

LISTA DE EXERCÍCIOS

Nome: Ana Carla da Silva Vidal

- 1) Considere a implementação da classe `Funcionario` (Figura 1). Suponha agora que você precise modelar um funcionário comissionado. Um funcionário comissionado tem um salário-base, mais uma pequena comissão por venda. Implemente a classe *FuncionarioComissionado* considerando disponibilizar métodos para cálculo de salário e controle de vendas. Implemente também uma aplicação que solicite os dados dos funcionários, sejam eles comissionados ou não, e exiba o salário a ser pago a um funcionário consultado.

```
1
2 public class Funcionario {
3     private String primeiro_nome;
4     private String segundo_nome;
5     private double salario;
6
7     public Funcionario(String primeiro_nome, String segundo_nome, double salario) {
8         this.primeiro_nome = primeiro_nome;
9         this.segundo_nome = segundo_nome;
10        this.salario = salario;
11    }
12
13    public String getPrimeiro_nome() {
14        return primeiro_nome;
15    }
16
17    public void setPrimeiro_nome(String primeiro_nome) {
18        this.primeiro_nome = primeiro_nome;
19    }
20
21    public String getSegundo_nome() {
22        return segundo_nome;
23    }
24
25    public void setSegundo_nome(String segundo_nome) {
26        this.segundo_nome = segundo_nome;
27    }
28
29    public double getSalario() {
30        return salario;
31    }
32
33    public void setSalario(double salario) {
34        this.salario = salario;
35    }
36 }
```

Figura 1

2) Indique os erros das classes abaixo:

- a) A herança de classes deve ser precedida pela palavra “Super”, pelo conceito não faz sentido, pois um Ponto3D não é um Ponto2D, e portanto Ponto3D não pode ser composto do Ponto2D.

```
1 class Ponto2D
2 {
3     private double x,y;
4     Ponto2D(double _x,double _y)
5     {
6         x = _x; y = _y;
7     }
8 }
9
10 class Ponto3D extends Ponto2D
11 {
12     private double z;
13     Ponto3D(double _x,double _y,double _z)
14     {
15         x = _x;
16         y = _y;
17         z = _z;
18     }
19 }
```

- b) Java não permite herança múltipla. Seria mais viável a utilização da chamada toString que retorna uma string representando o objeto, sem a necessidade do super.

```
1 class DataHora extends Data,Hora
2 {
3     public DataHora(byte d,byte m,short a,byte hor,byte min,byte seg)
4     {
5         super(d,m,a);
6         super(hor,min,seg);
7     }
8     public String toString()
9     {
10         return super.toString()+" "+super.toString();
11     }
12 }
```

- c) O método construtor na classe “Filho” onde está o super, não é o método construtor da classe.

```
1 class Pai
2 {
3     private char x,y;
4     Pai(char _x,char _y)
5     {
6         x = _x; y = _y;
7     }
8 }
9
10 class Filho extends Pai
11 {
12     private char z;
13     public void inicializa(char _x,char _y,char _z)
14     {
15         super(_x,_y);
16         z = _z;
17     }
18 }
```

- d) O super deve estar na primeira chamada do método construtor.

```
1 class DataHora extends Data
2 {
3     private Hora hora;
4     public DataHora(byte d,byte m,short a,byte hor,byte min,byte seg)
5     {
6         hora = new Hora(hor,min,seg);
7         super(d,m,a);
8     }
9     public String toString()
10    {
11        return super.toString()+" "+hora.toString();
12    }
13 }
```

- 3) O que cada trecho de código faz?

- a) Suponha que a seguinte chamada de método esteja presente em um método *earnings()* sobrescrito em uma subclasse: **super.earnings()**;

Invoca o método da superclasse

- b) Suponha que a seguinte linha de código apareça como a primeira instrução no corpo de um construtor: **super(primeiroArgumento, segundoArgumento)**;

Está recebendo os arquivamentos e passando para superclasse as variáveis da superclasse. E tem que está no método onde chama a superclasse.