

Polimorfismo com herança

Padrões de Projeto Orientado a
Objetos

Profa. Danielle Martin

Prof. Pedro Toledo

Universidade de Mogi das Cruzes

Polimorfismo

Polimorfismo é a característica da orientação a objetos que desacopla a **referência** de um objeto da **implementação** do mesmo, permitindo que um **tipo comum** (abstrato ou genérico) possa utilizar um objeto de qualquer classe polimórfica sem diferenciação.

Exemplo:

```
Animal objeto = new Cachorro();
```

Variavel de referência
do tipo Animal

Objeto instância do tipo
Cachorro

Polimorfismo

É possível implementar o polimorfismo usando:

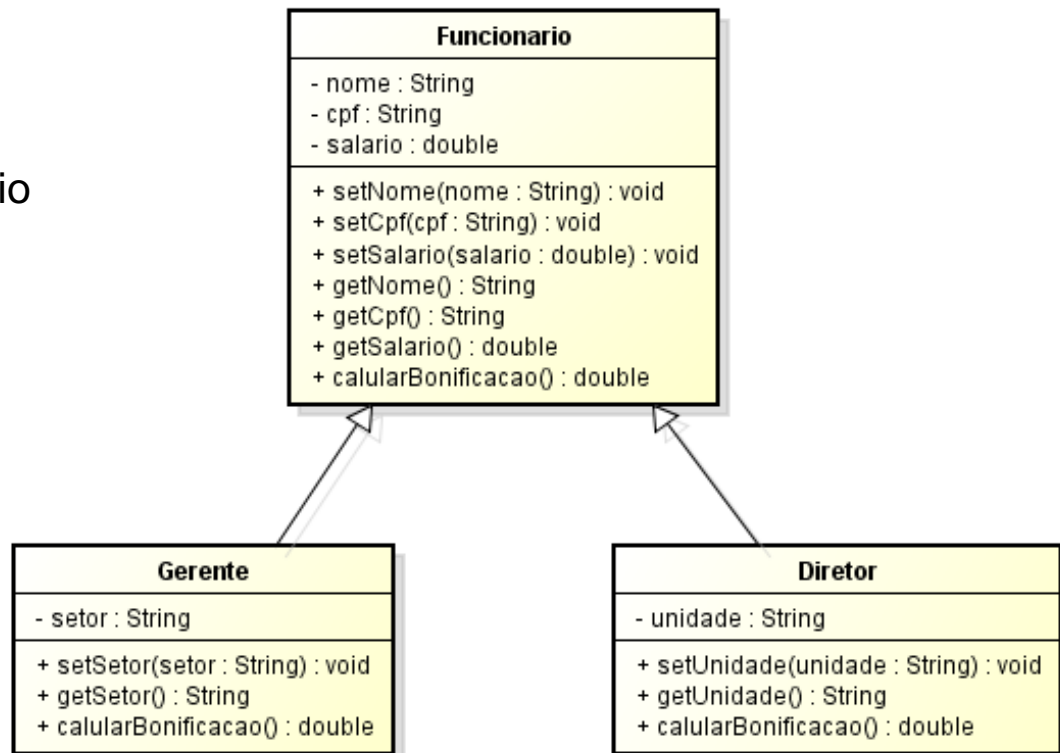
- Herança
- Classes abstratas
- Interfaces

I. HERANÇA

Herança

- Herança: Relacionamento hierárquico onde subclasses, ou classes filhas, herdam os atributos e os comportamentos de sua superclasse, podendo estender este comportamento para atender a sua necessidade específica.

- Relacionamento: IS-A
 - Gerente **É UM** Funcionário
 - Diretor **É UM** Funcionário



Características da herança

- A subclasse ou classe filha deve referenciar a superclasse usando a palavra reservada **extends**.
- Métodos e atributos declarados como **private** na superclasse não serão visíveis nem herdados pela subclasse.
- Métodos e atributos declarados como **public** ou **protected** na superclasse serão herdados pelas subclasses.
- Métodos declarados como **final** na superclasse não podem ser sobrescritos.

Características da herança

- Métodos da superclasse podem ser **sobrescritos** na subclasse com a mesma assinatura. A implementação da classe específica (subclasse) irá prevalecer sobre a implementação genérica (superclasse).
- Essa característica é chamada de **late binding** (acoplamento dinâmico), pois o método a ser executado é definido em tempo de execução.

```
public class Veiculo {  
    public void andar() {  
        System.out.println("Veiculo andando");  
    }  
}  
  
public class Carro extends Veiculo {  
    public void andar() {  
        System.out.println("Carro andando");  
    }  
}
```

```
Veiculo objeto = new Carro();  
  
objeto.andar();    //imprime "Carro andando"
```

Características da herança

- É possível referenciar um método sobrescrito da superclasse usando a palavra **super**.

```
public class Veiculo {  
    public void andar() {  
        System.out.println("Ligar motor");  
        System.out.println("Dar partida");  
    }  
}  
  
public class Carro extends Veiculo {  
    public void andar() {  
        super();  
        System.out.println("Carro andando");  
    }  
}
```

```
Veiculo objeto = new Carro();  
  
objeto.andar();    //imprime "Ligar motor"  
                  "Dar partida"  
                  "Carro andando"
```


Características da herança

- Construtores da subclasse podem referenciar construtores da superclasse também usando **super()**. Se o construtor da superclasse definir parâmetros de entrada, os mesmos devem ser passados.

```
public class Funcionario {  
  
    private String nome;  
    private String cpf;  
    private double salario;  
  
    public Funcionario(String nome, String cpf, double salario) {  
        this.nome = nome;  
        this.cpf = cpf;  
        this.salario = salario;  
    }  
}
```

```
public class Diretor extends Funcionario {  
  
    private String unidade;  
  
    public Diretor(String nome, String cpf, double salario, String unidade) {  
        super(nome, cpf, salario);  
        this.unidade = unidade;  
    }  
}
```

Polimorfismo com herança

```
Funcionario funcionario = new Funcionario();  
Funcionario gerente = new Gerente();  
Funcionario diretor = new Diretor();
```

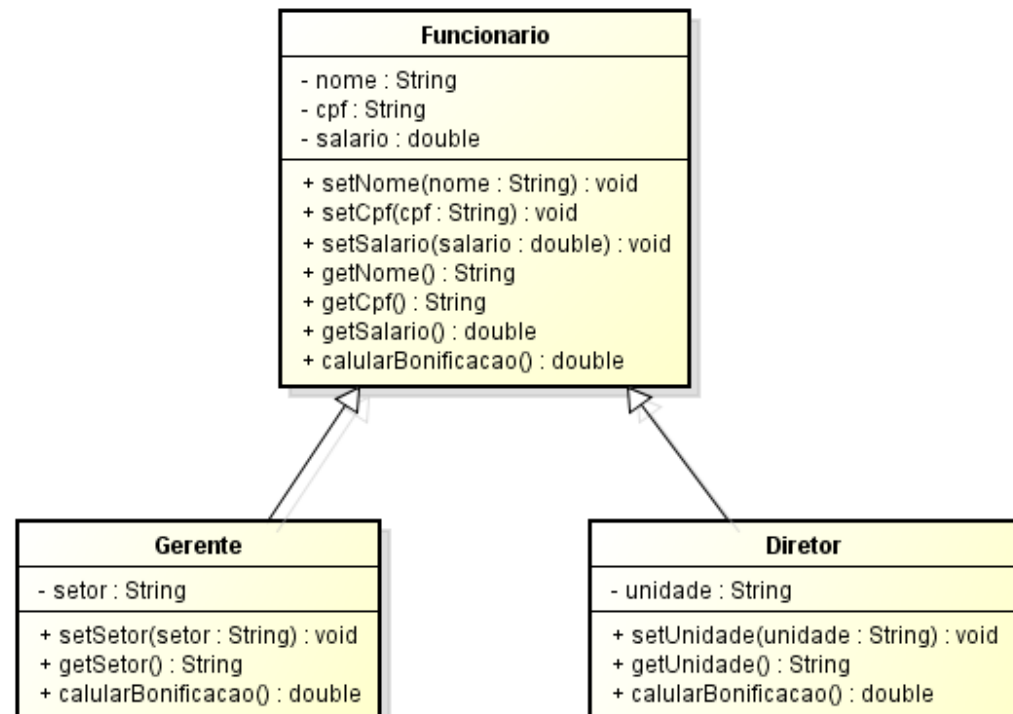
} objetos de classes diferentes

```
//Array de Funcionario  
Funcionario[] arrayFuncionarios = new Funcionario[3];
```

```
arrayFuncionarios[0] = funcionario;  
arrayFuncionarios[1] = gerente;  
arrayFuncionarios[2] = diretor;
```

```
for (Funcionario func : arrayFuncionarios)  
{  
    //aciona o método calcularBonificacao  
    //para todos os objetos do array  
    func.calcularBonificacao();  
}
```

Todos os objetos são tratados pelo tipo genérico Funcionário, que permite o cálculo da bonificação. Os resultados são diferentes para cada tipo de objeto, pois as subclasses sobreescrevem o método calcularBonificacao.



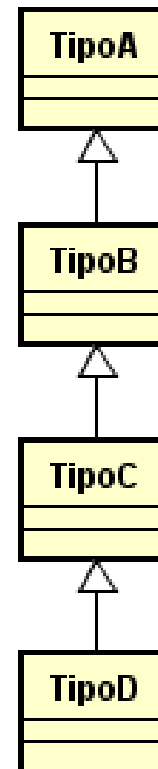
Polimorfismo com herança

- Quais alternativas representam atribuições válidas de objetos na variável de referência `obj`?

```
TipoA a = new TipoA();  
TipoB b = new TipoB();  
TipoC c = new TipoC();  
TipoD d = new TipoD();
```

```
TipoB obj;
```

- A) `obj = a;`
- B) `obj = b;`
- C) `obj = c;`
- D) `obj = d;`



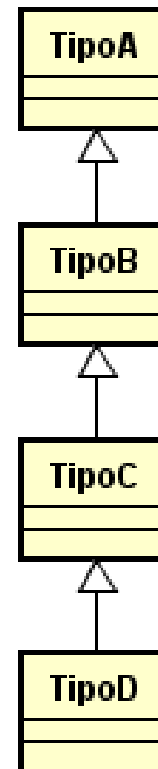
Polimorfismo com herança

- Quais alternativas representam atribuições válidas de objetos na variável de referência obj?

```
TipoA a = new TipoA();  
TipoB b = new TipoB();  
TipoC c = new TipoC();  
TipoD d = new TipoD();
```

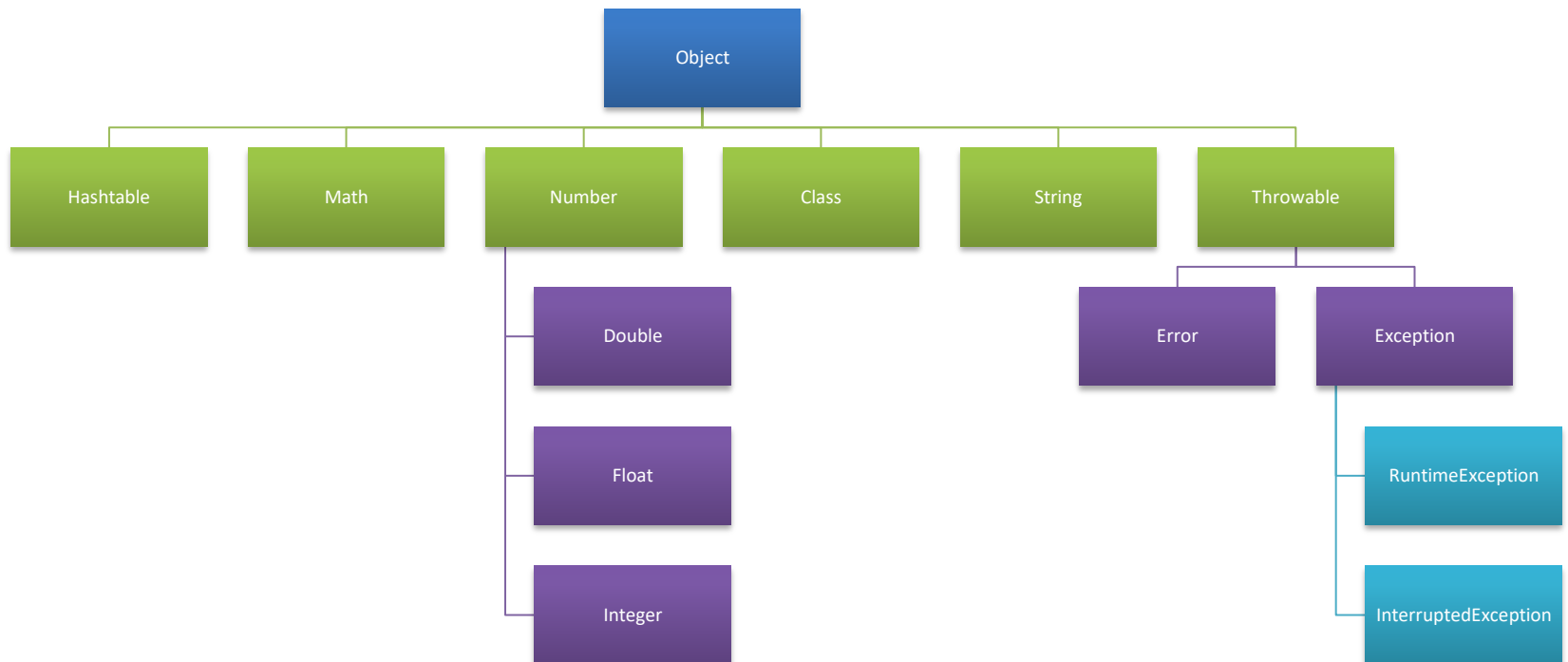
```
TipoB obj;
```

- A) obj = a;
- B) obj = b;**
- C) obj = c;**
- D) obj = d;**



Hierarquia de classes

- É uma estrutura onde várias classes estão conectadas através de relacionamentos de herança em diversos níveis.
- No Java, todas as classes são subclasses do supertipo Object.



Classe Object

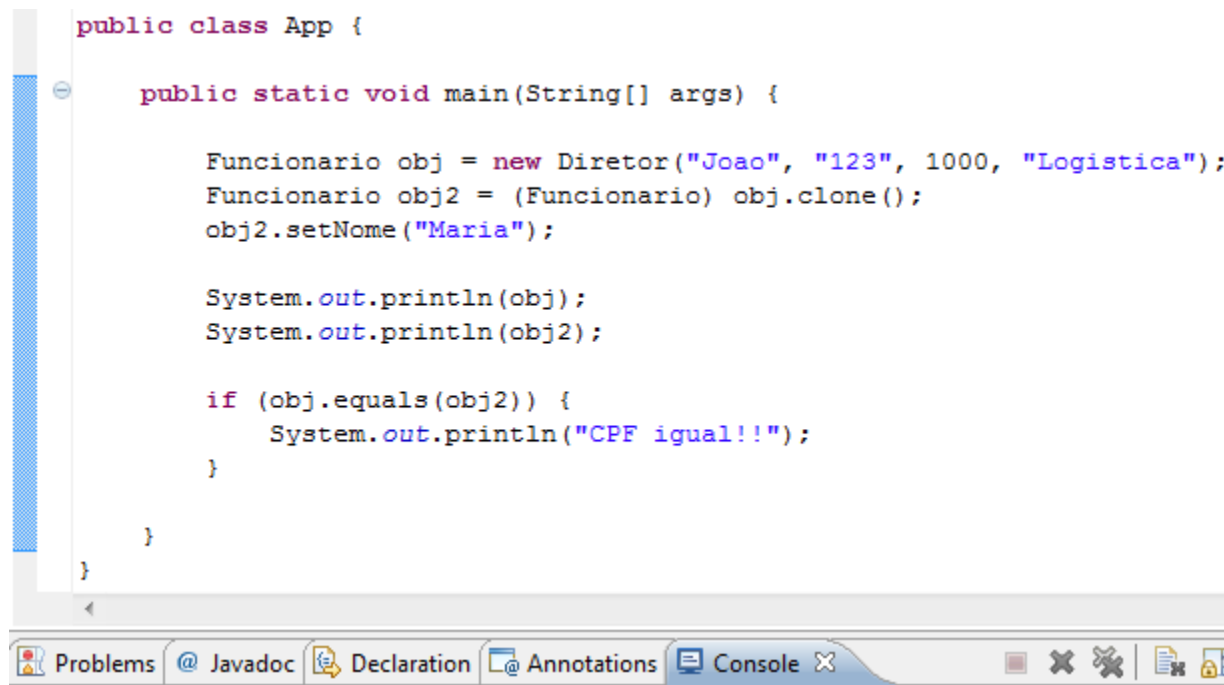
- Os métodos da classe Object, tais como toString(), clone() e equals(), são automaticamente herdados por qualquer classe criada, e podem inclusive ser sobrescritos por ela.

```
public class Funcionario {  
  
    private String nome;  
    private String cpf;  
    private double salario;  
  
    public Funcionario(String nome, String cpf, double salario) {  
        this.nome = nome;  
        this.cpf = cpf;  
        this.salario = salario;  
    }  
  
    public String toString() {  
        return "Funcionario [nome=" + nome + ", cpf=" + cpf + ", salario=" + salario + "];"  
    }  
  
    public boolean equals(Object obj) {  
        Funcionario other = (Funcionario) obj;  
        if (other == null || cpf == null || other.cpf == null || !cpf.equals(other.cpf))  
            return false;  
        return true;  
    }  
  
    public Object clone() {  
        return new Funcionario(nome, cpf, salario);  
    }  
}
```

Métodos da classe Object

- toString() – imprime uma versão String do objeto
- clone() - cria uma cópia do objeto atual em outro espaço de memória
- equals() – compara dois objetos e retorna true se forem iguais

```
public class App {  
  
    public static void main(String[] args) {  
  
        Funcionario obj = new Diretor("Joao", "123", 1000, "Logistica");  
        Funcionario obj2 = (Funcionario) obj.clone();  
        obj2.setNome("Maria");  
  
        System.out.println(obj);  
        System.out.println(obj2);  
  
        if (obj.equals(obj2)) {  
            System.out.println("CPF igual!!");  
        }  
  
    }  
}
```

The image shows a screenshot of an IDE window with a Java code editor. The code defines a class 'App' with a 'main' method. Inside 'main', it creates a 'Funcionario' object 'obj' of type 'Diretor' with parameters 'Joao', '123', 1000, and 'Logistica'. It then clones 'obj' into 'obj2' and changes 'obj2' name to 'Maria'. It prints both objects and checks if they are equal using 'equals()'. The IDE's bottom bar shows tabs for 'Problems', 'Javadoc', 'Declaration', 'Annotations', and 'Console'.

Métodos da classe Object

- toString() – imprime uma versão String do objeto
- clone() - cria uma cópia do objeto atual em outro espaço de memória
- equals() – compara dois objetos e retorna true se forem iguais

```
public class App {  
    public static void main(String[] args) {  
        Funcionario obj = new Diretor("Joao", "123", 1000, "Logistica");  
        Funcionario obj2 = (Funcionario) obj.clone();  
        obj2.setNome("Maria");  
  
        System.out.println(obj);  
        System.out.println(obj2);  
  
        if (obj.equals(obj2)) {  
            System.out.println("CPF igual!!");  
        }  
    }  
}
```

Problems | Javadoc | Declaration | Annotations | Console

<terminated> App [Java Application] C:\Program Files (x86)\IBM\SDP\jdk\bin\javaw.exe (10/02/2016 19:08:31)
Funcionario [nome=Joao, cpf=123, salario=1000.0]
Funcionario [nome=Maria, cpf=123, salario=1000.0]
CPF igual!!