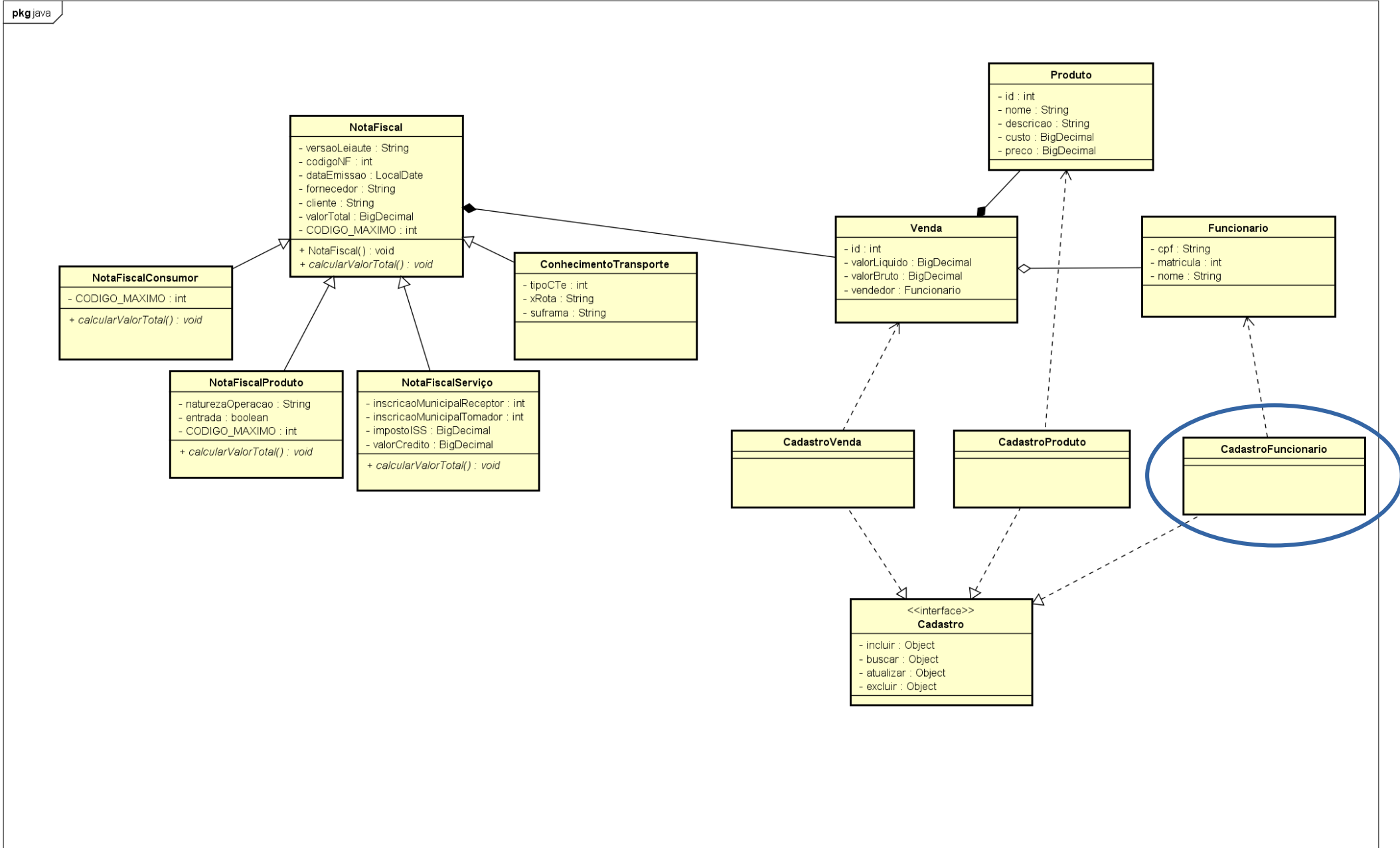


Coleções em Java

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?

Um sistema de vendas hipotético

Já vimos que trabalhar com arrays pode não ser tão trivial. Uma hora ou outra podemos nos deparar com algumas exceções, como por exemplo a *NullPointerException* ou *ArrayIndexOutOfBoundsException*.

Então, vamos simplificar as coisas!

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






Determinamos o “tipo” do nosso ArrayList através do *Generics*. Isso indica que nosso ArrayList só irá armazenar variáveis do tipo *Funcionario*.

```
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
- Collections.reverse  inverte a ordem dos elementos

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());  
}
```

Conhecendo a classe Collections

Para a comparação de valores numéricos utilizamos os métodos das classes *Wrappers*. Estas classes de modo geral, podem ser entendidas como **classes de tipos primitivos**, portanto, elas possuem métodos que podem nos ajudar.

Conhecendo a classe Collections

Iremos utilizar do método *compare*, dependendo do valor a ser comparado.

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



Objeto atual



Objeto a ser comparado

Conhecendo a classe Collections

E para comparar Strings (Ordem alfabética) ficaria assim

```
@Override  
public int compareTo(Funcionario func) {  
    return this.getNome().compareTo(func.getNome());  
}
```

Temas dos Projetos

- Definir o grupo (até duas pessoas)
- Qual tema do projeto ?
- Consistência de dados:
 - Arquivo .txt para que não faz a disciplina de Banco de Dados;
 - Banco SQL para que cursa a disciplina;
- Entrega dos temas até dia **01/11 (Semana da AV3)** por [e-mail](#) ou pessoalmente (aulas / atendimento)

13	17 de Outubro	Coleções
14	24 de Outubro	Interface Gráfica com Usuário - GUI
15	31 de Outubro	AV3
16	7 de Novembro	Auxílio ao Projeto Final (Debugging)
17	14 de Novembro	Auxílio ao Projeto Final
18	21 de Novembro	Auxílio ao Projeto Final
19	28 de Novembro	Apresentação dos Projetos Finais
20	5 de Dezembro	NP3 e Revisão da NP3

Exercícios

1. Implemente um Sistema de *Carrinho de Compras*, onde serão adicionados vários produtos. Cada produto possui um Nome e Preço.
 - I. Crie ao menos 5 produtos e armazene estes produtos em um ArrayList e mostre qual o maior e menor preço dentre os produtos;
 - II. Ordene o ArrayList em ordem crescente de preço;
 - III. Ordene o ArrayList em ordem alfabética;
 - IV. **Desafio:** Ordene o ArrayList em ordem decrescente de preço.

Exercícios

2. Desenvolva um programa que cadaster funcionários (id, nome, idade, sexo) usando JOptionPane e salve todos em um ArrayList.

Quando o usuário optar por sair, toda a informação contida no array deve ser salva em um arquivo .txt, permitindo que quando o programa for aberto novamente os dados sejam carregados e mostrados ao usuário.

Obs: O array sempre deve estar ordenado em relação à idade;

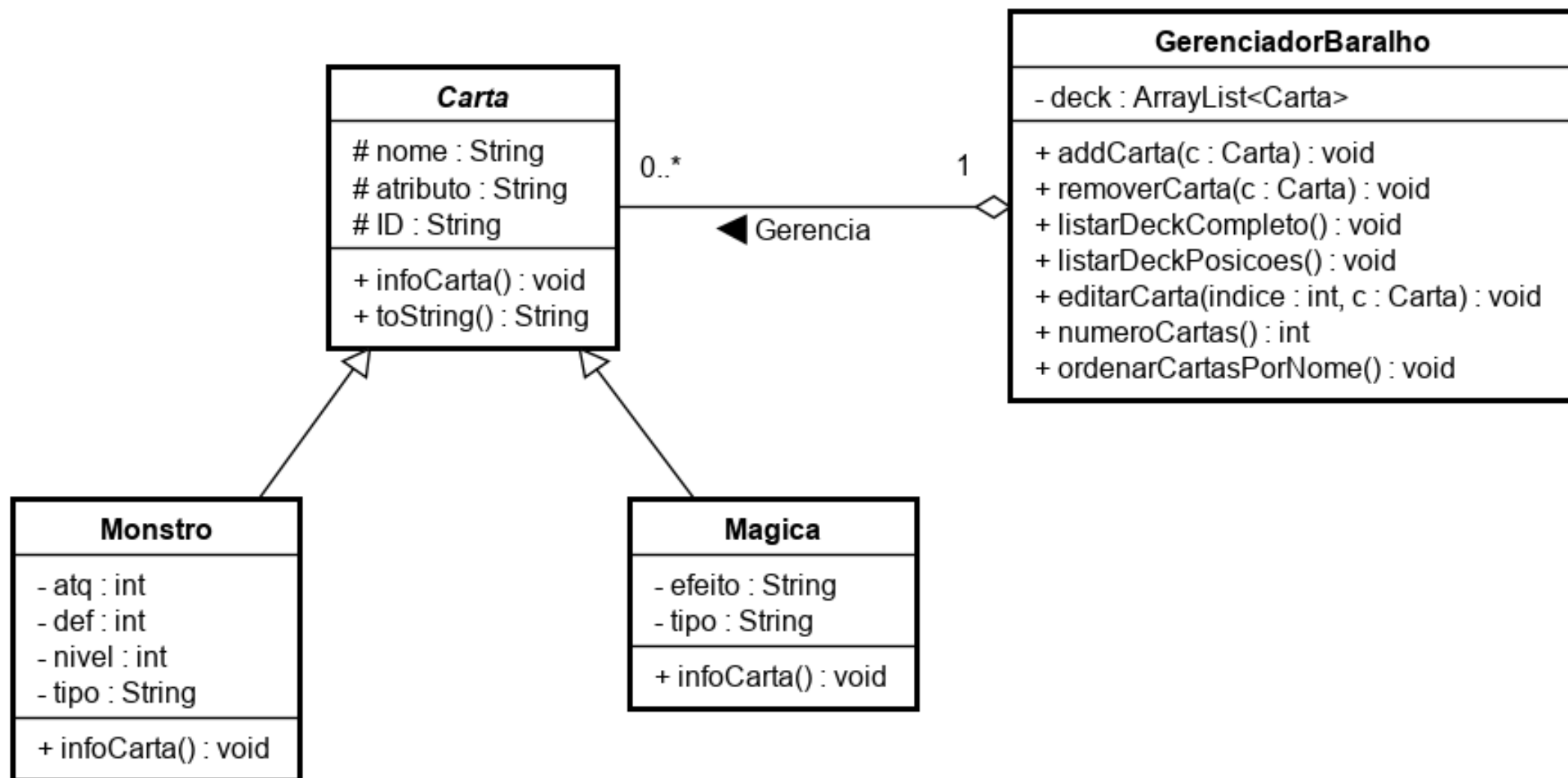
Exercício extra: **Desafio**

Yu-Gi-Oh! É um jogo de cartas baseado no anime e mangá de mesmo nome escrito e ilustrado por Kazuki Takahashi.



A partir do diagrama de classes dado no próximo slide, crie uma aplicação para gerenciar um *deck* de cartas. Sua aplicação deverá realizar um **CRUD completo** de cartas: Cadastrar, Listar, Editar e Excluir.

Exercício extra: **Desafio**



Exercício extra: **Desafio**

Importante !! As regras abaixo deverão ser seguidas.

- Seu projeto deverá seguir a arquitetura MVC;
- A classe Carta deverá ser *abstrata*;
- As classes filhas deverão ser *final*;
- Crie *Getters* e *Setters* quando necessário;
- O método *infoCarta()* deverá ser sobrescrito;
- Não esqueça de implementar a interface *Comparable* para ordenar as cartas;
- No método *listarDeckCompleto()* você deverá realizar o **casting** e chamar o método *infoCarta()* de acordo com a instância naquela posição do ArrayList;
- No método *listarDeckPosicoes()* você deverá imprimir a posição e o nome da carta. Use do método *get()*;
- O método *numeroCartas()* deverá retornar o número de cartas inseridas no ArrayList, ou seja, seu tamanho;
- **Não deverão** existir *Getters* e *Setters* na classe **GerenciadorBaralho**;
- **Você deverá chamar todos os métodos da classe GerenciadorBaralho**;
- A entrada de dados fica a seu critério!! Você poderá preencher os dados na *Main*, solicitar para o usuário, etc ...

Obrigado!