

## Exercício Prático 2 - 16/03/2017

Programação de Computadores 2 - Programação Orientada a Objetos  
Classes, objetos e arranjos

Prof. Daniel Hasan Dalip – hasan@decom.cefetmg.br

### Como enviar os exercícios:

Crie um projeto para o exercício. Crie um pacote por exercício, cada pacote chamará **exercícioNum** em que Num é o número do exercício. Após finalizado os exercícios, salve o projeto em um zip e envie-o pelo Moodle (Entrega Roteiro Aula Prática 3). Não será aceito o recebimento via e-mail. Plágio não será tolerado e tanto a pessoa que forneceu quanto a que utilizou perderão os pontos da prática. Entrega até: 22/03/2017

**Envio de arquivo com pacotes com nome diferente acarretará em perda de 50% dos pontos.**

**Exercício 1:** Implemente uma classe Analista que tenha as seguintes características: Nome, matrícula (String), nível (júnior, pleno, sênior), salário (1500 – júnior, 3000 – pleno, 5000 – sênior) e adicional (*double*).

O construtor da classe deverá receber como parâmetro o nome e matrícula do analista. O analista, ao ser criado, deve ter seu nível, salário e promoção inicializados com um valor *default* (nível = *júnior*, adicional = 0). A classe deverá ter métodos:

- *Get* e *set* quando for apropriado (os atributos nome e matrícula não são alteráveis). E o seu *toString*.
- O método *setNivel* deverá receber como parâmetro o novo nível, atualizar o nível e o salário correspondente.
- Deverá haver um método *getSalárioTotal* que retorna a soma do salário e do adicional.

Implemente uma aplicação (uma outra classe que possua um método *main*) que :

- a. Crie um analista *p1* com nome *José*;
- b. Dê um adicional de 500.0 a *p1*;
- c. Exiba os dados de *p1*;
- d. Crie um analista *p2* com nome *João*;
- e. Dê um adicional de 300.0 a *p2*;
- f. Exiba os dados de *p2*.
- g. Crie um analista *p3* com nome *Maria*;
- h. Dê um adicional de 800.0 a *p3*;
- i. Exiba os dados de *p3*;
- j. Promova *p2* a *pleno*;
- k. Promova *p3* a *pleno*;
- l. Exiba os dados de *p2* e de *p3*;
- m. Faça *p1=p3*;
- n. Exiba os dados de *p1*.
- o. Promova *p1* a *sênior*;
- p. Exiba os dados de *p1* e *p3*. **O que ocorre e por que? (Responda essa pergunta deixando um comentário no código)**

**Exercício 2:** Implemente as seguintes classes e métodos:

a) Classe Data com atributos dia, mês e ano (todos inteiros). Crie o construtor, métodos de acesso e modificadores (get e set) além do toString.

a) Crie o método getNomeMes (sem parâmetros) que retorna uma string representando o mês de acordo com o seu número. Para isso, utilize o switch, por exemplo:

```
switch(x){  
    case 1:  
        return "blah";  
    case 2:  
        return "oioi";  
}
```

c) Adicione, na classe Analista do exercício anterior o atributo dataAdmissao do tipo Data. Modifique o construtor para receber esta data. Modifique o toString. Nele, deve ser exibido a data de admissão no seguinte formato: "12 de janeiro de 2016". Utilize o método getNomeMes para isso. Execute novamente a classe de teste com o código modificado.

### Exercício 3 [depuração de código (debug)]:

Uma das tarefas que podem gastar muito tempo no desenvolvimento de Softwares é a depuração de erros. Para isso, as IDEs possuem um forma de depurar o erro em que o desenvolvedor pode acompanhar a execução do programa passo a passo. Neste exercício, iremos apresentar um programa que possui erros. Você deverá usar a ferramenta de depuração do Eclipse ou do Netbeans para encontrá-los e corrigi-los.

Considere a seguinte implementação de uma TV que possui o volume.

```
public class Volume {  
    private int id;  
    private int altura;  
    private boolean estereo;  
    private static int ultId = 1;  
    public Volume(int altura, boolean stereo){  
        this.altura = altura;  
        this.stereo = stereo;  
        id = ultId;  
        ultId++;  
    }  
    (...)  
}
```

```
public class Televisao {  
    private int id;  
    private int canal;  
    private Volume volume;  
    private static int ultId = 1;  
    public Televisao(int c, Volume vol){  
        canal = c;  
        volume = vol;  
        id = ultId;  
        ultId++;  
    }  
    (...)  
}
```

```
public class Teste {  
    public static void alteraVolume(Volume vol){  
        vol.setVolume(12);  
    }  
    public static void main(String[] args){  
        Volume vol1 = new Volume(20,vol1);  
        Volume vol2 = new Volume(15,vol2);  
        Televisao tv1 = new Televisao(22,vol1);  
        Televisao tv2 = new Televisao(10,vol2);  
        //vol1 = vol2;  
        alteraVolume(vol1);  
        System.out.println(tv1);  
        System.out.println(tv2);  
    }  
}
```

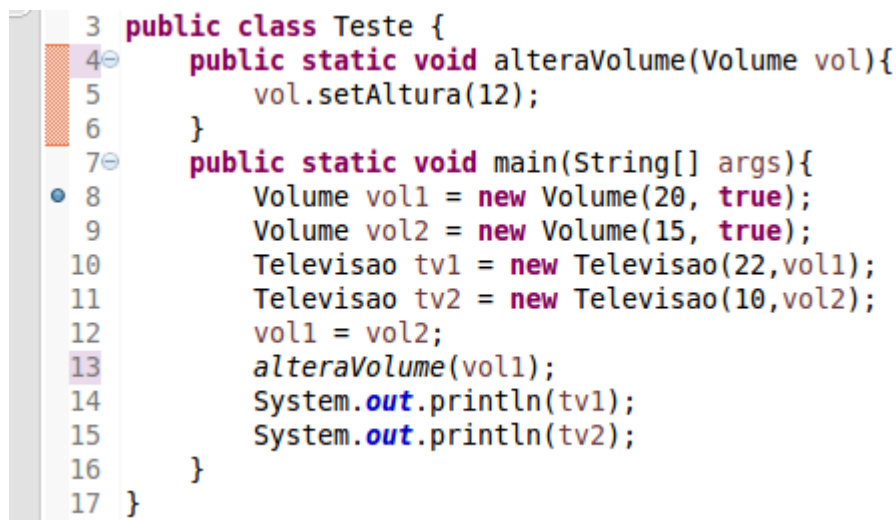
Implemente o código. Faça o toString do volume e da televisão mostrando, inclusive, a altura do volume e o id tanto da televisão quanto do volume.

Entenda o resultado e, através do debug, observe o porquê do valor diferente quando comentamos ou não a linha "vol1 = vol2". A seguir, é demonstrado como usar o debug.

### Utilização do DEBUG

Para usar o debug, tanto no eclipse quando no netbeans, primeiramente adicionamos breakpoints, que são pontos em que a execução do código será pausada para possamos investigar valores de variáveis e a pilha de execução do programa.

No Eclipse, para criar um breakpoint, clique duas vezes no número da linha do código desejada. Aparecerá um círculo nela:

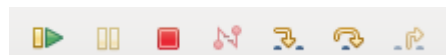





```
3 public class Teste {
4     public static void alteraVolume(Volume vol){
5         vol.setAltura(12);
6     }
7     public static void main(String[] args){
8         Volume vol1 = new Volume(20, true);
9         Volume vol2 = new Volume(15, true);
10        Televisao tv1 = new Televisao(22,vol1);
11        Televisao tv2 = new Televisao(10,vol2);
12        vol1 = vol2;
13        alteraVolume(vol1);
14        System.out.println(tv1);
15        System.out.println(tv2);
16    }
17 }
```

Logo após, clique para executar a depuração:  
debug as... → Java Application



Ao começar a executar, no canto superior esquerdo, irá aparecer a pilha de execução do programa, ou seja, qual linha o programa irá executar. No canto superior direito, é apresentado as variáveis do programa e seus valores. Para continuar a execução passo a passo do programa, use a seguinte barra de controle:



Nesta barra, o play irá executar o programa até o próximo breakpoint. Pause, pausa o programa e stop interrompe o mesmo. O botão seguinte não será necessário, é para depuração remota. O próximo botão "step into" (  ) entra no método que será executado na próxima instrução. O botão "step over" (  ) executa o próximo método por completo e pausa na instrução seguinte. O "step return" (  ) termina de executar o método atual e pausa em seguida.

**Perspectiva no eclipse:** quando você começa a depurar, a perspectiva (ou seja, a configuração de visualização do eclipse) é alterada para "debug" você pode alterar novamente para perspectiva Java no botão no canto direito superior ou pelo menu: Janela → Abrir Perspectiva → Java