**História do Java**

A História da Linguagem Java se inicia em 1990, quando a Sun Microsystems criou uma equipe chamada Green Team, liderada por James Gosling.

O grupo Green Team criou uma linguagem para substituir o C em uma atividade específica, chamada GreenTalk, que posteriormente foi batizada de Oak.

O Star Seven foi criado para interagir com os eletrodomésticos da casa, porém não teve sucesso na época e em 1992 o projeto foi engavetado.

Em 1994 surgiu o WebRunner, que era para ser um navegador interativo usando as ideias do projeto anterior.

O Green Team não podia registrar o nome Oak pois já existia uma linguagem com esse nome. Foi assim que eles mudaram para Java, que na época remetia a café forte.

**Como o Java funciona?**

O código fonte em Java é compilado pelo JavaC que gera um código binário chamado ByteCode, que ainda não é compreendido pelo computador, esse Bytecode precisa passar por uma JVM (Java Virtual Machine) que gera um código que a máquina consiga compreender.

O mesmo Bytecode pode ser enviado para todas as diferentes plataformas.

**JRE** – Java Runtime Environment

A JRE contém as bibliotecas e a JVM que contém o Loader / Verificado, Interpretado / Gerenciador e um compilador JIT.

**JDK** – Java Development Kit

A JDK contém a JRE, a JavaLang e a JavaTools que contém o JavaC, Debugger e APIs.

Obs: A JDK pode usar uma IDE que facilita o desenvolvimento.

**Compilador & Interpretador**

**Compilador:**

* O compilador passa por 2 processos, mas antes ele analisa tudo antes de traduzir.
* Esse código gerado ocupa mais memória.
* O tempo de execução é menor.

**Interpretador:**

* O Interpretador pega linha por linha e joga na memória. Após ser executada ela é apagada da memória.
* Ocupa menos memória.
* O tempo de execução é maior por conta de ele precisar traduzir.

**Java POO**

O criador da POO foi **Alan Kay**, um biólogo e matemático.

Todo software orientado a objeto é:

**Confiável:** O isolamento entre as partes gera software seguro. Ao alterar uma parte, nenhuma outra é afetada.

**Oportuno:** Ao dividir tudo em partes, várias delas podem ser desenvolvidas em paralelo.

**Manutenível:** Atualizar um software é mais fácil. Uma pequena modificação vai beneficiar todas as partes que usarem o objeto.

**Extensível:** O software não é estático. Ele deve crescer para permanecer útil.

**Reutilizável:** Podemos usar objetos de um sistema que criamos em outro sistema futuro.

**Natural:** Mais fácil de entender. Você se preocupa mais na funcionalidade do que nos detalhes de implementação.

**O que é um Objeto?**

Coisa material ou abstrata que pode ser percebida pelos sentidos e descrita por meio das suas características, comportamentos e estado atual.

Os objetos são baseados em classes. Essas classes precisam ter 3 coisas:

**Atributo** - Coisas que eu tenho?

Características

**Métodos** - Coisas que eu faço?

Ações

**Estado** - Como eu estou agora?

Estado atual do objeto são as características e a ação atual.

**Modificadores de Visibilidade**

Indicam o nível de acesso aos componentes internos de uma classe.

**+ public**

A classe atual e todas as outras classes podem acessar.

**- private**

Acesso somente a classe atual pode acessar.

**# protected**

A classe atual e todas as suas subclasses podem acessar.

**Métodos Especiais**

**Métodos Assessores** – Getters

**Métodos Modificadores** – Setters

Obs: Cada atributo precisa conter o seu getter e setter.

**Pilares da POO - EHP**

**Encapsulamento:** Ocultar partes independentes da implementação, permitindo construir partes invisíveis ao mundo exterior.

A troca de informações entre o mundo externo e um objeto é chamado de mensagem.

O canal de comunicação externa desse objeto se chama interface.

**Interface:** Lista de serviços fornecidos por um componente. É o contato com o mundo exterior, que define o que pode ser feito com o objeto dessa classe.

Obs: Encapsular não é obrigatório, mas é uma boa prática para produzir Classes mais eficientes.

**Vantagens de Encapsular:** Tornar mudanças invisíveis, facilitar reutilização do código e reduzir efeitos colaterais.

**Herança:** Permite basear uma nova classe na definição de uma outra classe previamente existente.

A herança será aplicada tanto para as características quanto para os comportamentos.

Uma classe progenitora tem subclasses.

Super classe é a classe que está logo acima da classe.

Classe que não possui super classe é considerada classe raiz.

As classes que não tem subclasses são chamadas de folhas.

Descendente é a classe que está logo acima da classe progenitora.

Ancestral é a classe que está logo abaixo da subclasse.

Quando se percorre a árvore de cima para baixo se chama Especialização.

Quando se percorre a árvore de baixo para cima se chama Generalização.

**Classe Abstrata:** Não pode ser instanciada. Só pode servir como progenitora.

**Método Abstrato:** Declarado, mas não implementado na progenitora.

**Classe Final:** Não pode ser herdada por outra classe. Obrigatoriamente folha.

**Método Final:** Não pode ser sobrescrito pelas suas subclasses. Obrigatoriamente herdado.

**Polimorfismo:** Permite que um mesmo nome represente vários comportamentos diferentes.

**Assinatura do método:** Consiste em quantidade e os tipos dos parâmetros.

**Sobreposição:** Acontece quando substituímos um método de uma superclasse na sua subclasse, usando a mesma assinatura.

**Sobrecarga:** Assinaturas diferentes com a mesma classe.

**Implements** é usado quando uma classe implementa uma interface, **Extends** é quando uma classe estende uma outra classe (concreta ou abstrata) ou quando uma interface estende uma outra interface.

**Exercícios**

1. **O que é algoritmo?**

Algoritmo é uma descrição das etapas de resolução de um problema ou a indicação ordenada de uma sequência de ações bem-definidas.

**2 - Qual é o utilitário responsável por gerar, a partir de um programa escrito em uma linguagem de alto nível, um programa em linguagem de máquina não executável chamado módulo-objeto?**

Compilador.

**3 - Uma casa está para uma planta arquitetônica assim como um objeto está para** uma classe.

**4 - A POO se preocupa em produzir software que tenha as seguintes características:**

COMERN - Confiável, Oportuno, Manutenível, Extensível, Reutilizável e Natural.

**Um objeto é uma instância de uma classe.**

**Um objeto é uma construção de software que encapsula estado e comportamento.**

**Uma classe define os atributos e comportamentos compartilhados por um tipo de objeto.**

**Em uma Linguagem POO pura, tudo é um objeto, desde os tipos mais básicos.**

**Uma variável interna ou atributo é um valor mantido dentro do objeto.**

**Atributos são características de uma classe visíveis externamente.**

**Comportamento são funcionalidades internas.**

**O estado de um objeto é o significado combinado das variáveis internas do objeto.**

**Atributo** - Característica de uma classe que é visível.

**Classe** – Define os atributos e comportamentos comuns.

**Comportamento** - Ação executada por um objeto.

**Domínio** - Espaço onde o problema reside.

**Objeto** - Construção de software que encapsula estado e comportamento.

**Construtor** - Métodos usados para inicializar objetos durante a instanciação.

**Acessor** - Métodos que dão acesso aos dados internos.

**Mutante** - Métodos que permitem que se altere o estado de um objeto.

**Tipos –** Define as diferentes espécies de valores que podem ser usados.

**Interface –** Define o que uma entidade pode fazer com o objeto.

**5 - A maioria das linguagens orientadas a objetos suporta 3 níveis de acesso:**

Público, protegido e privado.

**6 – Quando se utiliza o conceito de encapsulamento da POO, enquanto a** implementação **define os detalhes internos do componente, a**  interface **lista os serviços fornecidos por ele.**

**7 - O conceito de POO que significa representar uma entidade, incluindo apenas seus atributos mais relevantes é** abstração.

**8 - Um objeto pode conter outro objeto por meio de uma** agregação**. Quando temos variáveis internas que são instâncias de outras classes, temos uma** composição**.**