UFRR

Programação Orientada a Objetos
Atividade em Sala – Herança e Polimorfismo
Professor: Filipe Dwan Pereira

1. Vamos criar uma classe Conta, que possua um saldo os métodos para pegar saldo, depositar e sacar.
    a. Crie a classe Conta:
    b. Adicione o atributo saldo
    c. Crie os métodos getSaldo(), deposita(double) e saca(double)
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.

- Crie as classes ContaCorrente e ContaPoupanca. Ambas são filhas da classe Conta:
- Reescreva o método atualiza na classe ContaCorrente, seguindo o enunciado. Repare que, para acessar o atributo saldo herdado da classe Conta, você vai precisar trocar o modificador de visibilidade de saldo para protected.
- Reescreva o método atualiza na classe ContaPoupanca, seguindo o enunciado:
- Na classe ContaCorrente, reescreva o método deposita para descontar a taxa bancária de dez centavos:

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

```
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("Saldo Conta: "+c.getSaldo());
System.out.println("Saldo Conta Corrente: "+cc.getSaldo());
System.out.println("Saldo Conta Poupança: "+cp.getSaldo());
```

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, não importando 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. Além disso, conforme atualiza as contas, o banco quer saber quanto do dinheiro do banco foi atualizado até o momento. Por isso, precisamos ir guardando o saldoTotal e adicionar um getter à classe:

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

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

    public void roda(Conta c) {
        // aqui você 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:

```
public 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 a classe 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 public void adiciona(Conta c), um método public Conta pegaConta(int x) e outro public 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.

Atividades Extra (Não avaliativa):

Crie duas classes: Ponto2D e Ponto3D. Ponto2D possui como atributos as coordenadas x e y, enquanto Ponto3D, além delas, também possui a coordenada z. Utilize a relação de herança para representar estas classes.

UFRR

1- A respeito dos construtores, Ponto2D deve ter apenas um construtor, que recebe os valores de x e y como parâmetros (tipo double). Já Ponto3D também deve ter apenas um construtor, que deve receber x, y e z como parâmetros (também do tipo double). Dica: Se a relação de herança e a declaração dos construtores foram feitas corretamente, você deverá, obrigatoriamente, chamar o construtor da superclasse explicitamente.

Ambas as classes devem sobrescrever o método toString(), que é originalmente declarado na classe Object. Este método deve retornar uma representação do objeto em forma de String, indicando qual o valor de cada coordenada. É importante que Ponto3D tire proveito do método toString() de Ponto2D para mostrar os valores das coordenadas x e y.

2- Crie uma classe Veiculo com um atributo ligado (privado), que indica se o carro está ligado ou não. Esta classe deve ter também os métodos ligar() e desligar(), que definem valor para este atributo, e um método getter (isLigado()).

Depois crie três subclasses de Veiculo: Automovel, Motocicleta e Onibus. Cada classe destas deve sobrescrever os métodos ligar() e desligar() e deve imprimir mensagens como “Automóvel ligado”, “Motocicleta desligada”, etc. Para manter a consistência do modelo, descubra como fazer para que o atributo ligado de Veiculo tenha o valor correto quando os métodos são chamados.

Crie uma aplicação que instancia três veículos, um de cada tipo, e chama os métodos ligar(), desligar() e isLigado(). O resultado obtido deve ser consistente com o que o modelo representa. Por exemplo, ao chamar o método ligar() de um Automovel, é esperado que o método isLigado() retorne true.