

Aula 11

Polimorfismo

Programação III

Prof. Augusto César Oliveira

augusto.oliveira@unicap.br

Na aula passada...

- Compreender o conceito de herança e sua aplicação na programação orientada a objetos;
- Compreender o significado e o uso da palavra-chave "final" em relação a classes, métodos e atributos;
- Entender os modificadores de acesso (public, private, protected) e seu papel na encapsulação;

Na aula passada...

- Compreender o conceito de classes abstratas e seu propósito na modelagem de classes;
- Compreender o conceito de polimorfismo e sua importância na programação orientada a objetos.

O objetivo da aula de hoje...

- Amadurecer o conceito de polimorfismo na programação orientada a objetos;
- Entender como o polimorfismo permite que objetos de diferentes classes sejam tratados de maneira uniforme;
- Compreender o conceito de interfaces e seu papel na programação orientada a objetos;
- Aprender a criar e implementar interfaces em Java.


1. Polimorfismo

Polimorfismo

Um "Programador" é um "Funcionário"?

- Se a herança estabelece uma relação de "**É UM**", podemos dizer que um "Programador" é um "Funcionário".

```
public class ClassePrincipal {  
    public static void main(String[] args) {  
        Programador programador = new Programador();  
        programador.setNome("José Silva");  
        programador.setSalario(5000);  
        programador.setLinguagem("Java");  
        Funcionario funcionario = programador;  
    }  
}
```



Um objeto "**Programador**" pode ser atribuído à uma variável "**Funcionário**"

O que é polimorfismo?

- É a capacidade de uma classe **se comportar de diferentes formas**.
- Um objeto "**Programador**" pode assumir a forma de "**Funcionário**".
- **Herança** permite esse tipo de **comportamento polimórfico**.

E qual a utilidade disso?

```
public class ClassePrincipal {  
    public static void main(String[] args) {  
        Programador programador = new Programador();  
        programador.setNome("José da Silva");  
        Designer designer = new Designer();  
        designer.setNome("Maria Laura");  
  
        Funcionario[] funcionarios = new Funcionario[10];  
        funcionarios[0] = programador;  
        funcionarios[1] = designer;  
    }  
}
```

Um **array** que armazena qualquer tipo de "Funcionario"



Uma outra utilidade...

```
public class ClassePrincipal {  
    public static void main(String[] args) {  
        Programador programador = new Programador();  
        programador.setNome("José da Silva");  
        programador.setSalario(5000);  
        Designer designer = new Designer();  
        designer.setNome("Maria Laura");  
        designer.setSalario(3000);  
        aplicarAumento(programador);  
        aplicarAumento(designer);  
    }  
    public static void aplicarAumento(Funcionario funcionario) {  
        double salarioAumentado = funcionario.getSalario() + 1000;  
        funcionario.setSalario(salarioAumentado);  
    }  
}
```

O método pode receber qualquer tipo de "Funcionário" como parâmetro



Padronização de código

Polimorfismo permite que tratemos objetos de uma mesma hierarquia de forma uniforme.

Do contrário, precisaríamos de um array e um método para cada tipo de funcionário...

E se fizermos isso?

```
public class ClassePrincipal {  
    public static void main(String[] args) {  
        Programador programador = new Programador();  
        programador.setNome("José da Silva");  
        programador.setSalario(5000);  
  
        Designer designer = new Designer();  
        designer.setNome("Maria Laura");  
        designer.setSalario(3000);  
  
        Funcionario[] funcionarios = new Funcionario[10];  
        funcionarios[0] = programador;  
        funcionarios[1] = designer;  
        for (int i = 0; i < funcionarios.length; i++) {  
            System.out.println(funcionarios[i].calcularBonificacao());  
        }  
    }  
}
```

**Método com a implementação
específica para cada subclasse**



E se fizermos isso?

```
public class ClassePrincipal {  
    public static void main(String[] args) {  
        Programador programador = new Programador();  
        programador.setNome("José da Silva");  
        programador.setSalario(5000);  
  
        Designer designer = new Designer();  
        designer.setNome("Maria Laura");  
        designer.setSalario(3000);  
  
        Funcionario[] funcionarios = new Funcionario[10];  
        funcionarios[0] = programador;  
        funcionarios[1] = designer;  
        for (int i = 0; i < funcionarios.length; i++) {  
            System.out.println(funcionarios[i].calcularBonificacao());  
        }  
    }  
}
```

Comportamento polimórfico garante
a chamada correta do método

Saída:

1000

500



Sendo assim...

Polimorfismo permite que objetos assumam **múltiplos tipos** e se **comportem** de **múltiplas formas** baseado em suas hierarquias

2.

Interfaces em Java

Polimorfismo

E se quiséssemos herdar mais de uma classe?

- **Java não permite herança múltipla.**
- Mas possui um mecanismo que "**simula**" **herança múltipla** de tipo.
- Para isso é possível usar o conceito de **interfaces**.

Situação-problema

- Você foi contratado para implementar um sistema para uma empresa que oferta **cursos de programação**. Sendo assim, "**Programadores**" também são "**Professores**".

```
public interface Professor {  
    public void darAula();  
}
```


Exemplo: interfaces em Java

```
public interface Professor {  
    public void darAula();  
}
```

Classe que implementa a interface **"Professor"**

```
public class Programador extends Funcionario implements Professor {  
    private String linguagem;  
    public double calcularBonificacao() {  
        double bonificacao = salario * 0.2;  
        return bonificacao;  
    }  
    public double darAula() {  
        System.out.println("DANDO AULA DE PROGRAMAÇÃO");  
    }  
}
```

Implementação do método da interface



Características de interfaces em Java

1. Permite "**herança múltipla**" de tipo.
2. Especifica um **contrato** que define tudo que uma classe faz, mas não **como** ela faz.
3. Uma interface **não pode ter atributos** (a menos que seja uma **constante**), pois não há herança de estado.

Características de interfaces em Java

4. Uma interface deve definir somente a **assinatura dos métodos**, mas não o corpo (**até Java 7**).
5. Até o **Java 8**, todos os métodos de uma interface só podiam ser **públicos**. Do Java 9 em diante, é possível ter **métodos privados**.

Exemplo: método "default" em interfaces

```
public interface Professor {  
    public default void darAula() {  
        System.out.println("DANDO AULA DE PROGRAMAÇÃO");  
    }  
}
```

Implementação padrão

```
public class Programador extends Funcionario implements Professor {  
    private String linguagem;  
    public double calcularBonificacao() {  
        double bonificacao = salario * 0.2;  
        return bonificacao;  
    }  
}
```

A classe não precisa implementar o método abstrato

Outras observações

- E se houver **conflito** da implementação padrão de um **método da interface** com a implementação de um **método herdado** (de classe concreta ou abstrata)?
- **A implementação da classe sobrescreve** a implementação padrão da interface.

Outras observações...

- E se houver **conflito** entre as **implementações padrão de duas ou mais interfaces**?
- **A classe** que implementa ambas as interfaces **é forçada a implementar uma versão que sobrescreva** todas as implementações padrão.

3.

Considerações finais

Polimorfismo

O que aprendemos hoje?

- Amadurecer o conceito de polimorfismo na programação orientada a objetos;
- Entender como o polimorfismo permite que objetos de diferentes classes sejam tratados de maneira uniforme;
- Compreender o conceito de interfaces e seu papel na programação orientada a objetos;
- Aprender a criar e implementar interfaces em Java.

Próxima aula...

-

4.

Exercício de fixação

Teams



Polimorfismo

- **Link da atividade**: clique aqui.

Aula 11

Polimorfismo

Programação III

Prof. Augusto César Oliveira

augusto.oliveira@unicap.br