UAST
Unidade Acadêmica de Serra Talhada - PE
UFRPE
Desde 2006

Bacharelado em Sistemas de Informação

MPOO

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

http://ava.ufrpe.br/

https://sigs.ufrpe.br/sigaa/ava/index.jsf

Disciplina: Modelagem e Programação Orientada a Objetos (MPOO)

Profº: Richarlyson D'Emery

LISTA DE EXERCÍCIOS V

Leia atentamente as instruções gerais:

- No Eclipse crie um novo **projeto chamado br.edu.mpoo.listaV.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!

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
        }
}
```

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); }
```

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();
```

Responda:

1) Em Java podemos utilizar a estrutura de dados lista pela classe ArrayList, mas existem outras formas em Java de armazenar uma lista das quais devem ser apresentadas e em que situações são utilizadas. Qual a melhor do ponto de vista de custo computacional?

2) Analise o código da classe SaibaMais

```
import java.util.ArrayList;

public class SaibaMais {

    public static void main(String[] args) {

        String saida1 = "Aprendendo";
        String saida2 = "a";
        String saida3 = "Programar";
        String saida4 = "Java";

        ArrayList<String> saidas = new ArrayList<>();
        saidas.add(saida1);
        saidas.add(saida2);
        saidas.add(saida3);
        saidas.add(saida4);
       //?
    }
}
```

Responda:

2.1) Qual a saída do programa quando substituímos `//?` por:

2.1.1) `System.out.println(saidas);`

2.1.2) `saidas.forEach(saida -> { System.out.print(saida + " "); });`

2.2) Pesquise qual conceito é pertencente ao Java 8 e está sendo utilizado no método nativo `forEach` da questão 2.1.2)

Fique Atento!

Utilizando `foreach` e **`instanceof`** podemos reformular App, evitando erros de **downcast** e utilizar a vinculação dinâmica de `toString` em 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() + "]";
	}
}

 // 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 superClasseCurrente:superClasses){
			System.out.println("Element: [" + cont + "]");
			if(superClasseCurrente instanceof SuperClasse)
				System.out.println(superClasseCurrente.getAtr_SuperClasse());
			if(superClasseCurrente instanceof SubClasse)
				System.out.println(((SubClasse)superClasseCurrente).getAtr_SubClasse());
			cont++;
		}

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

Fique Mais Atento!

Mas não é necessário explicitar `toString`:

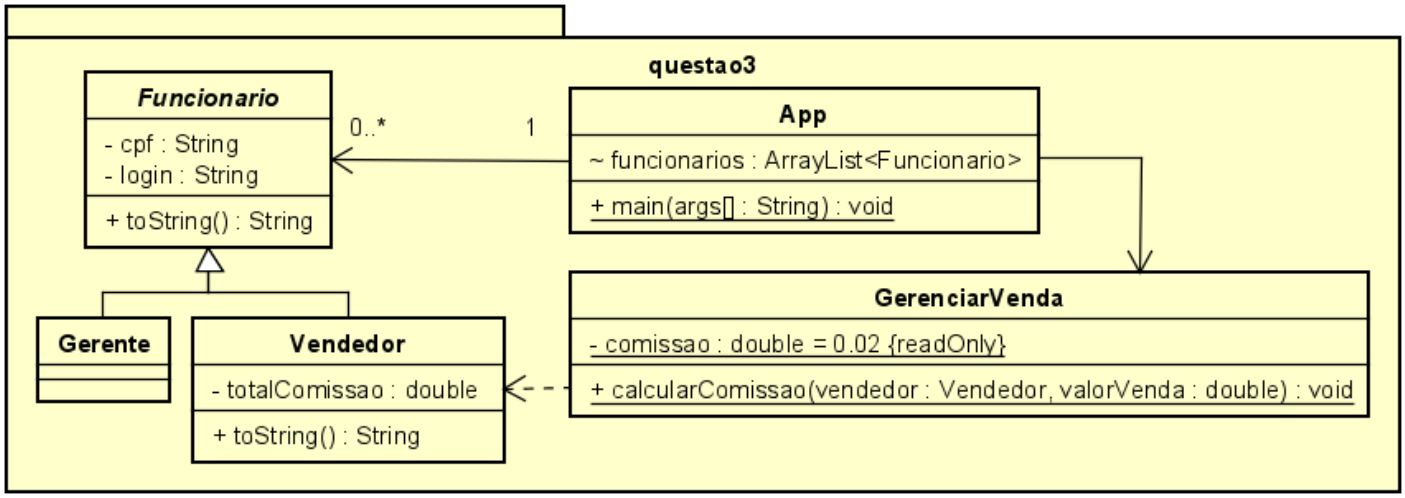
```
        // Usando vinculação dinâmica com toString():
                cont=0;
                for (SuperClasse superClasseCurrente:superClasses){
                        System.out.println("Element: [" + cont + "]");
                        System.out.println(superClasseCurrente);
              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

Lembre-se:

Mesmo que:
`superClasseCurrente.toString()`

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.

3) O diagrama de classes abaixo ilustra a vinculação dinâmica da chamada de toString para o polimorfismo entre as especializações Vendedor e Gerente e a generalização Funcionário. A codificação está demonstrada na pasta de pacote questao3 do projeto disponibilizado juntamente com esta Lista.



Na seção "Fique Atento!" foi apresentado um exemplo de polimorfismo de herança/objetos no qual realiza a vinculação dinâmica da chamada do método `toString()`. Para isso, o método foi reescrito pela sobreposição de métodos `toString` de `Object`.

3.1) Observe o erro de semântica na saída da codificação, uma vez que ao invés de "Gerente" tem-se "Funcionário". Realize a devida sobreposição de maneira a garantir a saída correta de acordo com a função definida através da Herança.

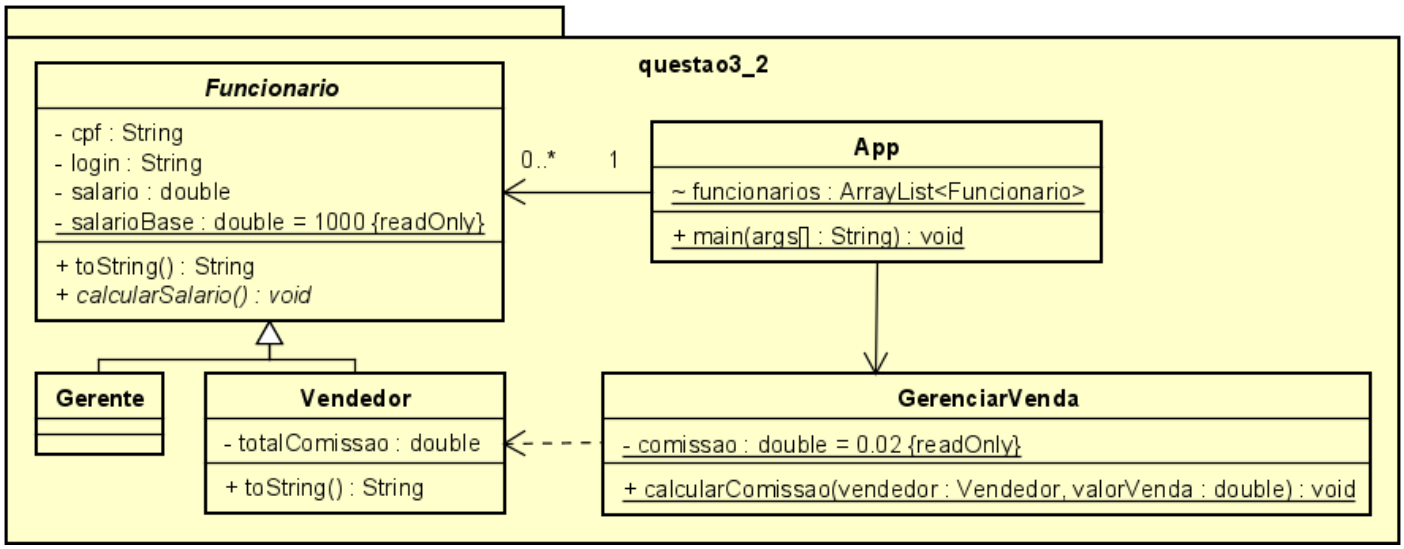
```
Funcionario [cpf=222.222.222-22, login=Ana Souza]
Vendedor [totalVendas=600.0, Funcionario [cpf=111.111.111-11, login=João Silva]]
```

3.2) Assumindo o fato de que um funcionário possui um salário e este deve ser definido a partir da sua função: um gerente recebe 5 vezes um salário base de R$1.000,00; enquanto um vendedor tem ao seu salário base a adição de uma comissão sobre as vendas realizadas. A partir da codificação disponibilizada, implemente em Java o diagrama de classes abaixo e exiba os dados (inclusive o salário) do Gerente e Vendedor (com uma venda de R$ 30.000,00) conforme ilustrado na saída:.

Saída:

```
Gerente [cpf=222.222.222-22, login=Ana Souza, salario=5000.0]
Vendedor [totalVendas=600.0, cpf=111.111.111-11, login=João Silva, salario=1600.0]
```

Diagrama de classes:
(próxima página)

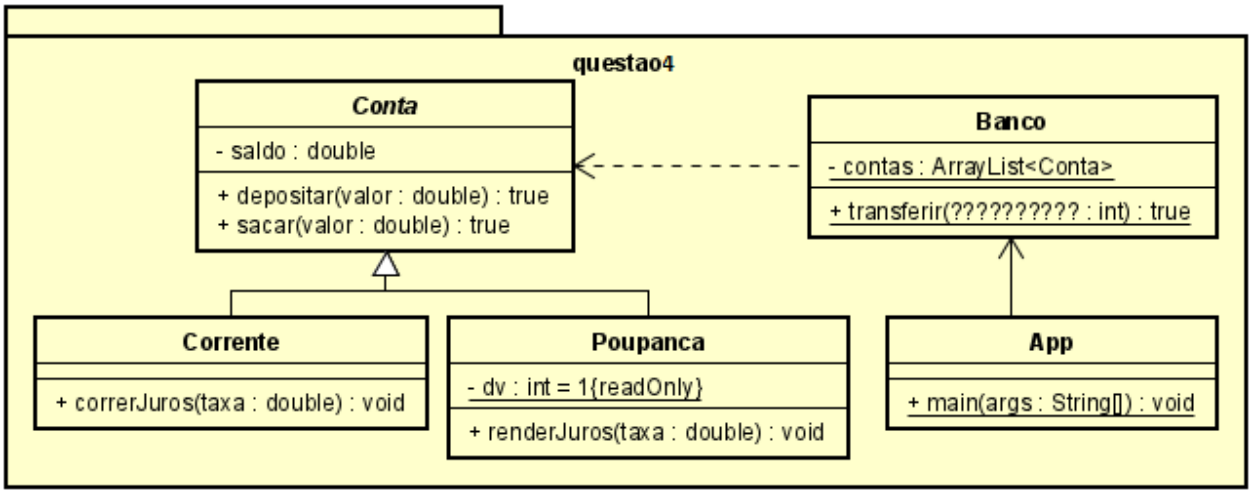


4) Comumente o brasileiro possui dois tipos de contas: uma Corrente e uma Poupança. Um banco possui diversas contas e pode realizar a transferência entre duas contas distintas (Corrente para Corrente, Corrente para Poupança, Poupança para Corrente e Poupança para Poupança). Sendo assim, quantos métodos para realizar transferências entre contas são necessários para a classe Banco? Responda codificando o problema em Java.

   Mas antes, analise as seguintes regras de negócio:

   - RN01 – O saldo de uma conta só pode ser movimentado pelos métodos depositar (adiciona um valor ao saldo) e sacar (retira um valor do saldo);
   - RN02 – Um saque só pode ser realizado se o saldo de uma conta for igual ao maior ao valor a ser retirado;
   - RN03 – Conta corrente possui o comportamento de correr juros que pode ter uma taxa a ser incidida sobre o saldo de maneira a diminuir o valor do saldo;
   - RN04 – Conta poupança possui o comportamento render juros que pode ter uma taxa a ser incidida sobre o saldo de maneira a aumentar o valor do saldo;

   É diagrama do sistema:



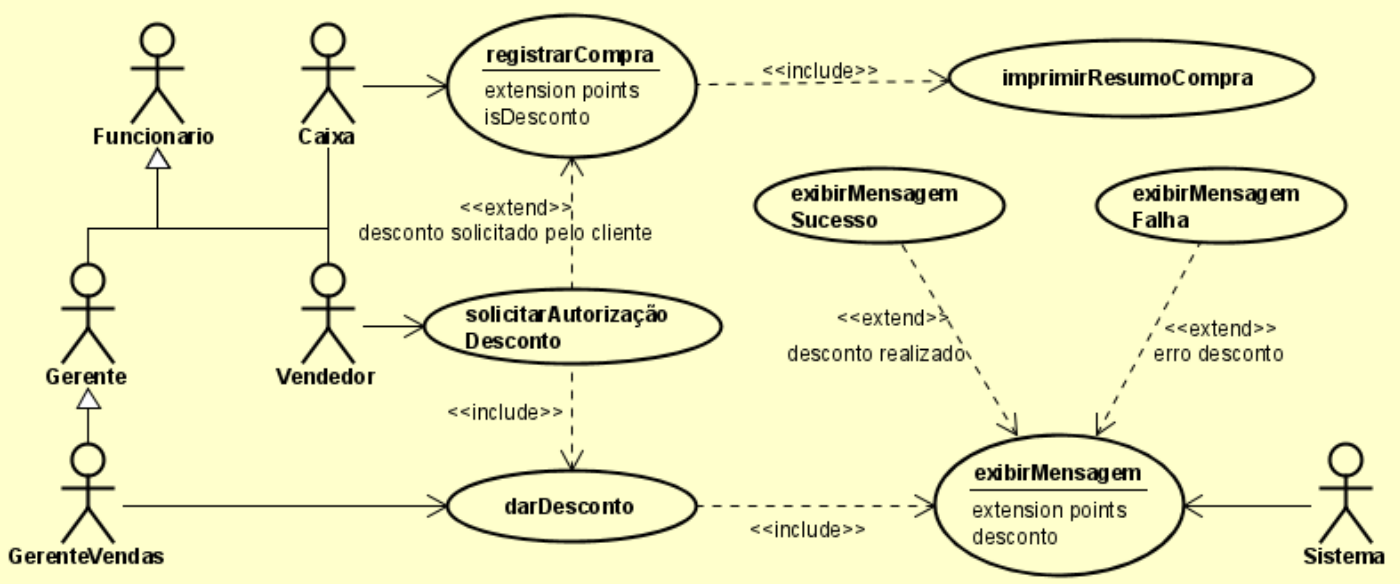
   Então codifique uma aplicação em que se demonstrada:

   - A criação de duas contas correntes e duas poupanças. (Atribua saldos distintos a essas contas);
   - Implemente a problemática de transferência entre contas;
   - Demonstre as transferências entre as contas: Corrente para Corrente, Corrente para Poupança, Poupança para Corrente e Poupança para Poupança;
   - Demonstre o rendimento de juros e a operação de correr juros em duas contas.

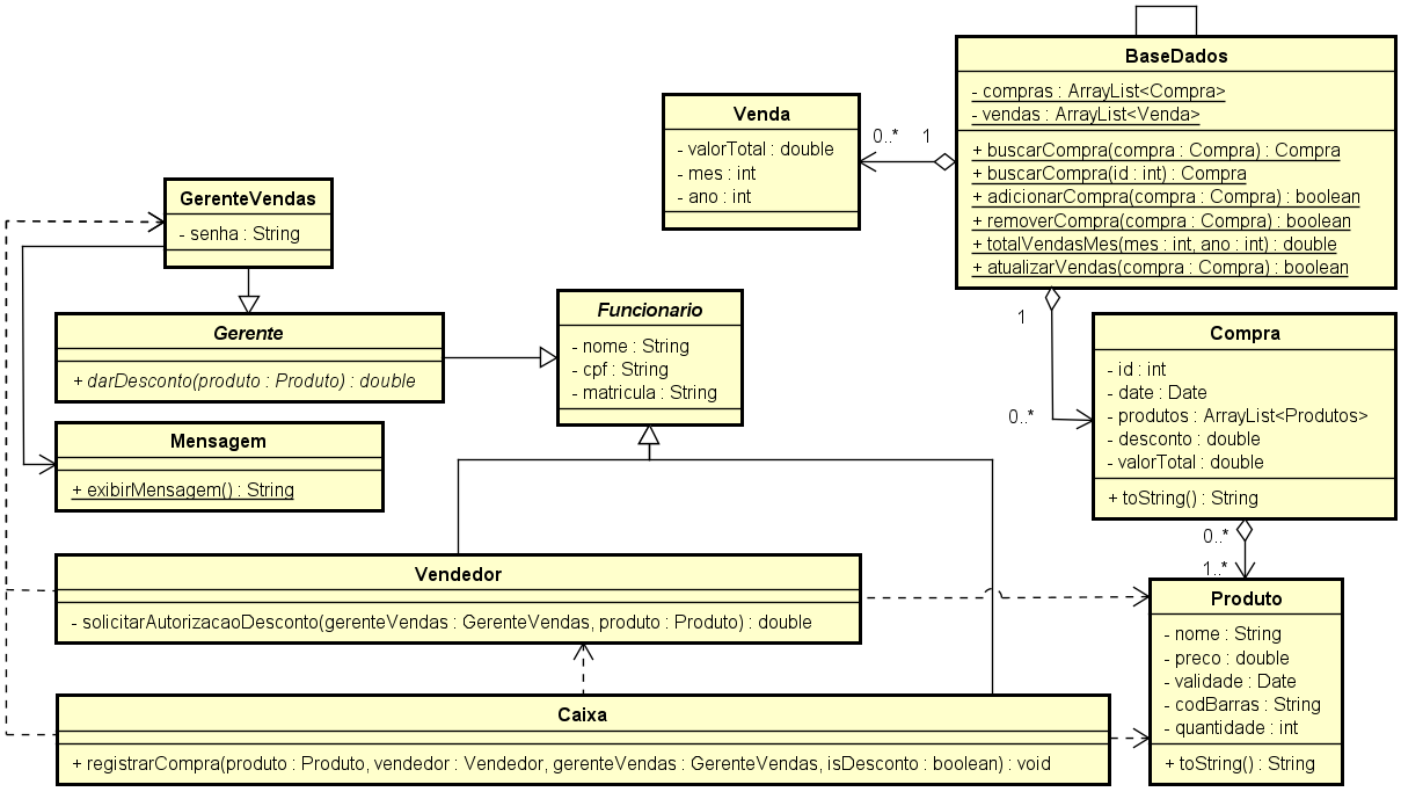
Desafio

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.

5) **(Desafio)** Uma Empresa solicitou a um de seus programadores (de codinome *mustela putórius furo* – "O Furão") que resolvesse uma demanda de funcionalidades de seu sistema criada pelo setor de venda da empresa. Cada ator detém comportamento(s) específico(s) conforme sua função ilustrada no diagrama de use case abaixo:



Sabe-se que o sistema da empresa deve ser codificado em Java e também deve refletir o diagrama de classes abaixo:



Antes de responder, analise as situações contendo as seguintes regras de negócios:

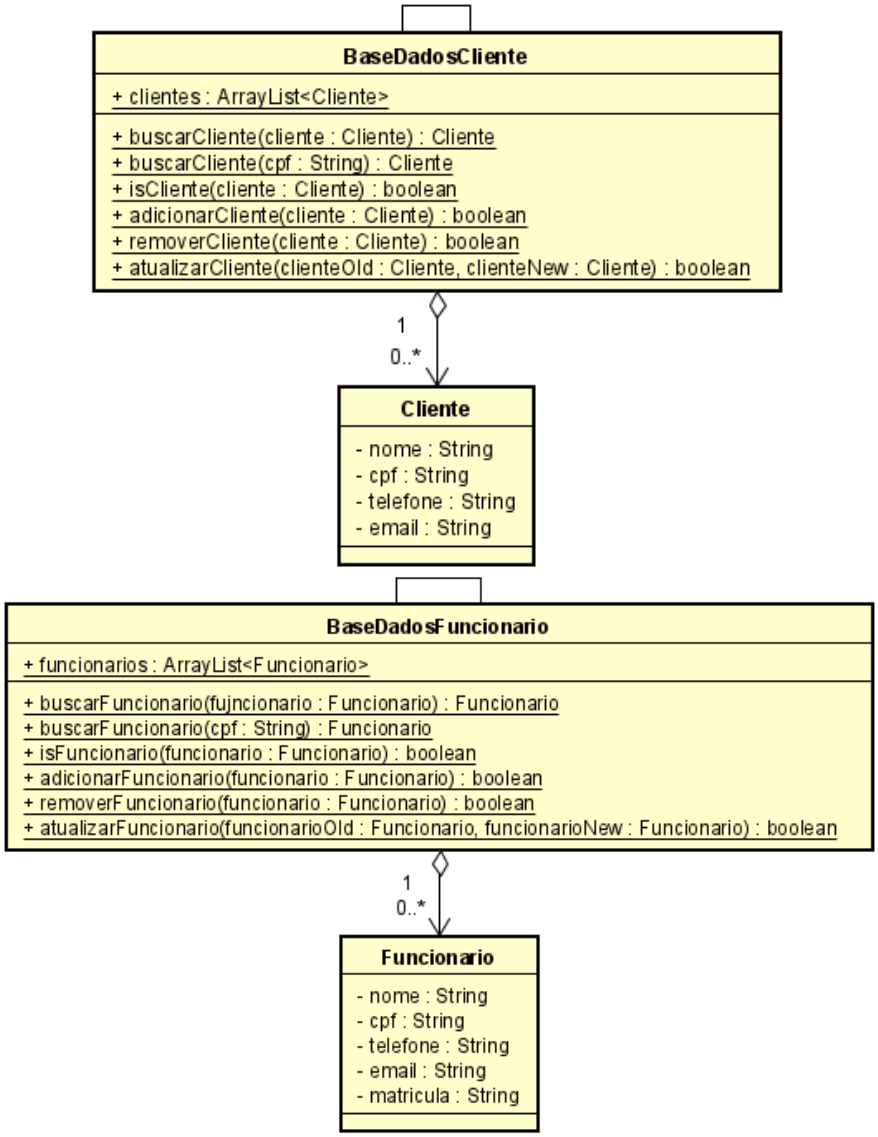
- RN01 – uma compra possui id autoincrementável;
- RN02 – a data de uma compra utiliza a data do sistema de acordo com o momento em que é realizada;
- RN03 – toda compra é registrada na base;
- RN04 – toda nova compra deve atualizar o valor de vendas de um mês;
- RN05 – toda compra cancelada deve atualizar o valor de vendas de um mês;
- RN06 – um funcionário só pode executar os comportamentos definidos;
- RN07 – a codificação deve aproveitar comportamentos já definidos, evitando a duplicidade de programação;
- RN08 – uma compra tem identificação autoincrementável;
- RN09 – um desconto só é atribuído a uma compra se for solicitado verbalmente por um cliente no momento do registro de uma compra;
- RN10 – Um desconto ou solicitação de autorização de desconto só poderá ser executado por um funcionário válido; e
- RN11 – Em nenhuma hipótese deve-se alterar o valor de um produto.

Ilustre em uma aplicação: (i) a compra de um produto em que o cliente solicita verbalmente um desconto; (ii) a compra de um produto em que o cliente não solicita um desconto; e (iii) a atualização das vendas para uma compra realizada e para outra cancelada. Em todos os casos deve-se exibir em console o resumo da compra e das vendas.

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.

6) 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:

6.1) Apresente uma solução (diagrama de classes e codificação Java) de maneira a ter uma única base com uso de polimorfismo.

6.2) 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)

6.3) A partir do novo sistema, ilustre em uma aplicação o funcionamento dos serviços da base para dois clientes e dois funcionários.