

Avançando em Java com **Polimorfismo**

Prof.: Hugo Barros



Tópicos da Aula

- Polimorfismo:
 - Exemplos de polimorfismo
 - Vantagens do polimorfismo
 - Tipos de polimorfismo
- Métodos Abstratos
- Classes Abstratas
- Vinculação Dinâmica



Tópicos da Aula

- Hierarquia Empregado
 - Classe Empregado
 - Classe EmpregadoHorista
 - Classe EmpregadoPorComissao
 - Classe EmpregadoFixoMaisComissao
- Processamento Polimórfico
 - Operador instanceof
 - A classe Class



Tópicos da Aula

- Exemplo de Polimorfismo
- Casting
 - Cast
 - Upcast
 - Downcast
- Métodos Final
- Classes Final

Polimorfismo

- Permite que um método produza resultados diferentes, dependendo do objeto ao qual é aplicado.
- A mesma invocação pode produzir "muitas formas" de resultados.



Polimorfismo

- Um nome para várias formas, ou ainda, um nome e várias implementações(vários métodos com o mesmo nome).
- Capacidade de uma referência de variável mudar seu comportamento de acordo com o objeto a que ela está conectada.
 - Isto permite que objetos diferentes, de subclasses distintas, sejam vistos como objetos de uma mesma superclasse – o método apropriado a ser executado será escolhido automaticamente, baseado na subclasse a qual o objeto pertence.



Exemplos de Polimorfismo

- Sobrecarga de um método:
 - Mesmo nome, parâmetros diferentes:
 - `setData(int dia, int mes, int ano);`
 - `setData(int dia, int mes, int ano, int hora, int minuto, int segundo);`



Exemplos de Polimorfismo

- Sobrescrita de um método
 - Uma subclasse redefine um comportamento de sua superclasse:
 - `UmaPessoa.toString();`
 - `UmEmpregado.toString();`
 - `UmGerente.toString();`



Exemplos de Polimorfismo

- Vinculação Dinâmica
 - Dynamic Biding
 - O método a ser chamado só pode ser determinado em tempo de execução.



Vantagens do Polimorfismo

- Simplicidade:
 - Quando necessário escrever um código que manipula uma família de tipos, o código pode ignorar detalhes específicos de cada tipo;
 - Mesmo que o código aparente estar manipulando um objeto do tipo da classe-mãe. O objeto na verdade pode ser do tipo da classe-mãe, mas também do tipo das classes filhas;
 - Isso torna o código mais fácil de se escrever e de se entender.



Vantagens do Polimorfismo

- Escalabilidade:
 - Num momento futuro, outras classes poderão ser adicionadas à família de tipos, sendo que os seus objetos executarão seu código específico.



Tipos de Polimorfismo

- Sobrecarga (overloading)
 - Ocorre na mesma classe ou entre classe-mãe e classe-filha;
 - O métodos diferem pela assinatura (ordem e número de parâmetros).



Tipos de Polimorfismo

- Sobreposição (overriding):
 - Métodos na classe filha que redefinem ou sobrepõem o comportamento do mesmo método da classe mãe.
 - Apresenta mesma lista de parâmetros.
 - Existe também a sobreposição (ou implementação) de métodos concretos (classe-filha) em métodos abstratos (classe mãe ou interface).



Métodos Abstratos

- Métodos declarados sem implementação:
 - `abstract void mover(double X, double Y);`



Classes Abstratas

- São classes em que um ou mais métodos são abstratos:

```
public abstract class Animal{  
    private String nome;  
    public Animal(String nome) {  
        this.nome = nome;  
    }  
    public String getNome() {  
        return this.nome;  
    }  
    public abstract void emitirSom();  
}
```



Classes Abstratas

- Uma classe abstrata declara atributos e comportamentos comuns das várias classes em uma hierarquia de classes.
- As subclasses devem sobrescrever os métodos abstratos para se tornarem concretas.
- As classes abstratas não podem ser instanciadas.



Classes Abstratas

```
public class Vaca extends Animal{  
    public Vaca(String nome){  
        super(nome);  
    }  
  
    public void emitirSom(){  
        System.out.println("MUUUUUUUUUUU");  
    }  
}
```



Classes Abstratas

```
public class Cachorro extends Animal{  
    public Cachorro(String nome){  
        super(nome);  
    }  
    public void emitirSom(){  
        System.out.println("AUAU");  
    }  
    public void rosnar(){  
        System.out.println("GRRRRR");  
    }  
}
```

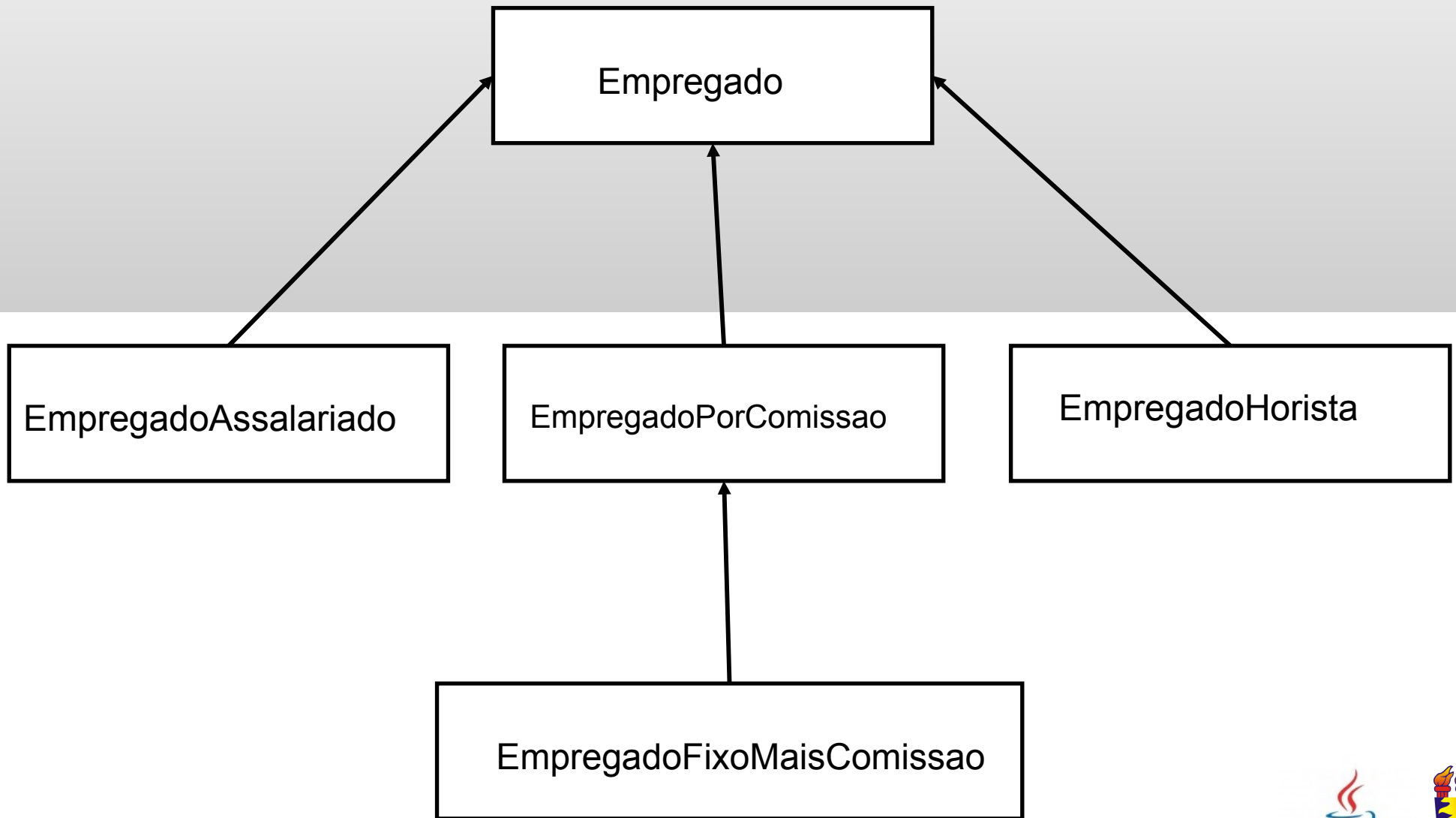


Vinculação Dinâmica

```
public class AnimalTest{  
    public static void main(String args[]){  
        Animal ref;  
        Vaca umaVaca = new Vaca("Mimosa");  
        Cachorro umCachorro = new Cachorro("Rex");  
        //Referenciada cada um como animal  
        ref = umaVaca;  
        ref.emitirSom();//Polimorfismo: Dynamic Binding  
        ref = umCachorro;  
        ref.emitirSom();//Polimorfismo: Dynamic Binding  
    }  
}
```



Hierarquia Empregado



Classe Empregado

- Superclasse abstract Empregado
 - Método ganhos é declarado abstract
 - Nemhna implementação pode ser dada ao método ganhos na classe abstract Empregado.



Classe Empregado

```
public abstract class Empregado{  
    private String nome;  
    private String rg;  
  
    public Empregado(String nome, String rg){  
        this.nome = nome;  
        this.rg = rg;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```

...



Classe Empregado

```
...  
public String getNome() {  
    return this.nome;  
}  
  
public void setRg(String rg) {  
    this.rg = rg;  
}  
  
public String getRg() {  
    return this.rg;  
}  
...
```



Classe Empregado

```
/*  
 * Método abstrato a ser sobrescrito pelas  
   subclasses  
 */  
    public abstract double ganhos();  
//Finaliza a classe empregado  
}
```



Classe EmpregadoHorista

```
public class EmpregadoHorista extends Empregado{  
    private double valorHora;  
    private double horas;  
  
    public EmpregadoHorista(String nome, String rg, double  
        valorHora, double horas){  
        super(nome, rg);  
        this.valorHora = valorHora;  
        this.horas = horas;  
    }  
  
    //Get's e Set's  
}
```



Classe EmpregadoHorista

```
//Implementando o método ganhos
```

```
public double ganhos () {  
    if (getHoras () <= 40 ) {  
        return getValorHora () * getHoras ();  
    }else{  
        return (getHoras () -40) * getValorHora () *1.5;  
    }  
}
```



Classe EmpregadoAssalariado

```
public class EmpregadoAssalariado extends Empregado{  
    private double salarioSemanal;  
  
    public EmpregadoAssalariado(String nome, String rg, double  
        salarioSemanal){  
        super(nome, rg);  
        this.salarioSemanal = salarioSemanal;  
    }  
    //Get's e Set's  
    public double ganhos(){  
        return getSalarioSemanal();  
    }  
}
```



Classe EmpregadoPorComissao

```
public class EmpregadoPorComissao extends Empregado{  
    private double vendasBrutas;  
    private double taxaComissao;  
    public EmpregadoPorComissao(String nome, String rg, double  
        vendasBrutas, double taxaComissao){  
        super(nome, rg);  
        this.vendasBrutas = vendasBrutas;  
        this.taxaComissao = taxaComissao;  
    }  
    //Get's e Set's  
    public double ganhos(){  
        return getTaxaComissao() * getVendasBrutas();  
    }  
}
```



Classe EmpregadoFixoMaisComissao

```
public class EmpregadoFixoMaisComissao extends EmpregadoPorComissao{  
    private double salarioBase;  
  
    public EmpregadoFixoMaisComissao(String nome, String rg, double  
        vendasBrutas, double taxaComissao, double salarioBase){  
        super(nome, rg, vendasBrutas, taxaComissao);  
        this.salarioBase = salarioBase;  
    }  
    //Get's e Set's  
    public double ganhos() {  
        return getSalarioBase() + super.ganhos();  
    }  
}
```



Processamento Polimórfico

Vinculação Dinâmica:

- Também conhecida como vinculação Tardia;
- Chamadas de métodos sobrescritos, são resolvidos em tempo de execução conforme objeto referenciado.



Operador instanceof

- Determina se um objeto é uma instância de determinado tipo



Casting

Cast: Converter um tipo de dados para outro

```
int i = 20;
```

```
double x = 2.5;
```

```
x = (double) i;
```



Conversão de Tipos

Supondo a variável x	Converter em	A variável y recebe o valor convertido
Int x = 30	float	float y = (float) x
Int x = 30	double	double y = (double) x
float x = 20.5	int	int y = (int) x
String x = "20"	int	int y = Integer.parseInt(x)
String x = "40.84"	float	float y = Float.parseFloat(x)
String x = "40.84"	double	double y = Double.parseDouble(x)
String x = "Java"	Vetor de bytes	byte y[] = x.getBytes()
Int x = 30	String	String y = String.valueOf(x)
float x = 20.5	String	String y = String.valueOf(x)
double x = 452.43	String	String y = String.valueOf(x)
Byte x[]	String	String y = new String(x)



Casting

Upcast: converter sem perda de informação

```
int i = 20;
```

```
double x = 2.5;
```

```
x = (double) i;
```



Casting

Downcast: converter com perda de informação

```
int i = 20;  
double x = 2.5;
```

```
i = (int) x;
```

Java Warning: Possible loss of precision



Casting

Cast: Conversão dentro de uma hierarquia de classes.



Casting

- Upcast: Subir na hierarquia
 - EmpregadoPorComissao → Empregado
 - EmpregadoFixoMaisComissao → EmpregadoPorComissao

- Downcast: Descer na hierarquia
 - EmpregadoPorComissao → EmpregadoFixoMaisComissao



Casting

- Para evitar erros em downcasting é necessário a utilização do operador `instanceof`, para assegurar que o objeto é de fato um objeto de um tipo de subclasse apropriado.

```
if(umAnimal instanceof Cachorro) {  
    umCachorro = (Cachorro) umAnimal;  
}
```



Métodos Final

- Não podem ser sobrescritos em uma subclasse
- São resolvidos em tempo de compilação, isto é conhecido como vinculação estática.



Classes Final

- Não podem ser estendidas por uma subclasse
- Todos os métodos em uma classe final são implicitamente final



Atividade

Usando a sua imaginação crie uma hierarquia de classes que deve conter no mínimo 4 classes, sendo obrigatoriamente uma a superclasse e outra uma classe que tenha um nível 2 de herança, depois implemente-as e teste-as usando uma classe que contenha um método Main.

