DESENVOLVIMENTO BÁSICO EM JAVA

Ferramentas utilizadas para instalação do Java:

* GANDLE - sistema avançado de automatização de builds
* MAVEN - **O Apache Maven é uma ferramenta de automação e gerenciamento de projetos Java**, embora também possa ser utilizada com outras linguagens. Ela fornece às equipes de desenvolvimento uma forma padronizada de automação, construção e publicação de suas aplicações, agregando agilidade e qualidade ao produto final
* INTELLIJ IDE

JAVA – Criação de Projetos

<https://start.spring.io/>

O QUE PRECISAMOS SABER SOBRE JAVA?

O que é JAVA?

É uma linguagem de programação e plataforma computacional criada em 95 pela San Microsystems, em um time comandado por James Gosling. Comprada anos depois pela Oracle.

Diferente de outras linguagens que são compiladas para código nativo, Java é compilado para um *bytecode* interpretado por uma máquina virtual.

O que é o COMPILADOR?

Programa que, a partir de um código fonte, cria um programa semanticamente equivalente, porém escrito em outra linguagem, código objeto. Ele traduz uma linguagem textual para uma linguagem de máquina específica para um processador e SO.

O nome compilador é usado principalmente para os programas que traduzem o código fonte de uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (ex.: ASSEMBLY, CÓDIGO DE MÁQUINA, etc).

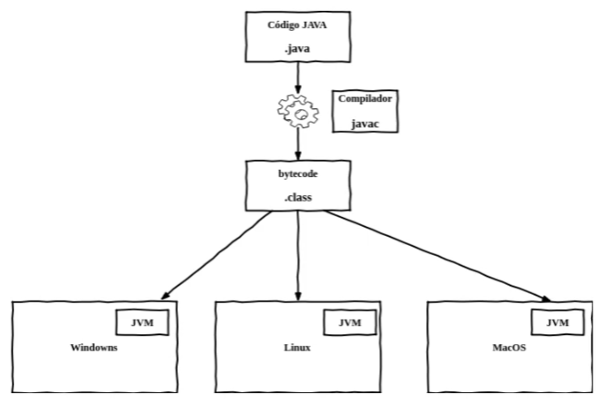
O que é o BYTECODE?

É o código originado da compilação de programas JAVA. É um programa interpretado e executado pela Máquina Virtual Java, JVM.

O que é uma VM?

Uma Máquina Virtual, ou Virtual Machine, é um software de simulação de uma máquina física que consegue executar vários programas, gerenciar processos, memória e arquivos. Isso tudo faz parte de uma plataforma com memória, processador e outros recursos totalmente virtuais, sem dependência do hardware.

O que é a JVM?

 É a máquina virtual que executa programas JAVA, executando os bytecodes em linguagem de máquina para sistema operacional.

Em linguagens compiladas diretamente para um SO específico, esse programa não irá executar em outro SO, sendo necessário então a compilação de uma versão do software para cada SO.

Com o JAVA, compilamos para a JVM, o bytecode será executado pela máquina virtual, e não diretamente no SO, desta forma o código escrito em JAVA possui portabilidade para qualquer SO, porém cada JVM deve ser construída para um SO específico.

JAVAC é o compilador do JAVA.

O que é a JRE?

JRE significa Java Runtime Environment, ou Ambiente de Execução do Java. Ele é composto pela Java Virtual Machine (JVM), bibliotecas e APIs da linguagem e outros componentes para suporte da plataforma JAVA.

É a parte responsável pela execução do software JAVA.

O que é o JAVA SE?

Java Standard Edition (SE), é a distribuição mínima da plataforma de desenvolvimento de aplicações JAVA.

OpenJDK é a implementação de referência opensource da plataforma JAVA, JAVA SE, que ainda é mantida pela Oracle.

O que é JAVA EE?

Java Enterprise Edition, é uma extensão do JAVA SE que possui suporte a desenvolvimento de sistemas corporativos.

Além do mínimo da plataforma JAVA, ele possui diversas especificações de partes da infraestrutura de aplicações, como acesso a banco de dados, mensageria, serviços web, parser de arquivos e outras.

Servidores de Aplicações Java EE, sabem seguir essas especificações e implementar os recursos para outros usuários.

Exemplos de servidores de aplicação Java EE: JBoss (RedHat), Weblogic (Oracle), WebSphere (IBM) e Glassfish (implementação de referência opensourse).

O que é JAKARTA EE?

Com a falta de investimento da Oracle no Java, ela cedeu todo o código, implementações e especificações do Java EE para a Eclipse Foundation, mas como o nome Java EE é uma marca registradas, foi escolhido no nome Jakarta EE.

Agora a evolução de especificações e padrões do Java será feita sob o nome Jakarta EE, com compatibilidade com o Java EE.

COMPILANDO CÓDIGO JAVA NO TERMINAL

LINUX - TERMINAL

Mkdir hello-java – cria o diretório

Cd hello-java – entra no diretório criado

Hello-java vim Hello.java – cria o arquivo .java

Após criar o arquivo .java inserimos um código dentro do arquivo para ser executado.

Javac Hello.java – compila o código do arquivo .java.

Java Hello – executa o código após ser compilado

Cat Hello.java – utilizado para ver o código fonte do arquivo .java.

INICIANDO UM PROJETO EM JAVA

1. Inicie o Intellij;
2. New Project
3. Escolha Maven ou Gradle
4. Name = empresa.da.nome – exemplo: one.innovation.digital
5. Finish
6. Verifique se as estruturas do projeto estão todas para o Java 11
   1. File -> Project Structure
   2. Project SDK = 11
   3. Project Language Level = 11 – Local variable syntax for lambda parametes

CLASSES

package one.digitalinnovation.classes;

public class Programa {

public static void main(String[] args){

System.out.println(“Hello world”);

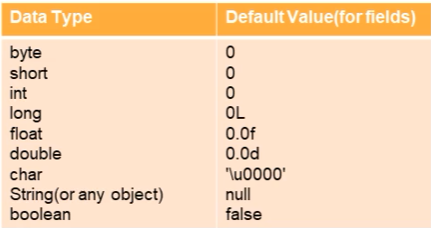
}

}

* Por convenção classes em Java sempre iniciam por letra maiúscula, nomes compostos também. Ex.:ImprimeTela, etc;
* O método principal para executar um programa em Java é o public static void main;
* (String[] args) – são os argumentos do método, sua assinatura, não necessariamente seu retorno
* Compõem uma classe: classe, atributos, construtores e métodos

TIPOS

* Primitivos
  + Não aceitam null
  + Possuem valores default



* Byte possuí 8 bits, portanto seu valor máximo vai de -128 até 127
* Short possuí 16 bits indo de -32768 até 32767
* Int possuí 32 bits indo de -2147483648 até 2147483647
* Long possuí 64 bits indo de -9223372036854745808L até 9223372036854775807L. É necessário inserir um L ao seu final
* Float possuí 32 bits e pode ser usado para números não inteiros. É necessário inserir um F ao final do número
* Double possuí 64 bits
* Char possuí 16 bits
* Boolean – TRUE ou FALSE
* Wrappers
  + São objetos que representam os primitivos através de uma classe
  + Auto-boxing – o compilador é responsável por autoconstruir o objeto, seguindo as mesmas regras dos tipos primitivos, quanto a seus tamanhos máximos e quantidade de caracteres. Aqui conseguimos atribuir null devido a variável se tornar um objeto.
  + Unboxing – conseguimos atribuir um wrapper a um tipo primitivo
    - Ex.: int inteiro = Integer.valueof(1024);
    - boolean b2 = Boolean.TRUE;

Então o unboxing é passar uma classe (instanciar um objeto) a um tipo inicialmente primitivo.

* Não primitivos
  + String – sequência de caracteres
  + Number -
  + Object – objeto principal do Java
  + Quaisquer outros objetos
* Tipagem forte (uma vez atribuído um tipo a variável não é possível alterá-lo) e estática (os tipos das variáveis são verificados em tempo de compilação, ou seja, não permite a variável receber outro tipo de atribuição além da qual foi criada. Ex.: Integer numero = “123456”; retorna erro porque não recebe String apenas números)
* Tipo Inferido – conseguimos criar variáveis a atribuir tipos a elas sem ter de falar de forma explícita qual o seu tipo, utilizando apenas VAR

var numero = Integer.valueOf(“123456”);

var texto = “O número é: “;

MODIFICADORES DE ACESSO

* Public
  + Pode ser acessado de qualquer lugar por qualquer entidade que possa visualizar a classe a que ela pertence
* Private
  + Métodos e atributos da classe não podem ser acessados ou usados por nenhuma outra classe. Esses atributos e métodos também não podem ser visualizados pelas classes herdadas.
* Protected
  + Torna o membro acessível às classes do mesmo pacote ou através de herança, seus membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.
* Default (padrão)
  + A classe e/ou seus membros são acessíveis somente por classes do mesmo pacote, na sua declaração não é definido nenhum tipo de modificador, sendo este identificado pelo compilador
* Abstract
  + Não pode ser aplicado em variáveis apenas em classes e métodos. Não pode ser instanciada. Caso houver algum método declarado como abstract a classe também deve ser marcada como abstract.
  + Ser uma classe abstrata significa que ela é uma ideia de uma classe, uma forma geométrica, por exemplo, onde podemos ter classes que são herdeiras dela, como um quadrado.
  + As classes herdeiras dela devem sobrescrever os métodos herdados.
* Static
  + É utilizado para criação de uma variável que estará disponível a todas as instâncias de objetos desta classe como uma variável comum. Ela será a mesma em todas as instâncias, ou seja, se modificada em uma parte essa alteração ocorre em todas as outras. Nas declarações de métodos ajudam no acesso direto à classe, portanto não é necessário instanciar um objeto para acessar o método.
* Final
  + Não permite estender, nos métodos impede que o mesmo seja sobrescrito (overriding) na subclasse, e nos valores de variáveis não pode ser alterado depois que já tenha sido atribuído um valor.

INTERFACES

A interface é um recurso muito utilizado em Java, bem como na maioria das linguagens orientadas a objeto, para “obrigar” a um determinado grupo de classes a ter métodos ou propriedades em comum para existir em um determinado contexto, contudo os métodos podem ser implementados em cada classe de uma maneira diferente. Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser implementado.

* Métodos Abstratos
  + Devem ser implementados por todos
  + Novos métodos quebram as implementações
* Métodos Default
  + São herdados a todos que implementam
  + Novos métodos não quebram as implementações
* Herança Múltipla – herda de mais de uma classe, em Java só é possível no conceito de interfaces
  + Ex.:

public class Fiesta implements Carro, Veiculo {

@Override

public String marca() {

return “Ford”;

}

@Override

public void ligar() {

Carro.super.ligar();

Veiculo.super.ligar();

}

}

INTERFACES vs ABSTRAÇÕES

Interfaces – é uma espécie de classe, que tem nome e informações, aqui não conseguimos colocar modificadores de acesso, portanto tudo inserido nela é public e static. Ex.: interface cliente possui nome, cpf, rg, etc.

Classe Abstrata – é uma classe que herda de outras classes. Ex.: classe Pessoa dela são herdadas outras classes como Cliente e Cidadão, por exemplo, que são herdeiras da classe Pessoa. Temos aqui o conceito de herança onde Cliente e Cidão herdam da classe Pessoa suas características, como por exemplo, atributos como Nome, Tipo Sanguíneo, Idade, Data de Nascimento, etc., métodos como andar, falar, pensar, etc..

Nela também é possível termos métodos privados, métodos públicos, métodos protegidos, etc. Não conseguimos instanciar a classe (new Cidadão). Nem toda linguagem suporta herança múltipla.

Quando usar interfaces?

Quando a classe necessitar de um contrato, ou seja, quando for preciso garantir que uma classe vai possuir exatamente aqueles métodos e atributos. Então basicamente ela é um contrato de garantia que o que for usado por ela terá exatamente aqueles métodos e atributos.

Quando usar classes abstratas?

Quando for aplicar padrões de projetos, para boas práticas de programação, principalmente pensando em reuso. Ela é uma abstração, serve efetivamente para conter informações abstratas, códigos abstratos, coisas que vão se concretizar futuramente, ou coisas que são efetivamente comuns para vários objetos.

É um contrato? Use interface

É uma abstração? Use uma classe abstrata

ENUMS

* São conjuntos de valores pré-definidos;
* É uma espécie de dicionário de dados imutável;
* Não é permitido criar novas instâncias;
* O construtor é sempre declarado como private;
* Por convenção, por serem objetos constantes e imutáveis (static final), os nomes são em MAIÚSCULOS.

Exemplo: Estamos fazendo o sistema de um curso, suas aulas podem acontecer de MANHÃ, TARDE ou NOITE.

public enum PeriodoEnum {

MANHA,

TARDE,

NOITE

}

Utilizamos enum para padronizar determinadas informações que serão padrão em todo o sistema e para garantir que serão sempre as mesmas informações.