**Java – Orientação a objetos**

**“Trazer uma representação do mundo real para o computador.”**

**\* Onde posso aplicar?**

– Os conceitos de orientação a objetos podem ser aplicados a qualquer linguagem de programação que tenha suporte, utilizaremos o Java como base.

**Diferenças entre programação estruturada e programação orientada a objetos.**

**→ Programação estruturada:**

– É linear e estruturada como o próprio nome diz, ou seja, as tomadas de decisões são feitas de forma ordenada, por meio de verificações condicionais e interações, utilizando como recursos variáveis e rotinas para o desenvolvimento dos programas. Exemplos: linguagem C, Pascal, basic, etc;

– Processamento de cima para baixo. Exemplos (antigamente sem suporte para OO): linguagem C, Pascal, Basic, etc;

– Desvantagem da programação estruturada: quando utilizada para programas complexos, a organização do código torna-se um desafio justamente pela estrutura do mesmo não oferecer uma forma simples de agrupar os dados e protegê-los, deixando a cargo do programador todo o trabalho.

**→ Programação orientada a objetos:**

– É possível termos programas em uma linguagem com suporte a POO que utilizam componentes da POO e a implementação é estruturada, portanto, dizer que simplesmente o uso de uma linguagem OO faz o seu programa ser OO é um mito;

– A primeira linguagem OO foi desenvolvida nos anos 60 pelos noruegueses Ole-Johan Dahl e Kristen Nygaard. O **Simula 67** continha muitos dos conceitos das principais linguagens OO de hoje, como por exemplo: classes e objetos, subclasses, métodos virtuais, frameworks, gerenciamento de memória, etc;

– Chamamos de objetos a representação de algo, como, por exemplo, um carro, onde temos atributos e ações tomadas que podem ser tomadas ou executadas por ele. Um carro tem (propriedades): altura, largura, cor, combustível, cilindradas, número de passageiros e velocidade máxima. Ele pode: ligar, desligar, parar, andar e acelerar;

**→ Conceitos básicos da orientação a objetos:**

**– Classe e Objeto:** uma representação dados em objetos ou entidades para o processamento de outros objetos;

**– Associação de Classes:** quando utilizamos uma classe dentro de outra classe;

**– Herança:** é a utilização de uma classe base, fazendo com que uma nova classe tenha todos os atributos e funções da classe pai, mais as suas próprias;

**– Encapsulamento:** e a possibilidade de proteger alguns dados ou funcionalidades da classe, não permitindo que seus consumidores possam acessá-la;

**– Polimorfismo:** aqui podemos criar funções que terão o mesmo nome, mas que podem ter diferentes processamentos, implementações, ou na mesma classe o mesmo nome e diferentes entradas.

**Classes/funções estáticas utilitárias**

– Primeira atenção no dia a dia como programador é criar classes utilitárias, agrupadas por tema, de forma que você possa compartilhar com todo seu time e assim evitar duplicidade de código;

**→ Como identificar uma função utilitária? Verifique se ela atende alguns requisitos simples:**

1) Ela consegue se resolver por ela mesma, sem dependências externas?

2) Os parâmetros de entrada são simples e diretos?

3) O resultado de saída também é simples e direto?

Exemplo: um código de validação de CPF ou CNPJ.

**Swing / JFrame**

**- Comandos:**

setTitle ("Título"); // Título da janela

setVisible(true/false); // Mostrar ou ocultar janela

setSize(x, y); // Dimensões da janela

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Parar de executar ao clicar no botão fechar (X)

setResizable(true/false); // Permitir ou não redimensionar janela

setLocationRelativeTo(null); // Mostrar janela no centro da tela