Exercício projeto bancov03. Exercício para ser feito em classe.

1) Abra o eclipse e crie um projeto chamado bancov04 a partir do projeto bancov03. Mantenha as classes Cliente, Conta e TestaAgregacao2.
2) Faça as seguintes modificações na classe TestaAgregacao2 (a partir da linha 10) e em seguida renomeie para TestaEncapsulamento (Botão direito do mouse → refactor → rename).

```
2  public class TestaEncapsulamento {
3      public static void main(String[] args) {
4          Conta umaConta = new Conta();
5          //umaConta.titular=new Cliente();
6          umaConta.numero=1;
7          umaConta.deposita(500);
8          umaConta.titular.nome="Rafael";
9          umaConta.titular.cpf="12345678910";
10         umaConta.titular.email="rafael@cefet-rj.br";
11
12         umaConta.saca(1001); // vai impedir
13         umaConta.deposita(-100); //vai impedir
14         System.out.println("Numero: "+umaConta.numero);
15         System.out.println("Saldo: "+umaConta.saldo);
16
17         //O sistema vai impedir a linha abaixo???
18         umaConta.saldo=-2000;
19         System.out.println("Numero: "+umaConta.numero);
20         System.out.println("Saldo: "+umaConta.saldo);
21     }
22 }
```

Reflita sobre as linhas 18 a 20. Só deveríamos poder alterar o saldo mediante operações bancárias. O atributo saldo não deveria estar exposto dessa forma.

3) Na classe Conta, acrescente o modificador de acesso private à frente do saldo (`private double saldo;`). Salve o código e olhe novamente para o TestaEncapsulamento.

```
2  public class TestaEncapsulamento {
3      public static void main(String[] args) {
4          Conta umaConta = new Conta();
5          //umaConta.titular=new Cliente();
6          umaConta.numero=1;
7          umaConta.deposita(500);
8          umaConta.titular.nome="Rafael";
9          umaConta.titular.cpf="12345678910";
10         umaConta.titular.email="rafael@cefet-rj.br";
11
12         umaConta.saca(1001); // vai impedir
13         umaConta.deposita(-100); //vai impedir
14         System.out.println("Numero: "+umaConta.numero);
15         System.out.println("Saldo: "+umaConta.saldo);
16
17         //O sistema vai impedir a linha abaixo???
18         umaConta.saldo=-2000;
19         System.out.println("Numero: "+umaConta.numero);
20         System.out.println("Saldo: "+umaConta.saldo);
21     }
22 }
```

Observe as linhas 15, 18 e 20.

Na linha 18 percebemos que nosso problema está resolvido. Não é mais possível acessar diretamente o atributo saldo. Ele agora é privado e só podemos modificar seu valor por meio das operações bancárias que são métodos public. No entanto, as linhas 15 e 20 não compilam. Afinal, precisamos exibir o saldo, mas não temos mais acesso direto a esse atributo. Como resolver o problema?

4) Na classe Conta, crie um método público que retorne o valor do atributo saldo. Assim esse método será acessível fora da classe Conta, assim como já acontece com saca, deposita e transferePara.

```
7     //métodos acessores
8     public double obtemSaldo() {
9         return this.saldo;
10    }
```

Em seguida, use o novo método na classe de testes e comente a linha 18. Essa operação realmente não é mais permitida!

```
2  public class TestaEncapsulamento {
3      public static void main(String[] args) {
4          Conta umaConta = new Conta();
5          //umaConta.titular=new Cliente();
6          umaConta.numero=1;
7          umaConta.deposita(500);
8          umaConta.titular.nome="Rafael";
9          umaConta.titular.cpf="12345678910";
10         umaConta.titular.email="rafael@cefet-rj.br";
11
12         umaConta.saca(1001); // vai impedir
13         umaConta.deposita(-100); //vai impedir
14         System.out.println("Numero: "+umaConta.numero);
15         System.out.println("Saldo: "+umaConta.obtemSaldo());
16
17         //O sistema vai impedir a linha abaixo???
18         //umaConta.saldo=-2000;//OPERAÇÃO PROIBIDA
19         System.out.println("Numero: "+umaConta.numero);
20         System.out.println("Saldo: "+umaConta.obtemSaldo());
21     }
22 }
```

Problema resolvido!! No entanto, não é boa prática deixar os atributos expostos dessa forma. De certa maneira, quando não declaramos um modificador de acesso para um atributo, o Java assume que ele é acessível fora da classe. Assume que ele é public. Vamos alterar o nível de visibilidade de número e titular e criar métodos “assessores” para estes atributos.

5) Altere numero e titular para private e crie os métodos obtemNumero, modificaNumero e obtemTitular. Veja o código abaixo:

```
2  public class Conta {
3      // Atributos
4      private Cliente titular = new Cliente();
5      private double saldo = 500;
6      private int numero;
7      //métodos acessores
8      public double obtemSaldo() {
9          return this.saldo;
10     }
11     public int obtemNumero() {
12         return this.numero;
13     }
14     public Cliente obtemTitular() {
15         return this.titular; //retorna um endereço de memória
16     }
17     public void modificaNumero(int novoNumero) {
18         if(novoNumero<=0)
19             return;//Early return
20         this.numero=novoNumero;
21     }
22     //Comportamento
```

CONSIDERAÇÕES IMPORTANTES:

- **Linhas 17 a 21** → Quando o acesso é indireto, podemos estabelecer regras para validar a informação que vai ser armazenada em um atributo. Não podemos aceitar um número de conta negativo.
- **Linha 19** → Se a condição não nos atende, não precisamos executar todo o método. Basta sair de forma precoce. Técnica conhecida como Early Return. Converse com seu professor (Eu mesmo) a respeito.

- **Por que não criamos um modificaTitular?** Isso depende muito do contexto da aplicação e das regras de negócio. No nosso banco não deve ser possível modificar o titular de uma Conta. Creio que no mundo real também é assim. Esse é o famoso princípio da imutabilidade. Uma vez inicializado, o atributo não pode mais sofrer modificações.

6) Agora faça os devidos ajustes na classe TestaEncapsulamento.

```
2  public class TestaEncapsulamento {
3      public static void main(String[] args) {
4          Conta umaConta = new Conta();
5          //umaConta.titular=new Cliente();
6          umaConta.modificaNumero(1);
7          umaConta.deposita(500);
8          umaConta.obtemTitular().nome="Rafael";
9          umaConta.obtemTitular().cpf="12345678910";
10         umaConta.obtemTitular().email="rafael@cefet-rj.br";
11
12         umaConta.saca(1001); // vai impedir
13         umaConta.deposita(-100); //vai impedir
14         System.out.println("Numero: "+umaConta.obtemNumero());
15         System.out.println("Saldo: "+umaConta.obtemSaldo());
16
17         //O sistema vai impedir a linha abaixo???
18         //umaConta.saldo=-2000;//OPERAÇÃO PROIBIDA
19         System.out.println("Numero: "+umaConta.obtemNumero());
20         System.out.println("Saldo: "+umaConta.obtemSaldo());
21     }
22 }
```

Observe nas linhas 8 a 10 que para ter acesso ao objeto cliente apontado pelo atributo titular, precisamos utilizar o método obtemTitular. Esse método vai retornar justamente o endereço de memória onde "vive" o objeto Cliente em questão.

7) Apesar de estar correto e funcionando, existe uma convenção em Java para nomear métodos que retornam e modificam seus atributos. Em vez de obtem e modifica, devemos utilizar, respectivamente, get e set. Altere todo o código. As classes deverão ficar conforme a ilustração abaixo.

Conta

```
2  public class Conta {
3      // Atributos
4      private Cliente titular = new Cliente();
5      private double saldo = 500;
6      private int numero;
7      //métodos acessores
8      public double getSaldo() {
9          return this.saldo;
10     }
11     public int getNumero() {
12         return this.numero;
13     }
14     public Cliente getTitular() {
15         return this.titular; //retorna um endereço de memória
16     }
17     public void setNumero(int numero) {
18         if(numero<=0)
19             return;//Early return
20         //atributo numero = valor do argumento número
21         this.numero=numero;
22     }
23     //Comportamento
```

Nas linhas 17,20 e 21, observe a importância da palavra chave this. Se não entender, pergunte ao seu professor.

TestaEncapsulamento

```java
public class TestaEncapsulamento {
    public static void main(String[] args) {
        Conta umaConta = new Conta();
        //umaConta.titular=new Cliente();
        umaConta.setNumero(1);
        umaConta.deposita(500);
        umaConta.getTitular().nome="Rafael";
        umaConta.getTitular().cpf="12345678910";
        umaConta.getTitular().email="rafael@cefet-rj.br";

        umaConta.saca(1001); // vai impedir
        umaConta.deposita(-100); //vai impedir
        System.out.println("Numero: "+umaConta.getNumero());
        System.out.println("Saldo: "+umaConta.getSaldo());

        //O sistema vai impedir a linha abaixo???
        //umaConta.saldo=-2000;//OPERAÇÃO PROIBIDA
        System.out.println("Numero: "+umaConta.getNumero());
        System.out.println("Saldo: "+umaConta.getSaldo());
    }
}
```

8) Já vimos que devemos impedir o acesso direto aos atributos de uma classe (modificador private) e ao mesmo tempo devemos promover o acesso indireto e controlado a esses atributos por meio de métodos “acessores” (métodos public onde há a possibilidade de validar um valor antes de guarda-lo no atributo. Portanto, faça essas alterações na classe cliente. Para criar os getters and setters, use CTRL+3 → ggas. Se não lembrar, pergunte ao seu professor (eu mesmo). :-D

```java
public class Cliente {
    String nome;
    String cpf;
    String email;
    //Métodos acessores
    public String getNome() {
        return this.nome;
    }
    public void setNome(String nome) {
        if(nome.length()<5)
            return;
        this.nome = nome;
    }
    public String getCpf() {
        return this.cpf;
    }
    public void setCpf(String cpf) {
        if(cpf.length()!=11)
            return;
        this.cpf = cpf;
    }
    public String getEmail() {
        return this.email;
    }
    public void setEmail(String email) {
        if(email.contains("@")==false)
            return;
        this.email = email;
    }
```

9) Crie a classe TestaEncapsulamento2 conforme exemplo abaixo. Utilize os métodos set e teste valores inválidos para todos os métodos set de Cliente e Conta.

```java
public class TestaEncapsulamento2 {
    public static void main(String[] args) {
        Conta umaConta = new Conta();
        //umaConta.titular=new Cliente();
        umaConta.setNumero(1);
        umaConta.deposita(500);
        umaConta.getTitular().setNome("Rafael");
        umaConta.getTitular().setCpf("12345678910");
        umaConta.getTitular().setEmail("rafael@cefet-rj.br");

        umaConta.saca(1001); // vai impedir
        umaConta.deposita(-100); //vai impedir
        System.out.println("Numero: "+umaConta.getNumero());
        System.out.println("Saldo: "+umaConta.getSaldo());
        System.out.println("Titular: "+umaConta.getTitular());
        System.out.println("Cpf: "+umaConta.getTitular().getCpf());
        System.out.println("Email: "+umaConta.getTitular().getEmail());
    }
}
```

10) Crie um método encapsulado para validar cpf na classe Cliente.

```java
    private boolean cpfEhValido(String cpf) {
        if(cpf.length()!=11)
            return false;
        return true;
    }
    public void setCpf(String cpf) {
        if(this.cpfEhValido(cpf)!=true)
            return; //Early return
        this.cpf = cpf;
    }
```

11) Qual é a razão para se ter métodos encapsulados?
12) Um método encapsulado faz parte da interface da minha classe?
13) Preciso ter um método encapsulado para cada regra?
14) Precisei alterar o código da minha classe de testes após a alteração?

Discuta essas questões com seu professor.