**Anotações sobre java**

O maven (baseado em xml, nasceu para substituir o Ant) é mais antigo que o Gradle que é uma ferramenta que está ficando bem popular e usa uma linguagem de programação chamada de groovy (os dois são ferramentas de bilding).

Um dos problemas de usar essas ferramentas e que se o trabalho for colaborativo todos os participantes têm que ter a mesma versão.

O Maven e o Gradle têm ciclos de vida de tarefas diferentes, eles seguem o mesmo conceito, mas fazem coisas diferentes.

**O QUE SABER SOBRE JAVA**

Java é uma linguagem de programação e de plataforma computacional lançada em 1995 pela Sun Microsystems pelo time comandado por James Gosling e que anos depois foi adquirida pela Oracle. O java é compilado para bytecode (código originado da compilação de programas java) que é interpretado por uma máquina virtual

**Compilador**

É um programa que a partir do código fonte cria um programa semelhante equivalente, mas escrito em outra linguagem (código objeto). O compilador vai traduzir um programa de uma linguagem textual para linguagem de máquina específica para um processador e S.O.O compilador traduzem o código fonte de uma linguagem de alto nível para uma linguagem de baixo nível.

**VM (Virtual Machine) e JVM**

Uma VM é um software que simula uma máquina física e consegue executar vários programas, gerenciar processos, memória e arquivos. Já a JVM é uma máquina que executa programa java que vai executar os bytescodes em linguagem de máquina. a vantagem de usar a JVM é que não vai ser precisos compilar o código pra cada sistema operacional.

**JRE**

Ambiente de execução do java é composto pela JVM, bibliotecas e APIS da linguagem Java e outros componentes para suporte da plataforma Java ele vai representar a parte responsável pela execução do software java.

$ sudo apt-get install openjdk-8-jre.

**JDK**

Kit de desenvolvimento java é um conjunto de utilitários que permitem softwares criar softwares para a plataforma java é composto pelo compilador java, bibliotecas de linguagem, ferramentas e o JRE.

**JAVA SE**

É a distribuição mínima da plataforma de desenvolvimento de aplicações java.

**JAVA EE**

É uma extensão da java se que possui suporte ao desenvolvimento de sistemas corporativos. Possui diversas especificações de partes da infraestrutura de aplicações, como acesso a banco de dados,mensageira, serviços web, parser de arquivos. Algumas aplicações sabem seguir essas especificações e implementar os recursos para usuários com o JBoss (REDHAT), Weblogic (Oracle), WebSphare (IBM) e Glassfish.

**Tipos**

* Primitivos

Não podem ser nulos, possuem valores default e podem não ser inicializados (byte(8bits) consegue receber no máx 127 e -128, short (16 bits), int(32 bites), long(64 bits), float, double(64 bits), char(16 bits), String e boolean).

* Wrappers

São objetos que representam os primitivos, mas que você pode atribuir valores nulos a eles e são autoboxing ex:Integer, Long (onde o compilador já sabe da classe que é automaticamente construída) e o unboxing (quando você tem a intenção de instanciar um objeto, e você consegue atribuir ao tipo primitivo)

* Não primitivos

Basicamente são os objetos

* Tipagem forte e estática

**Tipagem estática**: os tipos das variáveis são verificados em tempo de compilação

Exemplo: Integer num=”17767”

\*o código já vai mostrar o erro.

**Tipagem Forte**: quando eu atribuo um tipo a uma variável eu não posso mudá-la.

Exemplo: String txt= ” olá”; //ok

Texto=123; //não

**Tipo Inferido** :conseguimos atribuir valores as variáveis sem ter que especificar qual é o tipo delas.

**Modificadores de acesso**

**Public.**

Podem ser acessados de qualquer lugar por qualquer entidade que possa visualizar a classe a que ela pertence.

**Private**

Os métodos e atributos da classe não podem ser acessados ou usado por nenhuma outra classe e não podem ser visualizados pelas classes herdadas

**Default**

As classes e seus membros são acessíveis por classes do mesmo pacote, na sua declaração não é definido nenhum tipo de modificador (identificado pelo compilador)

**Protected**

Torna o membro acessível as classes do mesmo pacote ou através de herança, membros herdados não são acessíveis as outras classes fora do pacote.

**Abstract**

Não é aplicado nas variáveis, apenas nas classes e métodos. Uma classe abstrata não pode ser instanciada e se houver algum método declarado como abstract a classe também deve ser marcada como abstract.

\*métodos abstratos não tem corpo;

**Static**

Usada na criação de uma variável que poderá ser acessada por todas as instâncias de objetos desta classe como uma variável comum, ou seja, a variável criada será a mesma em todas as instancias e quando seu conteúdo é modificado numa instancia essa modificação vai acontecer em todas.

**Final**

Posso aplicar em classe, não permite estender, nos métodos impede que ele seja sobrescrito e nas variáveis ele não pode ser alterado.

**Métodos abstratos(interface)**

Implementados por todos e novos métodos quebram as implementações.

**Métodos default**

São herdados a todas que implementam e novos métodos não quebram as implementações.

Herança múltipla ( java não faz)

**Enum**

São dicionários de dados imutáveis, que não permitem criar instancias e o construtor e sempre private e por convenção os nomes são maiúsculos

**String**

É uma sequência de caracteres

IF e if tenário

**Convenções de nomes**

**Nome de classes**

Primeira letra do nome da classe em maiúscula e o resto minúsculo

**Nomes compostos**

Primeira letra do nome em maiúscula e o resto minúsculo e a mesma coisa com o segundo nome;

**Nomes de métodos**

Os nomes dos métodos devem ser bem descritivos, devem estar em letra minúscula, e se o nome for composto a segunda palavra deve começar com a letra maiúscula.

Checkstyle

 Analisa o estilo e convenções do código-fonte. Não analisa erros e possibilidade de erros no código, como o PMD e FindBugs, mas verifica a conformidade dele em relação aos padrões estabelecidos (documentação, comentários, sintaxe)

Pmd

Analisa o código-fonte à procura de problemas potenciais (bugs e warnings), código não utilizado, código de baixa qualidade, expressões complicadas ou ilegíveis e duplicidade de código.

**Orientação de Objetos**

*A programação Orientada a Objetos impõe disciplina sobre transferência indireta do controle.*by :Uncle Bob (Robert Martin)

**Java.Util.Date**

O java.util.date está na JDK desde sua versão 1.0. então algumas coisas podem estar obsoletas.

Date ();

Este construtor vai alocar um objeto da classe date e o inicializara com o milissegundo mais próximo do período da sua execução.

Date (long date)

Diferente do construtor anterior, esse construtor espera que você passe os milissegundos com base padrão de tempo que usa como referência 1 de janeiro de 1970 00:00:00.

Tabela

Descrição gerada automaticamente

**Classe Instant**

Imutável e Thread safe

Modela um ponto instantâneo de uma linha do tempo

Indicado para gravar marcações temporais em eventos da sua aplicação

**Java.util.Calendar**

Na JDK 1.1 foi observada a necessidade de facilitar alguns recursos que a classe Date oferecia. Com isso uma série de métodos e construtores da classe Date foi depreciada. EX: Date (int year, int month,int date)

Calendar é uma classe abstrata que provê métodos para converter data entre um instante específico.

Java +8

O java 8 veio com uma série de novidades para facilitar o trabalho com Datas.E a grande melhoria está no pacote java.time que foi herdado do projeto Joda Time.

**LocalDate** é uma classe imutável para representar uma data(yyy-MM-dd)

LocalTime é uma classe imutável que representa um padrão de hora-minuto-segundo. Pode ser representado até o nível de nanosegundos.

LocalDateTime é como se fosse uma junção dos dois, também é uma classe imutável e você consegue trabalhar com dia e hora de uma vez.

**Exceptions**

Exceções são todos os erros que ocorrem durante o processamento de um método que podem ser ou não esperados ou seja exceptions são falhas que não devem ocorrer rotineiramente no fluxo de um sistema.

**Checked Exceptions**

São exceções esperadas, cujo fluxo ou método de um sistema foi preparado para receber. Um exemplo seria as exceções de negócios.

**Unchecked Exceptions**

São exceções não esperadas para fluxo ou métodos de um sistema, um exemplo é a NullPointException que ocorre quando se tenta acessar uma referência de memória vazia ou recuperar alguma coisa que não existe.

**Try Catch**

Esse bloco sempre é utilizado quando no processo que será executado dentro de um método é esperado um erro, então cria-se um bloco protegido onde qualquer erro que ocorra dentro do try é direcionado para o catch e sofrerá o devido tratamento de erro.

**Finally**

É o bloco de código que pode ser utilizado junto ao try catch, este trecho de código será executado independentemente se ocorrer erro ou não dentro do fluxo. Normalmente o finally libera recursos ou para dar continuidade em um fluxo que deve ocorrer independente do erro.

**Throw**

É usado para lançar a exceção desejada, juntamente com a mensagem de erro, para o método que fez a chamada.

**Throws**

É a assinatura do método que será retornado caso ocorra erro para o método que fez a chamada, dentro de um fluxo encadeado.