Aula 07 – parte 1
Exercício projeto bancov08. Exercício para ser feito em classe.

1) Abra o eclipse e crie um projeto chamado bancov08 a partir do projeto bancov07. Mantenha todas as classes.
2) Modifique a classe Conta para que ela tenha um construtor único que recebe como argumento o número da conta.

```
public class Conta {
    // Atributos
    private Cliente titular; //get (IMUTÁVEL)
    private double saldo; // get e operações bancárias
    private int numero; // get (IMUTÁVEL)
    //construtor
    public Conta(int numero) {
        this.saldo=500;
        this.titular=new Cliente();
        if(numero<=0) {
            System.out.println("Numero invalido para uma conta. A aplicacao sera encerrada.");
            System.exit(0);
        }
        this.numero=numero;
    }
    //métodos acessores
```

3) Ainda na classe Conta, faça um pequeno ajuste em nosso método saca para que não seja possível sacar um valor negativo.

```
    public boolean saca(double valor) {
        if (valor>=0 || valor > this.saldo) {
            return false;
        } else {
            this.saldo -=valor;
            return true;
        }
    }
```

4) Crie a classe ContaCorrente que é uma extensão de Conta (Herança). Perceba que o Eclipse já vai acusar um problema. ContaCorrente precisa ter um construtor compatível com o da superclasse.

```
public class ContaCorrente extends Conta {

}
```

5) Vamos resolver este problema implementando o que chamamos de compatibilidade de construtores.

```
public class ContaCorrente extends Conta {
    public ContaCorrente(int numero) {
        // Invocando o construtor da superclasse. Construtores NÃO são herdados
        super(numero);
    }
}
```

6) ContaPoupanca também é uma extensão de Conta. Crie esta classe com os mesmos cuidados.

```
public class ContaPoupanca extends Conta {
    public ContaPoupanca(int numero) {
        // Invocando o construtor da superclasse. Construtores NÃO são herdados
        super(numero);
    }
}
```

7) Coloque uma mensagem em cada construtor. Veja como ficaria em Conta e faça o mesmo para ContaCorrente e ContaPoupança. Nas subclasses você vai perceber que essa instrução tem que ser colocada após a compatibilidade de construtores. A instrução super(numero); deve ser a 1ª linha do construtor das subclasses.

```
    public Conta(int numero) {
        System.out.println("Instanciando um objeto do tipo Conta...");
        this.saldo=500;
        this.titular=new Cliente();
        if(numero<=0) {
            System.out.println("Numero invalido para uma conta. A aplicacao sera encerrada.");
            System.exit(0);
        }
        this.numero=numero;
    }
```

8) Crie uma classe TestaContas conforme código abaixo.

```
6  public class TestaContas {
7      public static void main(String[] args) {
8          ContaCorrente cc = new ContaCorrente(1);
9          cc.deposita(6000);
10         cc.saca(1000);
11
12         ContaPoupanca cp = new ContaPoupanca(2);
13         cp.deposita(4000);
14
15         cc.transferePara(cp, 500);
16         System.out.println("Saldo de cc: "+cc.getSaldo());//5000
17         System.out.println("Saldo de cp: "+cp.getSaldo());//5000
18     }
19 }
```

9) Em nossa classe Conta, vamos dar uma boa olhada no método transferePara e fazer algumas considerações.

```
public boolean transferePara(Conta contaDestino, double valor) {
    //conta1 --> #abc -->this -->#abc
    if(this.saca(valor)==true) {
        boolean conseguiuDepositar = contaDestino.deposita(valor);
        return conseguiuDepositar; // O retorno será true, o mesmo do método deposita
    }
    return false;
}
```

Perceba que o método pode receber como 1º argumento, tanto uma ContaCorrente quanto uma ContaPoupanca. Afinal, ambas são Contas! ContaDestino é uma Conta e invoca métodos da classe Conta (saca e deposita). Quando nós criamos esse método, utilizamos polimorfismo sem nem mesmo ter conhecimento sobre esse poderoso recurso da orientação a objetos.

10) Nova regra. A cada saque em ContaCorrente devemos descontar uma taxa de 0,50 para cada saque. Obviamente precisaremos reescrever o método saca em ContaCorrente implementando todas as suas regras. Não se esqueça de modificar o nível de acesso dos atributos contidos em Conta para protected. Senão teremos problemas com o this.saldo. Faça isso.

```
3  public class ContaCorrente extends Conta {
4      public ContaCorrente(int numero) {
5          // Invocando o construtor da superclasse. Construtores NÃO são herdados
6          super(numero);
7      }
8
9      @Override
10     public boolean saca(double valor) {
11         if (valor>=0 || (valor+0.50) > this.saldo) {
12             return false;
13         } else {
14             this.saldo -=valor;
15             return true;
16         }
17     }
```

11) Se olharmos atentamente, perceberemos que a solução acima não é muito boa. A regra a respeito do saque agora está duplicada. Ela existe na classe Conta e na classe ContaCorrente. Se a regra mudar, teremos que lembrar de alterar em dois lugares. Implemente uma solução mais elegante que reaproveita regras contidas no método saca da superclasse.

```
9      @Override
10     public boolean saca(double valor) {
11         if(valor<=0)
12             return false;
13         return super.saca(valor+0.50);
14     }
15 }
```

12) Rode a classe de testes e perceba que tudo funciona normalmente.