==========

Programação Orientada a Objetos
Atividade 3
Herança e Polimorfismo
Fonte: Java e Orientação a Objetos – Caelum

==========

1. Vamos criar uma classe Conta, que possua um saldo, e os métodos para pegar saldo, depositar e sacar.

2. Adicione um método na classe Conta, que atualiza essa conta de acordo com uma taxa percentual fornecida.

3. Crie duas subclasses da classe Conta: *ContaCorrente* e *ContaPoupanca*. Ambas terão o método atualiza reescrito: A *ContaCorrente* deve atualizar-se com o dobro da taxa e a *ContaPoupanca* deve atualizar-se com o triplo da taxa. Além disso, a *ContaCorrente* deve reescrever o método deposita, a fim de retirar uma taxa bancária de dez centavos de cada depósito.

4. Crie uma classe com método *main* e instancie essas classes, atualize-as e veja o resultado. Algo como:

```
class TestaContas {
    public static void main(String[] args) {
        Conta c = new Conta();
        ContaCorrente cc = new ContaCorrente();
        ContaPoupanca cp = new ContaPoupanca();

        c.deposita(1000);
        cc.deposita(1000);
        cp.deposita(1000);

        c.atualiza(0.01);
        cc.atualiza(0.01);
        cp.atualiza(0.01);

        System.out.println(c.getSaldo());
        System.out.println(cc.getSaldo());
        System.out.println(cp.getSaldo());

    }
}
```

   Após imprimir o saldo (*getSaldo()*) de cada uma das contas, o que acontece?

5. O que você acha de rodar o código anterior da seguinte maneira:

```
Conta c = new Conta();
Conta cc = new ContaCorrente();
Conta cp = new ContaPoupanca();
```

   Compila? Roda? O que muda? Qual é a utilidade disso? Realmente, essa não é a maneira mais útil do polimorfismo - veremos o seu real poder no próximo exercício. Porém existe uma utilidade de declararmos uma variável de um tipo menos específico do que o objeto realmente é.

   É **extremamente importante** perceber que não importa como nos referimos a um objeto, o método que será invocado é sempre o mesmo! A JVM vai descobrir em tempo de execução qual deve ser invocado, dependendo de que tipo é aquele objeto e não de acordo com como nos referimos a ele.

6. Vamos criar uma classe que seja responsável por fazer a atualização de todas as contas bancárias e gerar um relatório com o saldo anterior e saldo novo de cada uma das contas.

```
class AtualizadorDeContas {
    private double saldoTotal = 0;
    private double selic;

    AtualizadorDeContas(double selic) {
        this.selic = selic;
    }

    void roda(Conta c) {
        // aqui voce imprime o saldo anterior, atualiza a conta,
        // e depois imprime o saldo final
        // lembrando de somar o saldo final ao atributo saldoTotal
    }

    // outros métodos, colocar o getter para saldoTotal!
}
```

7. No método *main*, vamos criar algumas contas e rodá-las:

```
class TestaAtualizadorDeContas {
    public static void main(String[] args) {
        Conta c = new Conta();
        Conta cc = new ContaCorrente();
        Conta cp = new ContaPoupanca();

        c.deposita(1000);
        cc.deposita(1000);
        cp.deposita(1000);

        AtualizadorDeContas adc = new AtualizadorDeContas(0.01);

        adc.roda(c);
        adc.roda(cc);
        adc.roda(cp);

        System.out.println("Saldo Total: " + adc.getSaldoTotal());
    }

}
```

8. Use a palavra-chave *super* nos métodos atualiza reescritos, para não ter de refazer o trabalho.

9. Se você precisasse criar uma classe *ContaInvestimento*, e seu método atualiza fosse complicadíssimo, você precisaria alterar as classes *Banco* e *AtualizadorDeContas*?

10. Crie uma classe Banco que possui um *array* de Conta. Repare que num *array* de Conta você pode colocar tanto *ContaCorrente* quanto *ContaPoupanca*. Crie um método *void adiciona(Conta c),* um método *Conta pegaConta(int x)* e outro *int pegaTotalDeContas(),* muito similar a relação anterior de Empresa-Funcionario. Faça com que seu método *main* crie diversas contas, insira-as no Banco e depois, com um for, percorra todas as contas do *Banco* para passá-las como argumento para o *AtualizadorDeContas.*