Disciplina: Programação Orientada à Objeto
Prof(a). Odara Sena
BCC - S4

LISTA DE EXERCÍCIOS

Nome: Arthur Germano de Oliveira

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. Implementa 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;
0          this.salario = salario;
1      }
2
3      public String getPrimeiro_nome() {
4          return primeiro_nome;
5      }
6
7      public void setPrimeiro_nome(String primeiro_nome) {
8          this.primeiro_nome = primeiro_nome;
9      }
0
1      public String getSegundo_nome() {
2          return segundo_nome;
3      }
4
5      public void setSegundo_nome(String segundo_nome) {
6          this.segundo_nome = segundo_nome;
7      }
8
9      public double getSalario() {
0          return salario;
1      }
2
3      public void setSalario(double salario) {
4          this.salario = salario;
5      }
6
7
8  }
```

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.

```
class Ponto2D
  {
  private double x,y;
  Ponto2D(double _x,double _y)
    {
    x = _x; y = _y;
    }
  }

class Ponto3D extends Ponto2D
  {
  private double z;
  Ponto3D(double _x,double _y,double _z)
    {
    x = _x;
    y = _y;
    z = _z;
    }
  }
```

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.

```
class DataHora extends Data,Hora
  {
  public DataHora(byte d,byte m,short a,byte hor,byte min,byte seg)
    {
    super(d,m,a);
    super(hor,min,seg);
    }
  public String toString()
    {
    return super.toString()+" "+super.toString();
    }
  }
```

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

```
class Pai
  {
  private char x,y;
  Pai(char _x,char _y)
    {
    x = _x; y = _y;
    }
  }

class Filho extends Pai
  {
  private char z;
  public void inicializa(char _x,char _y,char _z)
    {
    super(_x,_y);
    z = _z;
    }
  }
```

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

```
class DataHora extends Data
  {
  private Hora hora;
  public DataHora(byte d,byte m,short a,byte hor,byte min,byte seg)
    {
    hora = new Hora(hor,min,seg);
    super(d,m,a);
    }
  public String toString()
    {
    return super.toString()+" "+hora.toString();
    }
  }
```

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.