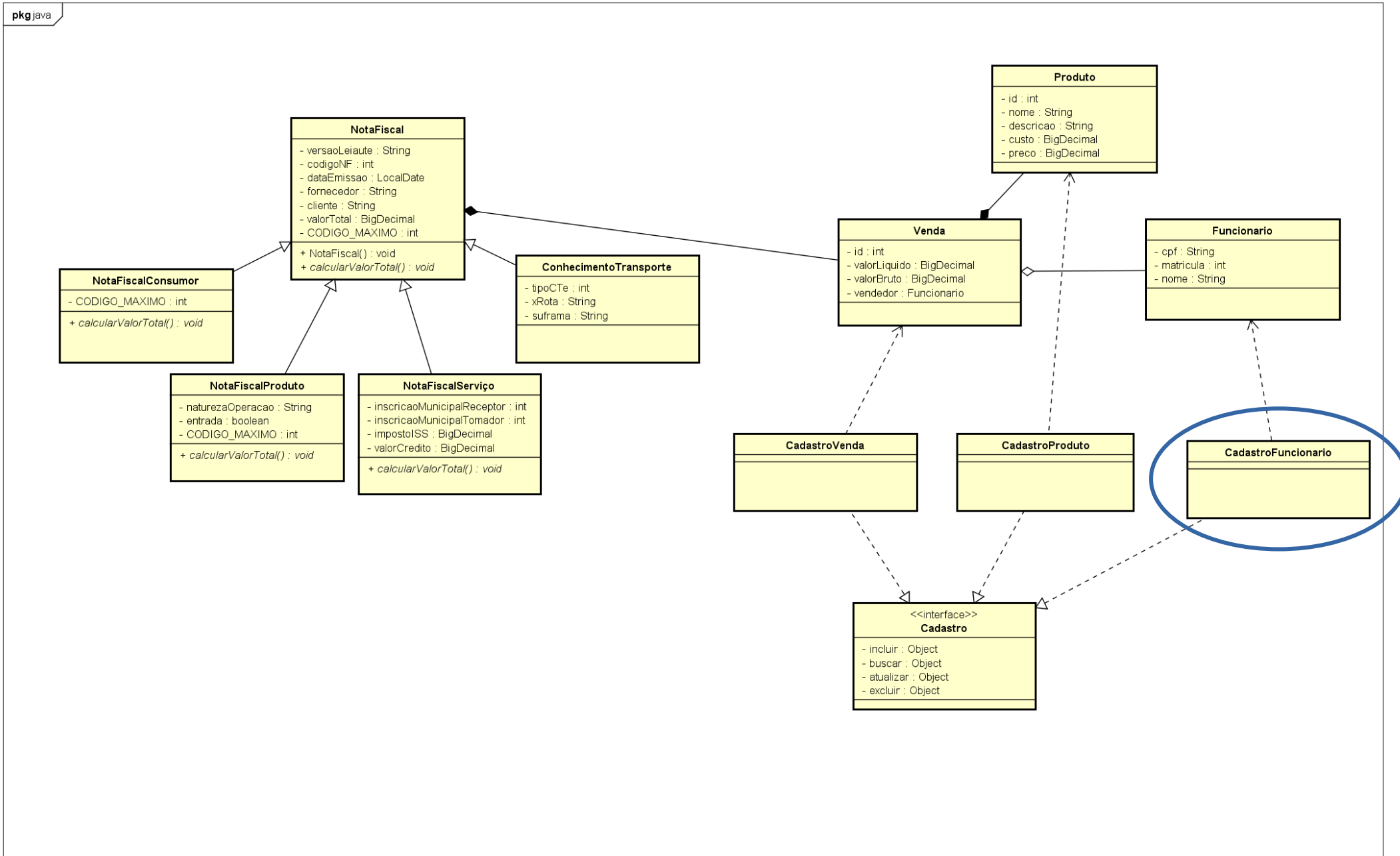


Coleções em Java

Capítulo XII

Um sistema de vendas hipotético



Um sistema de vendas hipotético

```
public void incluir(Funcionario func){
    int tamanho = funcionarios.length;
    tamanho++;
    Funcionario[] auxiliar = new Funcionario[tamanho];
    for (int i = 0; i < funcionarios.length; i++)
        auxiliar[i] = funcionarios[i];

    funcionarios = auxiliar;
    funcionarios[tamanho] = func;
}
```

```
public void atualizar(Funcionario func){
    String cpf = func.getCpf();
    for (int i = 0; i < funcionarios.length; i++) {
        if(funcionarios[i].getCpf().equals(cpf)){
            funcionarios[i] = func;
        }
    }
}
```

```
public Funcionario buscar(Funcionario func){
    String cpf = func.getCpf();
    for (Funcionario funcionario : funcionarios) {
        if(funcionario.getCpf().equals(cpf))
            return funcionario;
    }
    return null;
}
```

```
public void excluir(Funcionario func){
    String cpf = func.getCpf();
    int cont = 0;
    int tamanho = funcionarios.length;
    tamanho--;
    Funcionario[] auxiliar = new Funcionario[tamanho];
    for (int i = 0; i < funcionarios.length; i++)
        if(!funcionarios[i].getCpf().equals(cpf)){
            auxiliar[cont] = funcionarios[i];
            cont++;
        }
}
```

Esse cadastro segue uma boa prática?
Como ficaria a manutenção dele? E a performance?

Conhecendo o Collections Framework

- Visando nos auxiliar no trabalho com estruturas básicas de dados (Arrays) foi adicionado no pacote `java.util` um conjunto de classes e interfaces
 - Listas ligadas
 - Pilhas
 - Tabelas de espalhamento
 - Árvores
 - Entre outras

Conhecendo o ArrayList

```
public class CadastroList {  
    private ArrayList funcionarios;  
  
    public void incluir(Funcionario func){  
        funcionarios.add(func);  
    }  
  
    public Funcionario buscar(Funcionario func){  
        int index = funcionarios.indexOf(func);  
        return funcionarios.get(index);  
    }  
  
    public void atualizar(Funcionario func){  
        int index = funcionarios.indexOf(func);  
        funcionarios.set(index, func);  
    }  
  
    public void excluir(Funcionario func){  
        funcionarios.remove(func);  
    }  
}
```

*Ficou bem mais simples.
Mas por quê o erro?*

Conhecendo o ArrayList

```
public Funcionario buscar(Funcionario f,
    int index = funcionarios.indexOf(f)) {
    return funcionarios.get(index);
}
```

incompatible types: Object cannot be converted to Funcionario

(Alt-Enter, shows hints)

Para ser o mais genérico possível a API definiu que os parâmetros e retornos de métodos serão sempre “Object”

Em Java, todas as classes herdam de Object mesmo que de forma implícita

Então, como resolver esse problema?

Casting de referências

Precisamos avisar qual o tipo real daquele objeto, por esse motivos utilizamos do mesmo artifício que aplicamos nos tipos primitivos: o casting

```
public Funcionario buscar(Funcionario func){  
    int index = funcionarios.indexOf(func);  
    return (Funcionario) funcionarios.get(index);  
}
```

Aprimorando com a interface List





E se uma busca no Banco de Dados nos gerasse vários Funcionários, por exemplo, todos do Departamento de TI. Como armazenar?

```
public List<Funcionario> buscarPorDepartamento(String depto){  
    //consulta no BD  
    List<Funcionario> funcionarios = new ArrayList<>();  
    return funcionarios;  
}
```

O Generics nos ajuda a identificar possíveis erros de casting em tempo de compilação

Conhecendo a classe Collections

A classe Collections possui métodos estáticos que nos ajudam a trabalhar com o Framework Collection

- Collections.sort  ordena em ordem crescente
- Collections.binarySearch  busca binária
- Collections.max  busca maior elemento
- Collections.min  busca menor elemento

Conhecendo a classe Collections


```
public List<Funcionario> buscarPorDepartamento(String depto){  
    //consulta no BD  
    List<Funcionario> funcionarios = new ArrayList<>();  
    Collections.sort(funcionarios);  
    return funcionarios;  
}
```

Qual o motivo do erro?

Conhecendo a classe Collections

A definição de como a comparação é feita não estava clara

```
public class Funcionario implements Comparable<Funcionario>{  
  
    private String cpf;  
    private int matricula;  
    private String nome;  
}
```



Com o “contrato” assinado, sabemos como todo funcionário tem o método compareTo implementado

```
@Override  
public int compareTo(Funcionario func) {  
    return Integer.compare(this.getMatricula(), func.getMatricula());  
}
```

Temas dos Projetos

- Definir o grupo
- Qual tema do projeto ?

Exercícios

Faça crítica das entradas de dados

1. Implemente um programa para o controle de inventário de equipamentos da sua empresa. Neste primeiro momento serão levantados notebooks e smartphones
 - i. Notebooks: Marca, modelo, matrícula do responsável e número de série do aparelho
 - ii. Smartphone: Marca, modelo, IMEI, Centro de Custo e matrícula do responsável

2. O departamento de Relações com os Investidores da sua empresa gostaria de controlar os relatórios enviados e as comunicações com a imprensa que foram realizados. Para isso a TI foi contata e você o designado para desenvolver um sistema que automatize esse cadastro. As informações que deverão ser inseridas/exibidas são:
 - i. Relatórios: tema, funcionário responsável, funcionário revisor e data do envio
 - ii. Comunicações: tema, funcionário responsável, tipo de mídia e veículo (nome do jornal ou site)

Obs.: para os funcionários armazenar nome e matrícula