ExameN/Desenvolvimento/PadraoCommand

(3 valores) Considere o padrão *Command* e as classes `BankAccount` e `Bank`:

```
public interface Command {
    void execute();
    void unExecute();
}

public class BankAccount {
    private final String iban;
    private String owner;
    private double balance;

    //construtor, acessores e modificadores

    public void changeBalance(double amount) {
        balance += amount;
    }
}

public class Bank {
    private final Map<String, BankAccount> accounts;

    //...
    public boolean exists(String iban) {
        return accounts.containsKey( iban );
    }

    public double getBalance(String iban) {
        if(!exists(iban)) return -1;

        return accounts.get(iban).getBalance();
    }

    public boolean credit(String iban, double amount) {
        accounts.get(iban).changeBalance( amount );
    }

    public boolean debit(String iban, double amount) {
        accounts.get(iban).changeBalance( -amount );
    }
}
```

a) (1 valor) Sendo `Bank` o participante *Receiver*, implemente o comando `CommandTransfer` que permite transferir dinheiro entre duas contas `from` e `to`, dados os seus números *iban*.

O comando deverá conter toda a lógica da operação, incluindo a validação dos números de conta e verificação dos saldos para garantir o sucesso das operações; não são permitidos saldos negativos.

O comando também deverá permitir "reverter" a operação; tente evitar ao máximo código duplicado.

b) (1 valor) Crie a classe `BankManager` que assumirá o papel do participante *Invoker*; esta classe deve permitir reverter o último comando executado.

c) (1 valor) Elabore um método `main()` onde ilustre a utilização de todas as classes desenvolvidas.

Resolução

```
public interface Command {
    void execute();
    void unExecute();
}
```

```
/* a) */
public class CommandTransfer implements Command {

    private Bank bank; /* receiver */
    private String ibanFrom, ibanTo;
    private double amount;

    public CommandTransfer(Bank bank, String ibanFrom, String ibanTo, double amount) {
        this.bank = bank;
        this.ibanFrom = ibanFrom;
        this.ibanTo = ibanTo;
        this.amount = amount;
    }

    public void execute() {
        if(!bank.exists(ibanFrom)) return;
        if(!bank.exists(ibanTo)) return;

        /* getBalance -- método avisado por "chat" que poderiam utilizar*/
        if(!bank.getBalanceOf(ibanFrom) < amount) return;

        bank.debit(ibanFrom, amount);
        bank.credit(ibanTo, amount);

    }

    public void unExecute() {
        if(!bank.exists(ibanFrom)) return;
        if(!bank.exists(ibanTo)) return;

        /* getBalance -- método avisado por "chat" que poderiam utilizar*/
        if(!bank.getBalanceOf(ibanTo) < amount) return;

        bank.debit(ibanTo, amount);
        bank.credit(ibanFrom, amount);
    }

    /* Refactoring: extract method das verificações */
}

/* b) */
public class BankManager {
    private Stack<Command> commands;

    public BankManager() {
        commands = new Stack<>();
    }

    public void executeCommand(Command c) {
        commands.push(c);
        commands.execute();
    }

    public void undo() {
        if(!commands.isEmpty()) {
            Command c = commands.pop();
            c.unExecute();
        }
    }
}

/* c) */
public static void main(String[] args) {
    Bank bank = new Bank();
    /* assumir que contém contas */
    BankManager manager = new BankManager();

    CommandTransfer c = new CommandTransfer(bank, "1234", "9876", 100);
```

```
        manager.executeCommand(c);

        manager.undo();
    }
```