

Site: <https://sites.google.com/site/profricodemery/mpoo>

Site: <http://ava.ufrpe.br/>

Disciplina: Modelagem e Programação Orientada a Objetos (MPOO)

Profº: Richarlyson D'Emery

LISTA DE EXERCÍCIOS III

Leia atentamente as instruções gerais:

- No Eclipse crie um novo **projeto chamado br.edu.mpoo.listalll.SeuNomeSobrenome**, o qual deverá ter **pastas de pacotes** para cada questão: questao1, questao2, e assim sucessivamente, contendo todas as respostas da lista.
- Quando a questão envolver uma discussão teórica utilize um arquivo .txt (Menu File -> Submenu New -> Opção File), por exemplo, questao1.txt
- A lista envolve questões práticas e conceituais, então deverão ser entregues no AVA tanto os códigos-fonte (projeto completo) quanto às demais respostas. Em caso de imagens e digramas, você poderá salvar o arquivo também na pasta correspondente do projeto.
- A entrega da lista compõe sua frequência e avaliação na disciplina.

Fique atento!

Método construtor é um método que inicializa os atributos da classe. O nome do método construtor deverá ser o mesmo nome da classe.

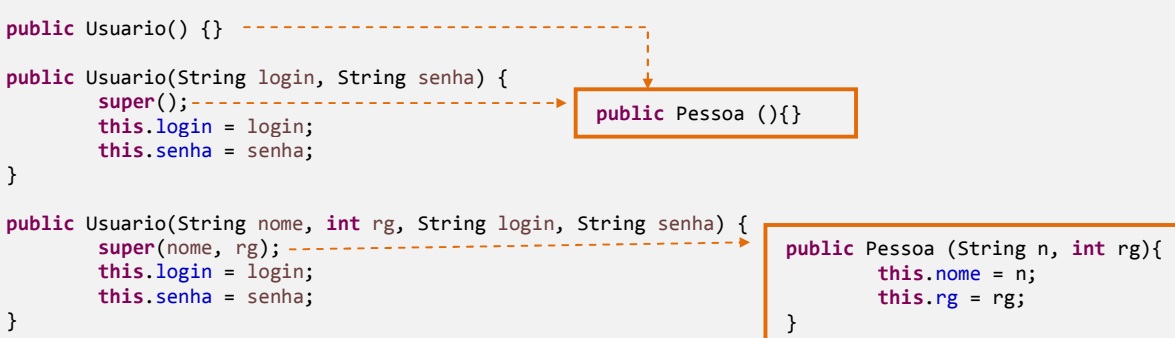
```
public class Pessoa{  
    String nome;  
    int rg;  
  
    public Pessoa (String n, int rg){  
        this.nome = n;  
        this.rg = rg;  
    }  
}
```

Em **Herança**, quando uma superclasse define um método construtor, logo o método **default deixa de existir!** Mas se houver a necessidade da existência de um "método construtor default" então se deve declará-lo!

```
public class Pessoa{  
    String nome;  
    int rg;  
  
    public Pessoa (){}  
  
    public Pessoa (String n, int rg){  
        this.nome = n;  
        this.rg = rg;  
    }  
}
```

Em **Herança**, subclasses podem definir pelo menos um construtor herdado para não ocorrer em erro de sintaxe. Entretanto, sugere-se que subclasses definam construtores para todos os herdados.

```
public class Usuario extends Pessoa{  
    String login;  
    String senha;  
  
    public Usuario() {}  
  
    public Usuario(String login, String senha) {  
        super();  
        this.login = login;  
        this.senha = senha;  
    }  
  
    public Usuario(String nome, int rg, String login, String senha) {  
        super(nome, rg);  
        this.login = login;  
        this.senha = senha;  
    }  
}
```



1) Preencha as lacunas:

- 1.1) Se a classe Pessoa herda da classe Animal, a classe Pessoa é chamada de _____ e a classe Animal é chamada de _____.
- 1.2) O conceito de herança permite a _____, que economiza tempo no desenvolvimento e estimula a utilização de programas previamente testados.
- 1.3) Quando uma classe é utilizada com o mecanismo de herança, ela se torna uma superclasse que fornece _____ e _____ para outras classes ou se torna uma subclasse.
- 1.4) O relacionamento “é um” entre as classes representa o conceito de _____, enquanto o relacionamento “tem um” entre classes representa _____.

2) Analise o código abaixo e aponte mudanças necessárias. As correções devem ser justificadas.

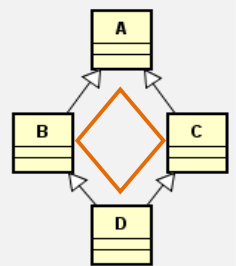
```
1 package questao3;
2
3 public class Conta{
4     private int num;
5     private double saldo;
6
7     public int Conta (int n, double saldo) {
8         num = n;
9     }
10
11     public void debito (double valor) {
12         this.saldo-=valor;
13     }
14
15     public void credito (double valor) {
16         this.saldo+=valor;
17     }
18 }
```

```
1 package questao3;
2
3 public class Poupanca extends Conta{
4     public Poupanca (int num){
5         super (num, saldo);
6     }
7
8     public void debito (double valor) {
9         this.saldo-=valor;
10    }
11
12    public void rendeJuros(){
13        this.saldo+=saldo*taxa/100;
14    }
15 }
```

Você Sabia?

Em POO, o conceito de Herança em Java só permite a [herança simples](#), diferentemente de outras linguagens como, por exemplo, Python que permite que uma classe herde de mais de uma classe, ou seja, permite a implementação de [herança múltipla](#).

Mas, uma classe ao herdar de várias classes não apenas podem herdar propriedades completamente diferentes, complicando-se quando superclasses possuem mesmos métodos ou atributos. Essa ambiguidade é conhecida como o [problema do diamante](#) (ou problema do losango), e diferentes linguagens resolvem esse problema de maneiras diferentes. O Python segue uma ordem específica para percorrer a hierarquia de classes, chamada de Ordem de Resolução de Métodos (MRO, do inglês [Method Resolution Order](#)), fazendo com que a escolha pelo método ou atributo seja dada a partir da ordem da explicitação da generalização, ou seja, a ordem será sempre da esquerda para direita:



```
class Subclasse(ClasseOrdem1,ClasseOrdem2,..., Classe OrdemN)
```

3) Sabemos que em Java não há herança múltipla, mas em outras linguagens sim. Vejamos um exemplo em Python:

```
#pai.py
class Pai:
    def __init__(self, nome, sobrenome='DEmery'):
        self.nome = nome
        self.sobrenome = sobrenome

    def metodo(self):
        print ('pai')
```

```
#mae.py
class Mae:
    def __init__(self, nome, sobrenome='A lves'):
        self.nome = nome
        self.sobrenome = sobrenome

    def metodo(self):
        print ('mae')
```

```
#filho.py

from pai import Pai
from mae import Mae

class Filho(Pai,Mae):
    def __init__(self, cpf, nome):
        super().__init__(nome)
        self.cpf=cpf
```

```
#main.py
from filho import Filho
from pai import Pai
from mae import Mae

filho = Filho('111.111.111-11', nome='Rico')
pai = Pai(nome='Emerson')
mae = Mae(nome='Sueli')
```

Responda:

- 3.1) Em #main.py, qual a saída para `print (filho.sobrenome)`?
- 3.2) Em #main.py, qual a saída para `filho.metodo()`?
- 3.3) Análise e explique o que acontece na herança implementada.

Você Sabia?

Uso de **downcast** em POO: Uma solução ao comportamento de Herança.

```
// SuperClasse.java
public class SuperClasse {
    private int atr_SuperClasse;

    public int getAtr_SuperClasse() { return atr_SuperClasse; }

    public void setAtr_SuperClasse(int atr_SuperClasse) {
        this.atr_SuperClasse = atr_SuperClasse;
    }
}

// SubClasse.java
public class SubClasse extends SuperClasse{
    private int atr_SubClasse;

    public int getAtr_SubClasse() { return atr_SubClasse; }

    public void setAtr_SubClasse(int atr_SubClasse) {
        this.atr_SubClasse = atr_SubClasse;
    }
}

//(continua...)
```

```
// App.java
public class App {
    public static void main(String[] args) {
        SubClasse subClasse = new SubClasse();
        System.out.println(subClasse.getAtr_SuperClasse());
        System.out.println(subClasse.getAtr_SubClasse());
    }
}

/*
 * Questionamento: e se subClasse fosse do tipo SuperClasse? como acessar o atr_SubClasse?
 * Solução: Usar downcast.
 */

SuperClasse subClasse2 = new SubClasse();
System.out.println(subClasse2.getAtr_SuperClasse());
System.out.println(((SubClasse)subClasse2).getAtr_SubClasse()); //solução
}
```

Fique atento!

Uso de **downcast** em POO: Uma solução ao comportamento de Herança.

```
// SuperClasse.java
public class SuperClasse {
    private int atr_SuperClasse;

    public int getAtr_SuperClasse() { return atr_SuperClasse; }

    public void setAtr_SuperClasse(int atr_SuperClasse) {
        this.atr_SuperClasse = atr_SuperClasse;
    }
}

// SubClasse.java
public class SubClasse extends SuperClasse{
    private int atr_SubClasse;

    public int getAtr_SubClasse() { return atr_SubClasse; }

    public void setAtr_SubClasse(int atr_SubClasse) {
        this.atr_SubClasse = atr_SubClasse;
    }
}

// App.java
import java.util.ArrayList;

public class App {
    public static void main(String[] args) {
        /*
         * Questionamento: e se subClasse fosse do tipo SuperClasse? como acessar o atr_SubClasse?
         * Solução: Usar downcast.
         */

        // Usando ArrayList:
        ArrayList<SuperClasse> superClasses = new ArrayList<SuperClasse>();
        superClasses.add(new SuperClasse());
        superClasses.add(new SubClasse());

        System.out.println(superClasses.get(0).getAtr_SuperClasse());
        System.out.println(superClasses.get(1).getAtr_SuperClasse());
        System.out.println(((SubClasse)superClasses.get(1)).getAtr_SubClasse()); //solução
    }
}
```

Você Sabia?

Se, em tempo de execução, a referência de um objeto de subclasse tiver sido atribuída a uma variável de uma das suas superclasses diretas ou indiretas, é aceitável fazer **downcast** da referência armazenada nessa variável de superclasse de volta a uma referência do tipo da subclasse. Antes de realizar essa coerção, utilize o operador **instanceof** para assegurar que o objeto é de fato um objeto de um tipo de subclasse apropriado.

```
SubClasse subClasse = new SubClasse();
SuperClasse superClasse;
superClasse = subClasse;
if(superClasse instanceof SubClasse) //PREVINA-SE: USE instanceof
    System.out.println(((SubClasse) superClasse).getAtr_SubClasse());
```

Mas atenção!



É um erro comum de programação atribuir uma variável de superclasse a uma variável de subclasse (sem uma coerção explícita) é um erro de compilação.

```
SubClasse subClasse = new SuperClasse();
```

Saiba Mais!

Em Java podemos utilizar o `for` de uma maneira especial, chamada de `enhanced for`, ou popularmente `foreach`. Lembrando que `foreach` não existe no Java como **comando**, mas como um caso especial do `for`

```
for (Classe classe: arrayList) {    System.out.println (classe); }
```

Fique Atento!

Utilizando `foreach` e **instanceof** podemos reformular App:

```
//SuperClasse.java
public class SuperClasse {
    private int atr_SuperClasse;

    public int getAtr_SuperClasse() { return atr_SuperClasse; }

    public void setAtr_SuperClasse(int atr_SuperClasse) {
        this.atr_SuperClasse = atr_SuperClasse;
    }

    @Override
    public String toString() {
        return "[atr_SuperClasse=" + atr_SuperClasse + "]";
    }
}

//SubClasse.java
public class SubClasse extends SuperClasse{
    private int atr_SubClasse;

    public int getAtr_SubClasse() {    return atr_SubClasse;}

    public void setAtr_SubClasse(int atr_SubClasse) {
        this.atr_SubClasse = atr_SubClasse;
    }

    @Override
    public String toString() {
        return "[atr_SubClasse=" + atr_SubClasse + "," + super.toString() + "]";
    }
}
//(continua...)
```

```
// App.java
import java.util.ArrayList;

public class App {
    public static void main(String[] args) {

        // Usando ArrayList:
        ArrayList<SuperClasse> superClasses = new ArrayList<SuperClasse>();
        superClasses.add(new SuperClasse());
        superClasses.add(new SubClasse());

        // Usando foreach e instanceof:
        int cont=0;
        for (SuperClasse superClasseCorrente:superClasses){
            System.out.println("Element: [" + cont + "]");
            if(superClasseCorrente instanceof SuperClasse)
                System.out.println(superClasseCorrente.getAtr_SuperClasse());
            if(superClasseCorrente instanceof SubClasse)
                System.out.println(((SubClasse)superClasseCorrente).getAtr_SubClasse());
            cont++;
        }

        // Usando vinculação dinâmica com toString():
        cont=0;
        for (SuperClasse superClasseCorrente:superClasses){
            System.out.println("Element: [" + cont + "]");
            System.out.println(superClasseCorrente.toString());
            cont++;
        }
    }
}
```

Lembre-se:

Devido ao polimorfismo de objetos, o processo de vinculação dinâmica faz com que toString () seja resolvido em tempo de execução (em vez de em tempo de compilação) de acordo com o objeto que o invoca.

4) Responda V se verdadeiro ou F se Falso. Justifique se falso.

- 4.1) () O objeto de uma subclasse pode ser tratado como um objeto de sua superclasse, mas o contrário não é verdadeiro.
- 4.2) () Uma superclasse representa um número maior de membros que sua subclasse
- 4.3) () O objeto de uma subclasse também é um objeto da superclasse dessa subclasse.

5) O que é toString()? E qual sua relação com Object?

6) Para as situações abaixo, crie diagramas de classes e use case e faça a devida codificação em Java:

- 6.1) O sistema de um Supermercado possui Funcionários, onde cada um possui matrícula, RG, nome, função, senha. Quando o funcionário possui a função “gerente”, ele poderá dar um desconto no total de uma compra. Possui Produtos, onde cada um possui nome, código e preço. Uma Compra possui um total, uma nota e descrição. Possui uma operação totalizar onde acrescenta ao total da compra o preço de um produto informado. A cada produto registrado na compra a descrição (nome do produto) e o seu valor é registrado na nota da compra (onde terá um detalhamento de todos os produtos comprados e o valor total da compra).

Sugestão:

- A operação registrar repassa o nome e o preço à nota após cada produto registrado.
- A operação resumir compra exibe toda a nota da compra, incluindo o total da compra.

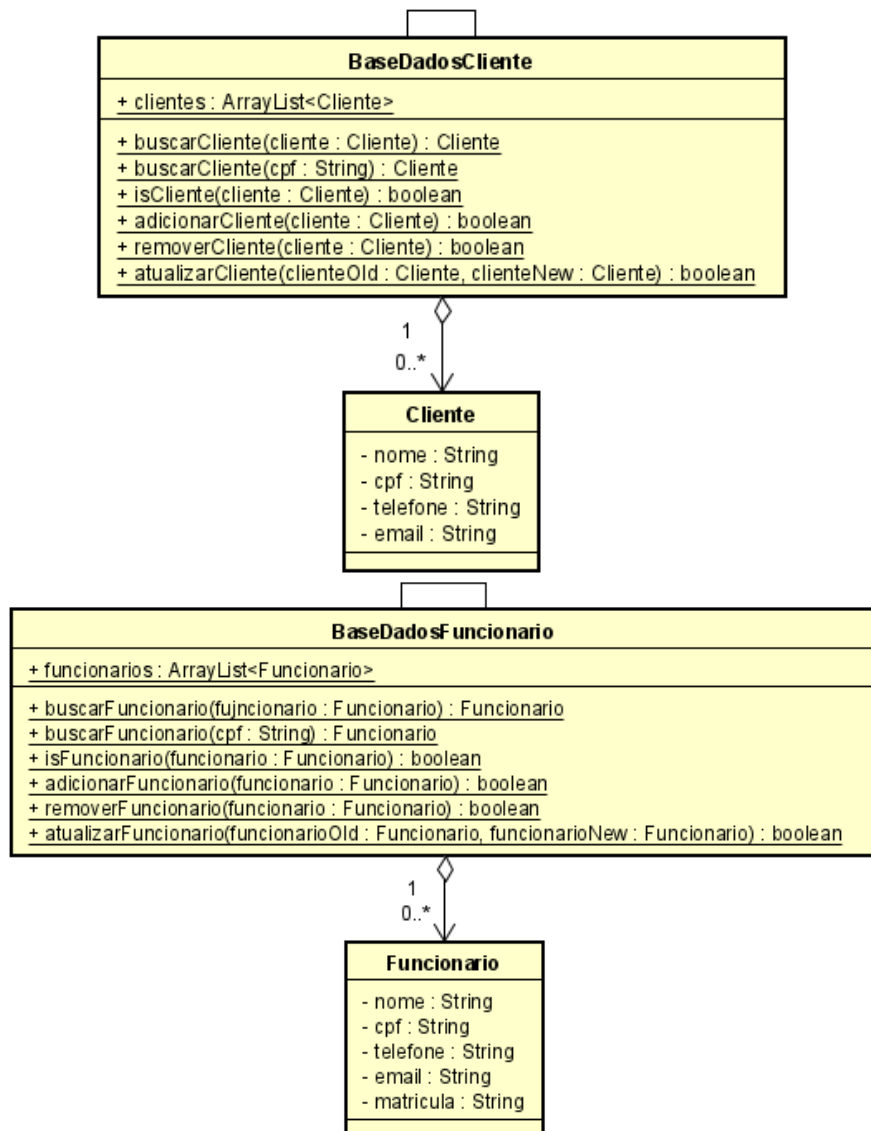
- 6.2) Crie uma classe Data que forneça a data em múltiplos formatos. Use construtores sobrecarregados para criar objetos Data inicializados com datas em diferentes formatos de apresentação.

Desafio: Mão na Massa!

Você, aluno de MPOO, está experienciando situações-problemas do universo de desenvolvimento de software e começará a ser desafiado a solucionar problemas a partir de conhecimentos de Programação e Orientação a Objetos.



- 7) Um contratante solicitou a empresa MPOOSoftware LTDA a atualização de um sistema de cadastro. O Scrum Master de MPOOSoftware LTDA solicitou a um de seus programadores (de codinome *mustela putórius furo* – “O Furão”) que resolvesse essa demanda. Para isso apresentou os seguintes diagramas de classes atuais da empresa:



Antes de responder, analise as seguintes regras de negócios:

- RN01 – um cliente ou funcionário é identificado pelo seu cpf;
- RN02 – um cliente ou funcionário só poderá ser cadastrado uma única vez;

Responda:

- 7.1) Apresente uma solução (diagrama de classes e codificação Java) de maneira a ter uma única base com uso de polimorfismo.
- 7.2) Do ponto de vista de segurança de dados, o sistema apresenta falhas. Justifique os porquês dessas falhas e como poderiam ser solucionadas. Também apresente uma solução (diagrama UML e codificação Java)
- 7.3) A partir do novo sistema, ilustre em uma aplicação o funcionamento dos serviços da base para dois clientes e dois funcionários.