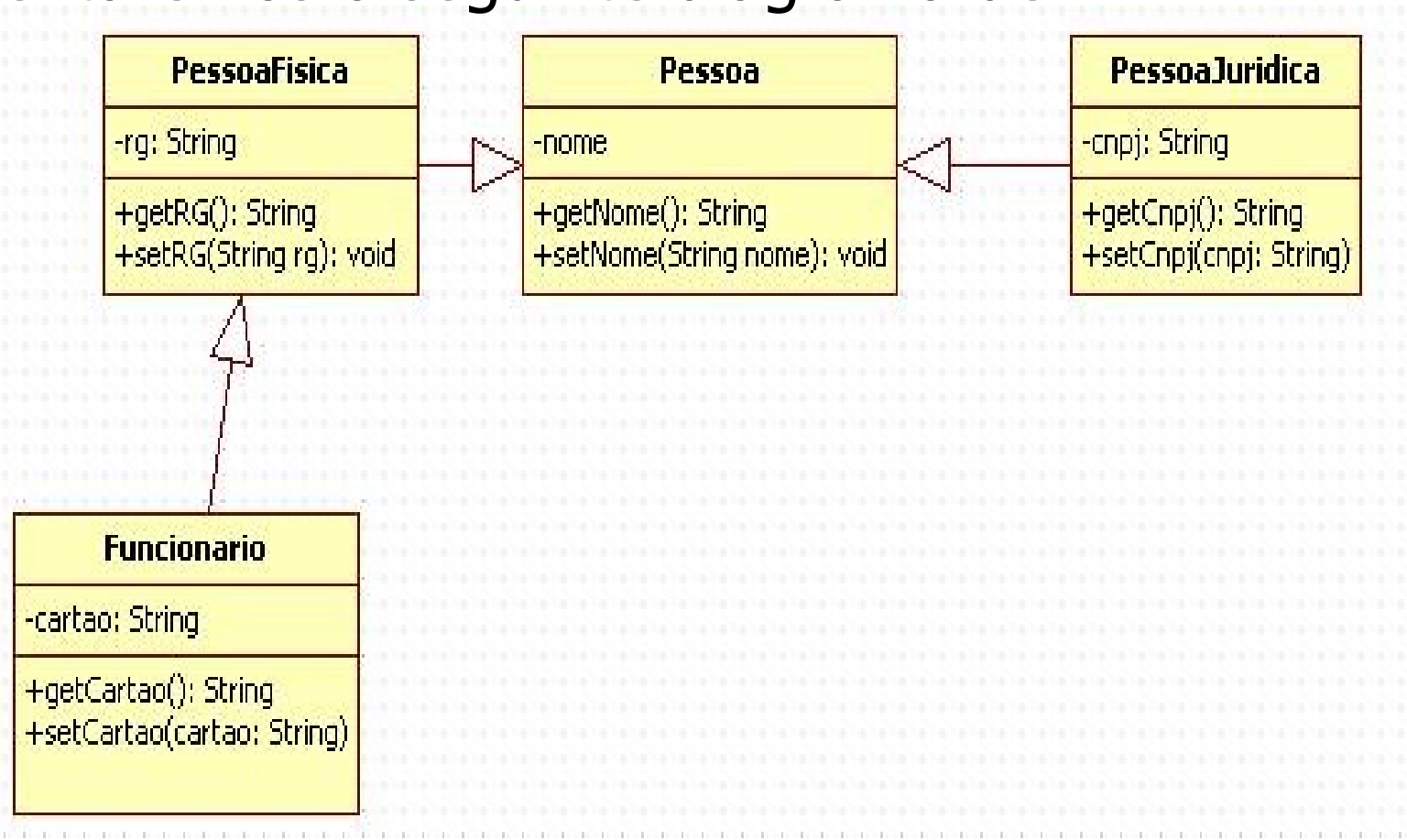
Polimorfismo e a classe Object

Instituto Federal do Espirito Santo – Ifes
Campus Colatina – Programação Orientada a Objetos I
Professor: Giovany Frossard Teixeira

Exemplo da última aula:

- Implementaremos o seguinte diagrama de classes:



Exemplo:

```
package introducao_oo;
public class Pessoa {
    private String nome;
    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

Exemplo:

```
package introducao_oo;
public class PessoaFisica extends Pessoa {
    private String rg;

    public String getRg() {
        return rg;
    }

    public void setRg(String rg) {
        this.rg = rg;
    }
}
```

```
package introducao_oo;
public class PessoaJuridica extends Pessoa {
    private String cnpj;
    public String getCnpj() {
        return cnpj;
    }

    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }
}
```

Exemplo:

```
package introducao_oo;

public class Funcionario extends PessoaFisica {

    private String cartao;

    public String getCartao() {

        return cartao;

    }


    public void setCartao(String cartao) {

        this.cartao = cartao;

    }

}
```

Polimorfismo - Conceito

- **Polimorfismo (várias formas)**: é a capacidade de assumir formas, em outras palavras, é um código que pode ser aplicado à várias classes de objetos. De maneira prática isto quer dizer que a operação em questão mantém seu comportamento transparente para quaisquer tipos de argumentos; isto é, o mesmo método é acionado por objetos de classes distintas e eles poderão reagir de maneiras diferentes.
- Uma definição alternativa mas dependente do conceito de herança é a seguinte: Polimorfismo é a capacidade de utilizar métodos definidos em um ancestral, implementados em seus descendentes e executados de acordo com a classe do objeto que aciona o método.

Polimorfismo – Ex:

- Será utilizado o mesmo exemplo utilizado para ilustrar herança, acrescentando o método imprimirNomeClasse em todos os níveis da hierarquia.
- O método deverá ser implementado na seguinte forma:

```
public void imprimirNomeClasse(){
        System.out.println("Classe <nome da classe>");
}
```

Na classe Pessoa será implementado:

```
public void imprimirNomeClasse(){
        System.out.println("Classe Pessoa");
}
```

Polimorfismo – Ex:

```
package introducao_oo;

import java.io.*;

public class UsaPessoaPolimorfa {


    public static void main(String[] args) {

        Pessoa pessoa = null;

        BufferedReader dado;

        String s;

        try {
```

```
System.out.println("Digite que
 objeto deseja criar:");

System.out.println("1 - Pessoa");

System.out.println("2 - Pessoa
 Física");

System.out.println("3 - Pessoa
Juridica");

System.out.println("4 -
 Funcionário");


dado = new BufferedReader(new
 InputStreamReader(System.in));

s = dado.readLine();
```

Polimorfismo – Ex:

```
switch (Integer.parseInt(s))
{
    case 1: pessoa = new Pessoa();
     break;
    case 2: pessoa = new
     PessoaFisica(); break;
    case 3: pessoa = new
     PessoaJuridica(); break;
    case 4: pessoa = new
     Funcionario(); break;
    default: System.out.println("tipo
     desconhecido");
}
```

```
    pessoa.imprimirNomeClasse();
}
catch (IOException erro) {
        System.out.println("Erro na
         entrada de dados");
 }
 catch (NumberFormatException erro) {
        System.out.println("Digite apenas
        números inteiros");
 }
}
}
```

Classe Object

- É a classe ancestral de todas as classes. Quando uma classe não especifica de quem ela herda, implicitamente ela herda de Object.

- **Object** possui um conjunto de métodos interessantes que podem ser implementados nas classes descendentes. No próximo slide veremos alguns deles.

Métodos da classe Object

- **toString()**: Esse método indica como transformar um objeto de uma classe em uma String. Vejamos um exemplo para as classes Pessoa e PessoaFisica:

Pessoa:

```
public String toString() {

      return ("Nome: " + this.nome);

}
```

Em PessoaFisica poderíamos ter:

```
public String toString() {

      return (super.toString() + " RG: " +
    this.rg);

}
```

Obs. Note o uso da palavra super para chamar o método do ancestral.

Métodos da classe Object

- **equals(Object obj):** determina se dois objetos são iguais.
  - Nesse contexto vem a pergunta por que não utilizar o operador de igualdade == ?
    - Essa pergunta só pode ser respondida através dos "famigerados" ponteiros.
    - Quando comparamos duas variáveis do tipo primitivo **int** estamos comparando seus valores, logo o operador == pode ser utilizado.
    - Quando comparamos dois objetos, na realidade estamos comparando ponteiros para espaços de memória. Isso significa que dois objetos podem ter os mesmos valores em seus atributos e não serem iguais, pois podem apontar para locais diferentes.

Métodos da classe Object

- equals: vejamos como o método equals poderia ser implementado para a classe Pessoa.

```
public boolean equals(Object obj) throws ClassCastException {
        Pessoa pessoa = (Pessoa)obj;
        return (this.nome.equals(pessoa.nome));
}
```

Obs1. Se obj não for da classe Pessoa será gerada uma exceção da classe ClassCastExcecption. Tal exceção não é obrigatoriamente tratada pelo método que chamou o equals da classe Pessoa.

Métodos da classe Object

- getClass(): retorna a classe do objeto, muito utilizado quando trabalha-se na criação de ferramentas geradoras de código ou frameworks.

Métodos da classe Object

- Existem diversos outros métodos na classe Object que podem ser utilizados e/ou re-implementados.

- É importante conhecermos a hierarquia de uma classe para evitarmos replicar códigos de forma desnecessária.

Dúvidas?