Iremos apresentar nossa apresentação no dia 13/06/2023

Falaremos sobre:

# **Paradigmas de Programação**

Explicando que, conforme programas de computador eram criados antigamente, e ficavam cada vez mais complexos, linguagens de programação e boas práticas evoluíram para atender as necessidades dos programadores. Finalmente, essas evoluções criaram o que chamamos de paradigmas de programação.

E que um paradigma de programação é um estilo específico de organizar e escrever programas com o objetivo de aumentar a organização, ter menos bugs e melhorar a manutenção do código.

E explicar alguns pontos:

* É um padrão conceitual que orienta soluções de projeto e implementação
* Paradigmas explicam como os elementos que compõem um programa são organizados e como interagem entre si
* Exemplos de paradigmas:

- Procedural

Sequência de instruções modificando o estado.

- Orientado a Objetos

Organizado em torno de objetos com propriedades e comportamentos.

- Funcional

Baseado em funções puras e evita a mudança de estado.

# **O que é programação orientada a objetos**

Explicaremos o que é programação orientada a objetos falando ela ser um modelo de programação, onde diversas classes possuem características que definem um objeto na vida real. Cada classe determina o comportamento do objeto, definido por métodos e seus estados possíveis definidos por atributos. São exemplos de linguagens de programação orientadas a objetos: *C++, Java, C#,*

*Object Pascal, entre outras*. Este modelo foi criado com o intuito de aproximar o mundo real do mundo virtual. Para dar suporte à definição de Objeto, foi criada uma estrutura chamada Classe, que reúne objetos com características em comum, descreve todos os serviços disponíveis por seus objetos e quais informações podem ser armazenadas.

# **Os Quatro Pilares**

Falaremos sobre:

## **Abstração**

Abstrair algo significa esconder os detalhes da implementação dentro de algo – às vezes um protótipo, às vezes em uma função. Portanto, quando você chama a função, não precisa entender exatamente o que ela está fazendo.

E daremos exemplos como o funcionamento de um carro. Quando acionamos ele para ligar, não precisamos saber quais passos ele faz para colocar o motor em funcionamento. Quando acionamos o freio, não precisamos saber todos os mecanismos que são acionados para fazer o carro frear. Apenas sabemos o que cada objeto ou função do carro produz como resultado.

E o exemplo do computador, onde você pode saber usar o computador, mas você não necessita saber como cada código ou peça funciona. Sendo assim, podemos também dividir internamente problemas complexos em problemas menores, onde resolvemos cada um deles até encontrarmos a solução do problema inteiro.

## **Encapsulamento**

Encapsulamento é um princípio de design de código, geralmente ligado a programação orientada, que nos orienta a esconder as funcionalidades e funcionamento do nosso código dentro de pequenas unidades (normalmente métodos e funções)

A ideai é tornar o software mais flexível, fácil de modificar e de criar novas implementações. O Encapsulamento serve para controlar o acesso aos atributos e métodos de uma classe.

## **Herança**

A herança é um mecanismo da Orientação a Objeto que permite criar novas classes a partir de classes já existentes, aproveitando-se das características existentes na classe a ser estendida. Este mecanismo é muito interessante, pois promove um grande reuso e reaproveitamento de código existente.  Com a herança é possível criar classes derivadas, subclasses, a partir de classes bases, superclasses. As subclasses são mais especializadas do que as suas superclasses, mais genéricas. As subclasses herdam todas as características de suas superclasses, como suas variáveis e métodos. A linguagem Java permite o uso de herança simples, mas não permite a implementação de herança múltipla. Para superar essa limitação o Java faz uso de interfaces, o qual pode ser visto como uma que certos métodos com características previamente estabelecidas serão implementados, usando inclusive a palavra reservada implements para garantir esta implementação.

## **Polimorfismo**

O Polimorfismo é um mecanismo por meio do qual selecionamos as funcionalidades utilizadas de forma dinâmica por um programa no decorrer de sua execução.

Com o Polimorfismo, os mesmos atributos e objetos podem ser utilizados em objetos distintos, porém, com implementações lógicas diferentes.

Por exemplo: podemos assumir que uma bola de futebol e uma camisa da seleção brasileira são artigos esportivos, mais que o cálculo deles em uma venda é calculado de formas diferentes.

Outro exemplo: podemos dizer que uma classe chamada Vendedor e outra chamada Diretor podem ter como base uma classe chamada Pessoa, com um método chamado Calcular Vendas. Se este método (definido na classe base) se comportar de maneira diferente para as chamadas feitas a partir de uma instância de Vendedor e para as chamadas feitas a partir de uma instância de Diretor, ele será considerado um método polimórfico, ou seja, um método de várias formas.

## **Polimorfismo – Sobrecarga e Sobreposição**

O polimorfismo é um conceito amplo na programação orientada a objetos que envolve a capacidade de objetos de diferentes classes responderem de maneira diferente a uma mesma chamada de método. O polimorfismo pode ser alcançado por meio de duas técnicas: sobrecarga (overloading) e sobreposição (overriding).

## **Polimorfismo - Sobrecarga**

Sobrecarga (Overloading): A sobrecarga ocorre quando uma classe possui vários métodos com o mesmo nome, mas com diferentes assinaturas (parâmetros). Esses métodos podem ter um número diferente de parâmetros ou tipos de parâmetros diferentes. A decisão sobre qual método executar é baseada nos parâmetros passados na chamada do método.

A sobrecarga permite que uma classe ofereça diferentes versões de um método, cada uma adaptada para diferentes conjuntos de parâmetros. Dessa forma, os métodos podem ter o mesmo nome, mas se comportam de maneira diferente com base nos argumentos.

## **Polimorfismo - Sobreposição**

Sobreposição (Overriding): A sobreposição ocorre quando uma classe derivada (subclasse) substitui um método da classe base (superclasse) com a mesma assinatura (nome e parâmetros). A classe derivada fornece uma implementação diferente do método, que é específica para a própria classe derivada. A sobreposição permite que a classe derivada modifique ou estenda o comportamento do método herdado da classe base.

Quando um método é chamado em um objeto da classe derivada, a implementação do método na classe derivada é executada em vez da implementação na classe base. Isso permite que objetos de diferentes classes respondam de maneira diferente à mesma chamada de método, dependendo da classe real do objeto.

# **Objetos Estáticos**

Objetos estáticos em JavaScript são membros de uma classe que pertencem à própria classe e não a instâncias individuais. Eles podem ser acessados diretamente usando o nome da classe e são compartilhados por todas as instâncias. São usados para dados ou funções compartilhadas.

Neste exemplo, contador é um objeto estático que conta o número de vezes que o método incrementarContador() é chamado. O método getContador() retorna o valor atual do contador. O acesso aos membros estáticos é feito diretamente usando o nome da classe, sem a necessidade de criar instâncias da classe.

E daremos exemplos em JavaScript, para n final aplicarmos os exercícios