado `execute()` que deverá **sortear uma** de **duas ações possíveis** a executar na conta: ou **depositar** ou **retirar** dinheiro;
- No programa de testes, a conta deverá ser criada com um **saldo inicial**;
- Os **valores** a serem **depositados** ou **retirados** devem ser escolhidos **aleatoriamente** do **conjunto** $\{10, 20, 50, 100\}$;
- **Apresentar mensagens** na **tela** de modo que seja **possível auditar** a **conta** até o momento que o programa for encerrado;
- Utilizar o **programa** do **Produtor-Consumidor** em Java como **inspiração**.

Exemplo de funcionamento

```
Conta criada com saldo inicial de: 1000
Cliente: Augustus depositou 10
Conta: saldo atualizado de 1010
Cliente: Lucius depositou 10
Conta: saldo atualizado de 1020
Cliente: Claudius retirou 100
Conta: saldo atualizado de 920
Cliente: Tiberius retirou 100
Conta: saldo atualizado de 820
Cliente: Augustus retirou 100
Conta: saldo atualizado de 720
Cliente: Tiberius depositou 50
Conta: saldo atualizado de 770
Cliente: Lucius retirou 50
Conta: saldo atualizado de 720
Cliente: Augustus depositou 100
Conta: saldo atualizado de 820
^CTerminando a simulação...
CLiente Tiberius encerrando...
CLiente Augustus encerrando...
CLiente Claudius encerrando...
CLiente Lucius encerrando...
```

Exemplo do programa principal:

```
package bank;
import sun.misc.Signal;

public class Main {
    public static void main(String[] args) {
        Account account = new Account(1000);
        Customer customers[] = {
            new Customer("Augustus", account),
            new Customer("Lucius", account),
            new Customer("Claudius", account),
            new Customer("Tiberius", account)};

        Signal.handle(new Signal("INT"), // CTRL+C
                (Signal signal) -> {
                    System.out.println("Terminando a simulação...");
                    for (Customer customer : customers) {
                        customer.interrupt();
                    }
                });
        for (Customer customer : customers) {
            customer.start();
        }
    }
}
```