

Apresentação do componente.**DESENVOLVIMENTO PARA SERVIDORES I (4)²⁴**

Objetivos gerais. Esta disciplina fornecerá uma visão geral da linguagem *script* PHP associado a um gerenciador de banco de dados que utilize a linguagem SQL e como usar essas tecnologias para gerar sites dinâmicos. Introduzir práticas de codificação seguras.

Objetivos específicos. Os estudantes deverão ser capazes de desenvolver um CMS simples ou um aplicativo *Web* completo (lado cliente e lado servidor).

Ementa. PHP histórico e emprego. Instalação e configuração básica do PHP e um IDE. Sintaxe básica do PHP. Usando o PHP como um mecanismo de modelo simples. Panorama das melhores práticas com PHP. Conceitos de programação HTTP. Codificação de caracteres. Localidades, fusos horários e funções de tempo. *Strings*. Uso de Array e funções de matriz. Orientação a objetos em PHP (Classes, objetos, herança, encapsulamento, polimorfismo, agregação, composição e métodos). Tratamento de exceções de erro. Arquitetura do lado do servidor. Manipulação de dados postados. Enviando e-mail. Sessões e autenticação. Cookies. Arquivo manuseio e armazenamento de dados em arquivos de texto. Gerenciador de banco de dados e suas funções. *Frameworks*. Web Services, API, RSS, JSON e Ajax. Hospedagem compartilhada.

Bibliografia básica

BEIGHLEY, L; MORRISON, M. *Use a cabeça! PHP & MySQL*. São Paulo: Alta Books, 2011.

DALL'OGGIO, P. *PHP - programando com orientação a objetos*. São Paulo: Novatec, 2009.

YANK, K. *Build your own database driven web site using PHP*. New York: Oreilly & Assoc, 2012.

Bibliografia complementar

DOYLE, M. *Beginning PHP 5.3*. Indianapolis: Wiley Pub, 2009.

GILMORE, W. J. *Dominando PHP e MYSQL do iniciante ao profissional*. Rio de Janeiro: Starlin Alta Consult, 2008.

1-) Comunicação de alunos com alunos e professores:

- Um e-mail para a sala é de grande valia para divulgação de material, notícias e etc.
- Criação de grupo nas redes sociais também é interessante.

2-) Uso de celulares:

Para o bom andamento das aulas, recomendo que utilizem os celulares em *vibracall*, para não atrapalhar o andamento da aula.

3-) Material das aulas:

A disciplina trabalha com NOTAS DE AULA que são disponibilizadas ao final de cada aula, para isso vamos utilizar o GitHub, nosso repositório de aula será: <https://github.com/marcossousa33dev/FatecSRDSI202501>

4-) Prazos de trabalhos e atividades:

Toda atividade solicitada terá uma data limite de entrega, de forma alguma tal data será postergada, ou seja, se não for entregue até a data limite a mesma receberá nota 0, isso tanto para atividades entregues de forma impressa ou enviadas ao e-mail da disciplina. Qualquer problema que tenham, me procurem com antecedência para verificarmos o que pode ser realizado.

5-) Qualidade do material de atividades:

- Impressas ou manuscritas:
Muita atenção na qualidade do que será entregue, atividades sem grampear, faltando nome e número de componentes, rasgadas, amassadas, com rebarba de folha de caderno e etc. serão desconsiderados por mim.
- Digitais:
Ao enviarem atividades para o e-mail da disciplina, SEMPRE no assunto deverá ter o nome da atividade que está sendo enviada, e no corpo do e-mail deverá ter o(s) nome(s) do(s) integrante(s) da atividade, sem estar desta forma a atividade será DESCONSIDERADA.

6-) Critérios de avaliação:

Cada trimestres teremos as seguintes formas de avaliação:

- Práticas em Laboratório;
- Assiduidade;
- Outras que se fizerem necessário.

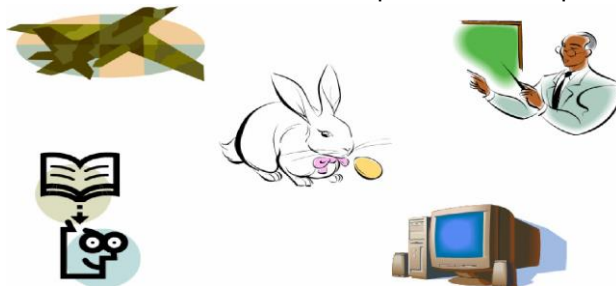
1. Introdução

Iremos relatar nessa fase de nosso curso a orientação à objetos com definições que serão importantes e que podemos implementar em qualquer linguagem de programação que seja OO.

2. Objetos

O Que São Objetos?

De acordo com o dicionário: - Objeto: "1. Tudo que se oferece aos nossos sentidos ou à nossa alma. 2. Coisa material: Havia na estante vários objetos. 3. Tudo que constitui a matéria de ciências ou artes. 4. Assunto, matéria. 5. Fim a que se mira ou que se tem em vista".



A figura destaca uma série de objetos. Objetos podem ser não só coisas concretas como também coisas inanimadas, como por exemplo uma matrícula, as disciplinas de um curso, os horários de aula.

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no *software*. Cada classe possui um comportamento (definidos pelos métodos) e estados possíveis (valores dos atributos) de seus objetos, assim como o relacionamento com outros objetos.

Há alguns anos, Alan Kay, um dos pais do paradigma da orientação a objetos, formulou a chamada "analogia biológica". Nessa analogia ele imaginou como seria um sistema de software que funcionasse como um ser vivo, no qual cada "célula" interagiria com outras células através do envio de mensagens para realização do objetivo comum.

De uma forma mais geral, Kay pensou em como construir um sistema de software a partir de agentes autônomos que interagem entre si. Ele então estabeleceu os seguintes princípios da orientação a objetos:

- (a) Qualquer coisa é um objeto;
- (b) Objetos realizam tarefas através da requisição de serviços a outros objetos;
- (c) Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares;
- (d) A classe é um repositório para comportamento associado ao objeto;
- (e) Classes são organizadas em hierarquias.

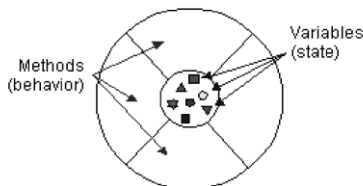
Quando temos um problema e queremos resolvê-lo usando um computador, necessariamente temos que fazer um programa. Este nada mais é do que uma série de instruções que indicam ao computador como proceder em determinadas situações. Assim, o grande desafio do programador é estabelecer a associação entre o modelo que o computador usa e o modelo que o problema lhe apresenta. Isso geralmente leva o programador a modelar o problema apresentado para o modelo utilizado em alguma linguagem. Se a linguagem escolhida for LISP, o problema será traduzido como listas encadeadas. Se for Prolog, o problema será uma cadeia de decisões. Assim, a representação da solução para o problema é característica da linguagem usada, tornando a escrita difícil.

A orientação a objetos tenta solucionar esse problema. A orientação a objetos é geral o suficiente para não impor um modelo de linguagem ao programador, permitindo a ele escolher uma estratégia, representando os aspectos do problema em objetos. Assim, quando se "lê" um programa orientado a objeto, podemos ver não apenas a solução, mas também a descrição do problema em termos do próprio problema. O programador consegue "quebrar" o grande problema em pequenas partes que juntas fornecem a solução.

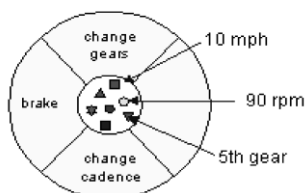
Olhando à sua volta é possível perceber muitos exemplos de objetos do mundo real: seu cachorro, sua mesa, sua televisão, sua bicicleta. Esses objetos têm duas características básicas, que são o **estado** e o **comportamento**. Por exemplo, os cachorros tem nome, cor, raça (estados) e eles balançam o rabo, comem, e latem (comportamento). Uma bicicleta tem 18 marchas, duas rodas, banco (estado) e elas brecam, aceleram e mudam de marcha (comportamento).

De maneira geral, definimos objetos como um conjunto de variáveis e métodos, que representam seus estados e comportamentos.

Veja a ilustração:



Temos aqui representada (de maneira lógica) a ideia que as linguagens orientadas a objeto utilizam. Tudo que um objeto sabe (estados ou variáveis) e tudo que ele pode fazer (comportamento ou métodos) está contido no próprio objeto. No exemplo da bicicleta, poderemos representar o objeto como no exemplo a seguir:



Temos métodos para mudar a marcha, a cadência das pedaladas, e para frear. A utilização desses métodos altera os valores dos estados. Ao frear, a velocidade diminui. Ao mudar a marcha, a cadência é alterada. Note que os diagramas mostram que as variáveis do objeto estão no centro do mesmo. Os métodos cercam esses valores e os “escondem” de outros objetos. Deste modo, só é possível ter acesso a essas variáveis através dos métodos. Esse tipo de construção é chamada de encapsulamento, e é a construção ideal para objetos.

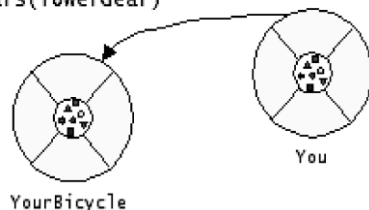
Mas um objeto sozinho geralmente não é muito útil. Ao contrário, em um programa orientado a objetos temos muitos objetos se relacionando entre si. Uma bicicleta encostada na garagem nada mais é que um pedaço de ferro e borracha. Por si mesma, ela não tem capacidade de realizar nenhuma tarefa. Apenas com outro objeto (você) utilizando a bicicleta, ela é capaz de realizar algum trabalho.

A interação entre objetos é feita através de mensagens. Um objeto “chama” os métodos de outro, passando parâmetros quando necessário. Quando você passa a mensagem “mude de marcha” para o objeto bicicleta você precisa dizer qual marcha você deseja.

A figura a seguir mostra os três componentes que fazem parte de uma mensagem:

- Objeto para o qual a mensagem é dirigida (bicicleta)
- Nome do método a ser executado (muda marcha)
- Os parâmetros necessários para a execução do método.

`changeGears(lowerGear)`



A troca de mensagens nos fornece ainda um benefício importante. Como todas as interações são feitas através delas, não importa se o objeto faz parte do mesmo programa; ele pode estar até em outro computador. Existem maneiras de enviar essas mensagens através da rede, permitindo a comunicação remota entre objetos.

Assim, um programa orientado a objetos nada mais é do que um punhado de objetos dizendo um ao outro o que fazer. Quando você quer que um objeto faça alguma coisa, você envia a ele uma “mensagem” informando o que quer fazer, e o objeto faz. Se ele precisar de outro objeto que o auxilie a realizar o “trabalho”, ele mesmo vai cuidar de enviar mensagem para esse outro objeto. Deste modo, um programa pode realizar atividades muito complexas baseadas apenas em uma mensagem inicial.

Você pode definir vários tipos de objetos diferentes para seu programa. Cada objeto terá suas próprias características, e saberá como interpretar certos pedidos. A parte interessante é que

objetos de mesmo tipo se comportam de maneira semelhante. Assim, você pode ter funções genéricas que funcionam para vários tipos diferentes, economizando código.

Vale lembrar que métodos são diferentes de procedimentos e funções das linguagens procedurais, e é muito fácil confundir os conceitos. Para evitar a confusão, temos que entender o que são classes.

3. O que é uma classe?

Assim como os objetos do mundo real, o mundo da Programação Orientada a Objetos (POO) agrupa os objetos pelos comportamentos e atributos comuns.

A biologia classifica todos os cães, gatos, elefantes e seres humanos como mamíferos. Características compartilhadas dão a essas criaturas separadas um senso de comunidade. No mundo do software, as classes agrupam objetos relacionados da mesma maneira.

Uma classe define todas as **características comuns a um tipo de objeto**. Especificamente, a classe **define todos os atributos e comportamentos** expostos pelo objeto. A classe define às quais mensagens o seu objeto responde. Quando um objeto quer exercer o comportamento de outro objeto, ele não faz isso diretamente, mas pede ao outro objeto para que se mude, normalmente baseado em alguma informação adicional. Frequentemente, isso é referido como “envio de mensagem”.

Uma *classe* define os atributos e comportamentos comuns compartilhados por um tipo de objeto. Imagine a classe como a forma do objeto. Os objetos de certo tipo ou classificação compartilham os mesmos comportamentos e atributos. As classes atuam de forma muito parecida com um cortador de molde ou biscoito, no sentido de que você usa uma classe para criar ou *instanciar* objetos.

Os *atributos* são as características de uma classe visíveis externamente. A cor dos olhos e a cor dos cabelos são exemplos de atributos. Um objeto pode expor um atributo fornecendo um link direto a alguma variável interna ou retornando o valor através de um método.

Comportamento é uma ação executada por um objeto quando passada uma mensagem ou em resposta a uma mudança de estado: é algo que um objeto faz. Um objeto pode executar o *comportamento* de outro, executando uma operação sobre esse objeto. Você pode ver os termos: *chamada de método*, *chamada de função* ou *passar uma mensagem*; usados em vez de executar uma *operação*. O que é importante é que cada uma dessas ações omite o comportamento de um objeto.

A definição de classes e seus inter-relacionamentos é o principal resultado da etapa de projeto de software. Em geral, esse resultado é expresso em termos de alguma linguagem de modelagem, tal como UML.

Uma classe é um gabarito para a definição de objetos. Através da definição de uma classe, descreve-se que propriedades — ou atributos — o objeto terá.

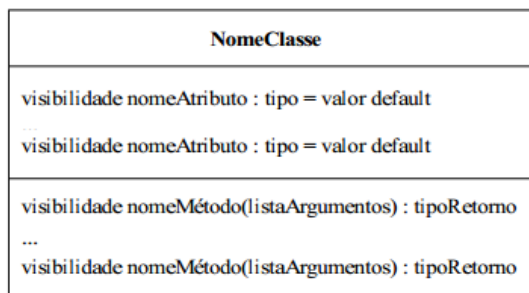
Além da especificação de atributos, a definição de uma classe descreve também qual o comportamento de objetos da classe, ou seja, que funcionalidades podem ser aplicadas a objetos da classe.

Essas funcionalidades são descritas através de métodos. Um método nada mais é que o equivalente a um procedimento ou função, com a restrição que ele manipula apenas suas variáveis locais e os atributos que foram definidos para a classe.

Uma vez que estejam definidas quais serão as classes que irão compor uma aplicação, assim como qual deve ser sua estrutura interna e comportamento, é possível criar essas classes em Java.

Na *Unified Modeling Language* (UML), a representação para uma classe no diagrama de classes é tipicamente expressa na forma gráfica, como mostrado na Figura a seguir.

Como se observa nessa figura, a especificação de uma classe é composta por três regiões: o nome da classe, o conjunto de atributos da classe e o conjunto de métodos da classe.



O nome da classe é um identificador para a classe, que permite referenciá-la posteriormente por exemplo, no momento da criação de um objeto.

O conjunto de atributos descreve as propriedades da classe. Cada atributo é identificado por um nome e tem um tipo associado. Em uma linguagem de programação orientada a objetos pura, o tipo é o nome de uma classe. Na prática, a maior parte das linguagens de programação orientada a objetos oferecem um grupo de tipos primitivos, como inteiro, real e caráter, que podem ser usados na descrição de atributos. O atributo pode ainda ter um valor_default opcional, que especifica um valor inicial para o atributo.

Os métodos definem as funcionalidades da classe, ou seja, o que será possível fazer com objetos dessa classe. Cada método é especificado por uma assinatura, composta por um identificador para o método (o nome do método), o tipo para o valor de retorno e sua lista de argumentos, sendo cada argumento identificado por seu tipo e nome.

Através do mecanismo de sobrecarga (*overloading*), dois métodos de uma classe podem ter o mesmo nome, desde que suas assinaturas sejam diferentes. Tal situação não gera conflito pois o compilador é capaz de detectar qual método deve ser escolhido a partir da análise dos tipos dos argumentos do método. Nesse caso, diz-se que ocorre a ligação prematura (*early binding*) para o método correto.

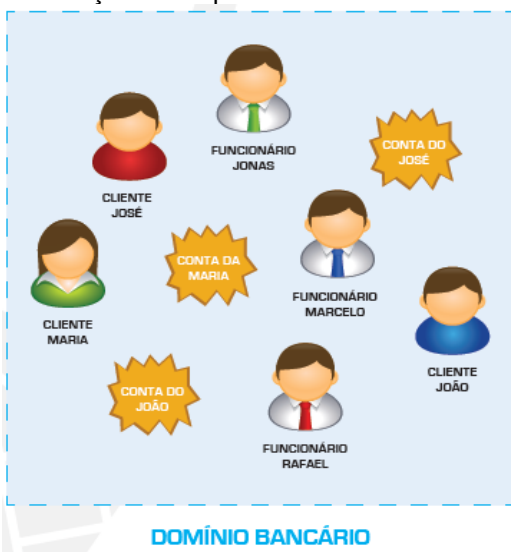
O modificador de visibilidade pode estar presente tanto para atributos como para métodos. Em princípio, três categorias de visibilidade podem ser definidas:

- público, denotado em UML pelo símbolo +: nesse caso, o atributo ou método de um objeto dessa classe pode ser acessado por qualquer outro objeto (visibilidade externa total);
- privativo, denotado em UML pelo símbolo -: nesse caso, o atributo ou método de um objeto dessa classe não pode ser acessado por nenhum outro objeto (nenhuma visibilidade externa);
- protegido, denotado em UML pelo símbolo #: nesse caso, o atributo ou método de um objeto dessa classe poderá ser acessado apenas por objetos de classes que sejam derivadas dessa através do mecanismo de herança.

4. Domínio e Aplicação

Um domínio é composto pelas entidades, informações e processos relacionados a um determinado contexto. Uma aplicação pode ser desenvolvida para automatizar ou tornar factível as tarefas de um domínio. Portanto, uma aplicação é basicamente o “reflexo” de um domínio.

Para exemplificar, suponha que estamos interessados em desenvolver uma aplicação para facilitar as tarefas do cotidiano de um banco. Podemos identificar clientes, funcionários, agências e contas como entidades desse domínio. Assim como podemos identificar as informações e os processos relacionados a essas entidades.



Observação:

A identificação dos elementos de um domínio é uma tarefa **difícil**, pois depende fortemente do conhecimento das entidades, informações e processos que o compõem. Em geral, as pessoas

que possuem esse conhecimento ou parte dele estão em contato constante com o domínio e não possuem conhecimentos técnicos para desenvolver uma aplicação.

Desenvolvedores de software buscam constantemente mecanismos para tornar mais eficiente o entendimento dos domínios para os quais eles devem desenvolver aplicações.

5. Objetos, Atributos e Métodos

As entidades identificadas no domínio devem ser representadas de alguma forma dentro da aplicação correspondente. Nas aplicações orientadas a objetos, as entidades são representadas por objetos.

- Uma aplicação orientada a objetos é composta por objetos.
- Em geral, um objeto representa uma entidade do domínio.

Para exemplificar, suponha que no domínio de um determinado banco exista um cliente chamado João. Dentro de uma aplicação orientada a objetos correspondente a esse domínio, deve existir um objeto para representar esse cliente.

Suponha que algumas informações do cliente João como nome, data de nascimento e sexo são importantes para o banco. Já que esses dados são relevantes para o domínio, o objeto que representa esse cliente deve possuir essas informações. Esses dados são armazenados nos atributos do objeto que representa o João.

- Um atributo é uma variável que pertence a um objeto.
- Os dados de um objeto são armazenados nos seus atributos.

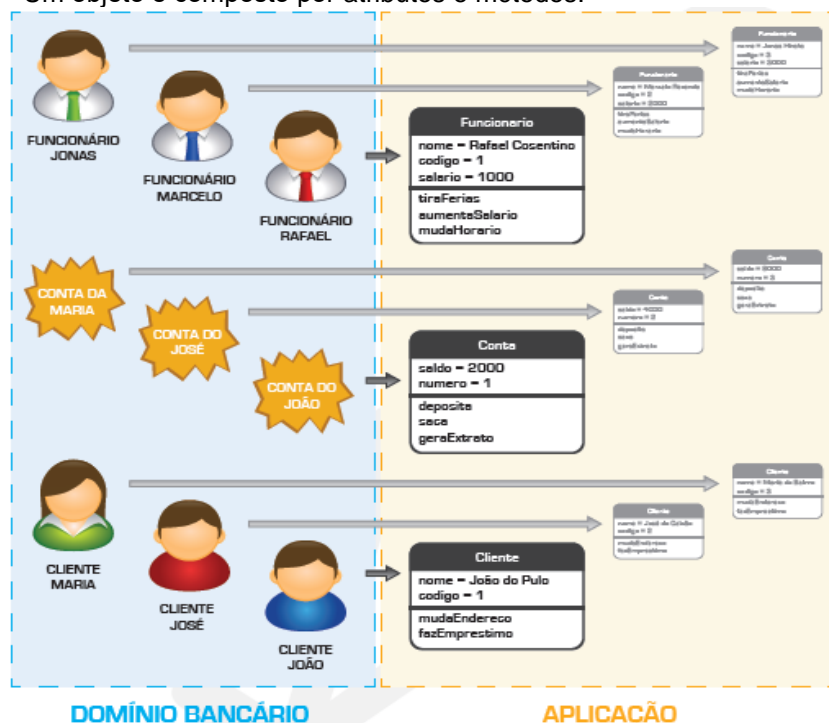
O próprio objeto deve realizar operações de consulta ou alteração dos valores de seus atributos.

Essas operações são definidas nos métodos do objeto.

Os métodos também são utilizados para possibilitar interações entre os objetos de uma aplicação.

Por exemplo, quando um cliente requisita um saque através de um caixa eletrônico do banco, o objeto que representa o caixa eletrônico deve interagir com o objeto que representa a conta do cliente.

- As tarefas que um objeto pode realizar são definidas pelos seus métodos.
- Um objeto é composto por atributos e métodos.



Observações:

- Em geral, não é adequado utilizar o objeto que representa um determinado cliente para representar outro cliente do banco, pois os dados dos clientes podem ser diferentes. Dessa forma, para cada cliente do banco, deve existir um objeto dentro

do sistema para representá-lo.

- Os objetos não representam apenas coisas concretas como os clientes do banco. Eles também devem ser utilizados para representar coisas abstratas como uma conta de um cliente ou um serviço que o banco ofereça.

6. Convenções de nomenclatura: Camel, Pascal, Kebab e Snake case

Se você estuda programação, provavelmente se vê a todo momento nomeando coisas, como variáveis, classes, métodos, funções e muito mais. Mas, afinal, será que existe alguma convenção para ajudar nesse processo de nomeação?

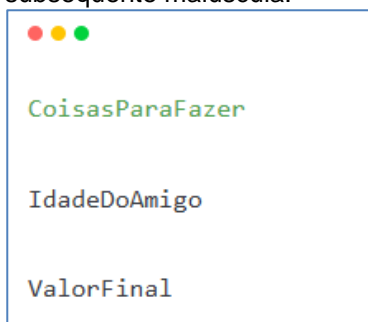
Há diversas maneiras de aplicar boas práticas no seu código, uma delas é a convenção de nomenclatura, que são recomendações para nomear identificadores ajudando a tornar seu código mais limpo e fácil de compreender, mantendo um padrão para o leitor e até mesmo podendo fornecer informações sobre a função do identificador.

Além disso, podemos encontrar outras linguagens, como Go, que dependem muito de você conhecer a diferença entre os casos de uso, pois a convenção utilizada no nome possui um efeito semântico também.

Agora, vamos conhecer os quatro tipos mais comuns de convenções de nomenclatura para combinar palavras em uma única *String*.

6.1. Camel Case

Camel case deve começar com a primeira letra minúscula e a primeira letra de cada nova palavra subsequente maiúscula:



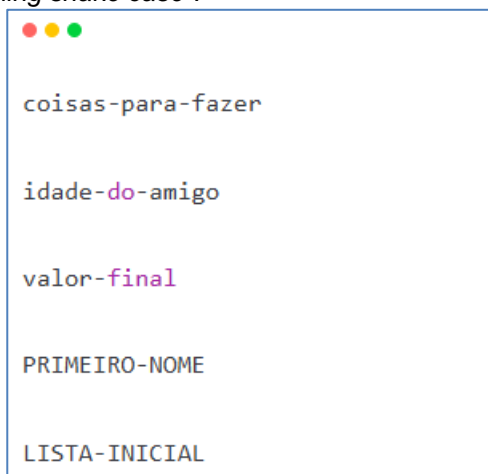
CoisasParaFazer

IdadeDoAmigo

ValorFinal

6.2. Snake case

Em snake case, conhecido também como “*underscore case*”, utilizamos *underline* no lugar do espaço para separar as palavras. Quando o *snake case* está em caixa alta, ele é chamado de “*screaming snake case*”:



coisas-para-fazer

idade-do-amigo

valor-final

PRIMEIRO-NOME

LISTA-INICIAL

6.3. Kebab case

Kebab case utiliza o traço para combinar as palavras. Quando o kebab case está em caixa alta, ele é chamado de “*screaming kebab case*”:



```
coisas-para-fazer
```

```
idade-do-amigo
```

```
valor-final
```

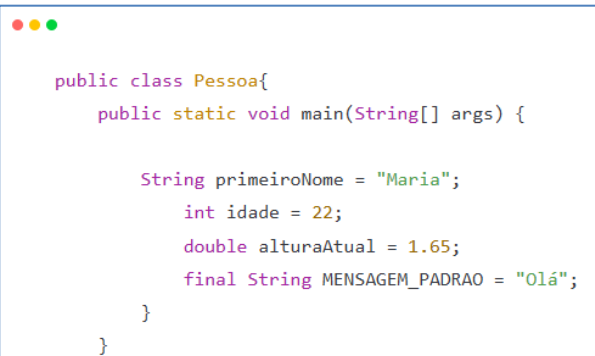
```
PRIMEIRO-NOME
```

```
LISTA-INICIAL
```

6.4. Convenções Java

O Java possui o *Java Secure Coding Guidelines*, uma documentação disponibilizada pela Oracle, baseada nos padrões de codificação apresentados no *Java Language Specification*. É importante ressaltar que essas convenções de código Java foram escritas em 1999 e não foram atualizadas desde então, portanto algumas informações contidas no documento podem não estar atualizadas. As convenções mais comuns são e que usamos:

- *camelCase* para variáveis, atributos e métodos;
- *PascalCase* para classes;
- *SCREAMING_SNAKE_CASE* para constantes.



```
public class Pessoa{  
    public static void main(String[] args) {  
  
        String primeiroNome = "Maria";  
        int idade = 22;  
        double alturaAtual = 1.65;  
        final String MENSAGEM_PADRAO = "Olá";  
    }  
}
```

7. Semântica e Sintaxe, vamos entender?

Se você respondeu sim! Parabéns!

Na vasta paisagem da programação, duas palavras - semântica e sintaxe - surgem frequentemente, mas nem sempre são entendidas em sua totalidade. A semântica e a sintaxe são como duas faces de uma mesma moeda, ambas essenciais para a compreensão e execução bem-sucedida de um código. No entanto, é fundamental entender a diferença entre elas e como elas se complementam para criar uma experiência de programação poderosa e eficaz.

7.1. O que é Semântica?

A semântica está relacionada ao significado ou interpretação do código. Em outras palavras, trata-se de entender o propósito e a função por trás das instruções escritas em um programa. Uma semântica forte garante que o código não apenas funcione corretamente, mas também faça sentido do ponto de vista lógico.

Por exemplo, ao escrever um código para calcular a média de uma lista de números, a semântica envolveria a compreensão de que estamos tentando encontrar um valor médio e que isso requer a soma de todos os números na lista, dividida pelo total de elementos na lista.

7.2. O que é Sintaxe?

A sintaxe, por outro lado, diz respeito à estrutura e à gramática do código. Ela se concentra em seguir as regras específicas da linguagem de programação para garantir que o código seja

escrito de forma correta e compreensível pelo computador. Uma sintaxe correta é fundamental para evitar erros de compilação ou interpretação.

Continuando com o exemplo anterior, a sintaxe seria a forma como escrevemos o código em uma linguagem de programação específica, seguindo suas regras gramaticais. Por exemplo, em Python, a sintaxe para calcular a média de uma lista de números pode envolver o uso de loops e operadores aritméticos de acordo com a estrutura definida pela linguagem.

7.3. O Equilíbrio entre Semântica e Sintaxe

Para escrever um código eficiente e de fácil manutenção, é essencial encontrar o equilíbrio certo entre semântica e sintaxe. Um código com uma semântica forte, mas com uma sintaxe confusa, pode ser difícil de entender e dar manutenção. Da mesma forma, um código com uma sintaxe impecável, mas com uma semântica fraca, pode não atender às necessidades do usuário final.

O verdadeiro poder da programação reside na capacidade de comunicar ideias complexas de forma clara e concisa. Isso requer não apenas um domínio da sintaxe da linguagem de programação escolhida, mas também uma compreensão profunda dos princípios subjacentes e da lógica por trás do problema a ser resolvido.

7.4. Como Melhorar a Semântica e a Sintaxe do seu Código

Compreender o Problema: Antes de começar a escrever código, certifique-se de entender completamente o problema que está tentando resolver. Isso ajudará a orientar tanto a semântica quanto a sintaxe do seu código.

Escolher a Linguagem Adequada: Cada linguagem de programação tem suas próprias nuances de semântica e sintaxe. Escolha a linguagem que melhor se alinha com os requisitos do seu projeto e com a sua própria compreensão.

Praticar Regularmente: A prática leva à perfeição. Dedique tempo para praticar a escrita de código em diferentes cenários para aprimorar tanto a semântica quanto a sintaxe.

Revisar e Refatorar: Sempre revise seu código em busca de possíveis melhorias na semântica e na sintaxe. A refatoração é uma parte essencial do processo de desenvolvimento de software e pode ajudar a tornar seu código mais claro e eficiente.

Em última análise, a semântica e a sintaxe são duas partes igualmente importantes do processo de programação. Ao compreender e aplicar esses conceitos de forma equilibrada, os programadores podem criar soluções poderosas e elegantes para uma ampla gama de problemas.

8. Linguagem Java

Iremos agora abordar nossas aulas uma das maiores linguagens e que demanda um conhecimento específico, por isso que iremos aplicar os conceitos vistos em **Orientação à Objetos** nesta linguagem.

Java é uma linguagem de programação orientada a objeto, multiplataforma que é executada em bilhões de dispositivos em todo o mundo. Ele capacita aplicativos, sistemas operacionais de smartphones, software empresarial e muitos programas conhecidos. Apesar de estar sendo usado há mais de 20 anos, Java é atualmente a linguagem de programação mais popular para desenvolvedores de aplicativos. Java é:

Multiplataforma: Java foi marcado com o slogan "escreva uma vez, execute em qualquer lugar", e isso ainda é válido hoje. O código de programação Java escrito para uma plataforma, como o sistema operacional Windows, pode ser facilmente transferido para outra plataforma, como um sistema operacional de um celular, e vice-versa, sem ser completamente reescrito. O Java funciona em várias plataformas porque quando um programa Java é compilado, o compilador cria um arquivo de código de bytes .class que pode ser executado em qualquer sistema operacional que tenha a Máquina Virtual Java (JVM) instalada nele. Normalmente, é fácil instalar o JVM na maioria dos principais sistemas operacionais, incluindo iOS, o que nem sempre foi o caso.

Orientado a objeto: Java estava entre as primeiras linguagens de programação orientadas a objeto. Uma linguagem de programação orientada a objeto organiza seu código em torno de classes e objetos, em vez de funções e comandos. A maioria das linguagens de programação modernas, incluindo C++, C#, Python e Ruby, é orientada a objeto.

8.1. Quando o Java foi criado?

Java foi inventado por James Gosling em 1995 enquanto ele estava trabalhando na Sun Microsystems. Embora tenha obtido popularidade rapidamente após seu lançamento, o Java não foi iniciado como a linguagem de programação *powerhouse* que é hoje.



James Gosling

O desenvolvimento do que viria a ser o Java começou na Sun Microsystems em 1991. O projeto, inicialmente chamado de Oak, foi originalmente projetado para televisão interativa. Quando o Oak foi considerado avançado demais para a tecnologia de cabo digital disponível na época, Gosling e sua equipe mudaram seu foco para a criação de uma linguagem de programação e renomearam o projeto como Java, em homenagem a um tipo de café da Indonésia. O Gosling viu o Java como uma chance de resolver problemas que ele previu que estavam a caminho de linguagens de programação menos portáteis à medida que mais dispositivos se tornavam conectados em rede.

O Java foi projetado com um estilo de sintaxe semelhante à linguagem de programação C++ para que já fosse familiar para os programadores quando eles comessem a usá-lo. Com o slogan "escreva uma vez, execute em qualquer lugar" em seu núcleo, um programador poderia escrever código Java para uma plataforma que seria executada em qualquer outra plataforma que tivesse um interpretador Java (ou seja, Máquina Virtual Java) instalado. Com o surgimento da Internet e a proliferação de novos dispositivos digitais em meados da década de 1990, o Java foi rapidamente adotado pelos desenvolvedores como uma linguagem de programação verdadeiramente multiplataforma.

A primeira versão pública do Java, Java 1.0, foi lançada em 1996. Em cinco anos, ele tinha 2,5 milhões de desenvolvedores em todo o mundo. Hoje, o Java capacita tudo, desde o sistema operacional móvel Android até software empresarial.

8.2. Para que a linguagem de programação Java é usada?

O Java é uma linguagem de programação extremamente transferível usada entre plataformas e diferentes tipos de dispositivos, de smartphones a TVs inteligentes. É usado para criar aplicativos móveis e Web, software empresarial, dispositivos de Internet das Coisas (IoT), jogos, Big Data, distribuídos e aplicativos baseados em nuvem entre outros tipos. Aqui estão alguns exemplos específicos do mundo real de aplicativos programados com o Java.

8.2.1. Aplicativos móveis

Muitos, se não a maioria dos aplicativos móveis são criados com Java. Java é a linguagem preferida dos desenvolvedores de aplicativos móveis devido à sua plataforma estável e versatilidade. Os aplicativos móveis populares codificados em Java incluem Spotify, Signal e Cash App.

8.2.2. Aplicativos Web

Uma ampla variedade de aplicativos Web é desenvolvida usando Java. O Twitter e o LinkedIn estão entre os mais conhecidos.

8.2.3. Software empresarial

O software empresarial destina-se a atender a um grande grupo ou organização. Inclui softwares como sistemas de cobrança e programas de gerenciamento da cadeia de fornecedores. A alta escalabilidade do Java o torna uma linguagem atraente para desenvolvedores que escrevem software empresarial.

8.2.4. Jogos

Os jogos populares escritos em linguagem de programação Java incluem o Minecraft original e o *RuneScape*.

8.2.5. Aplicativos de IoT

Os aplicativos de IoT estão em todos os lugares — TVs inteligentes, carros, máquinas pesadas, instalações de trabalho e muito mais — e o Java é usado para programar muitos deles. O Java é uma opção popular para desenvolvedores de IoT devido à facilidade com que seu código pode ser transferido entre plataformas.

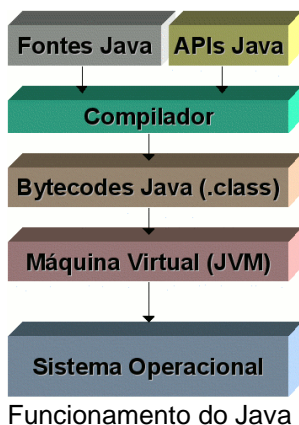
8.3. Como funciona o Java?

Conforme explicado anteriormente, o Java é uma linguagem de programação multiplataforma. Isso significa que ele pode ser escrito para um sistema operacional e executado em outro. Como isso é possível?

O código Java é escrito primeiro em um Kit de Desenvolvimento Java, que está disponível para Windows, Linux e macOS. Os programadores escrevem na linguagem de programação Java, que o kit traduz em código de computador que pode ser lido por qualquer dispositivo com o software certo. Isso é feito com um software chamado compilador. Um compilador usa um código de computador de alto nível como Java e o converte em uma linguagem que os sistemas operacionais entendem chamado código de bytes.

O código de bytes é então processado por um interpretador chamado Máquina Virtual Java (JVM). As JVMs estão disponíveis para a maioria das plataformas de software e hardware, e isso é o que permite que o código Java seja transferido de um dispositivo para outro. Para executar o Java, os JVMs carregam o código, verificam-no e fornecem um ambiente de *runtime*.

Dada a alta portabilidade do Java, não é de admirar que muitas pessoas queiram aprender como escrevê-lo. Felizmente, há muitos recursos disponíveis para começar a aprender Java.



9. Classes em Java

O conceito de classe apresentado anteriormente é genérico e pode ser aplicado em diversas linguagens de programação. Mostraremos como a classe Conta poderia ser escrita utilizando a linguagem Java. Inicialmente, discutiremos apenas sobre os atributos. Os métodos serão abordados posteriormente.

A classe Java Conta é declarada utilizando a palavra reservada **class**. No corpo dessa classe, são declaradas três variáveis que são os atributos que os objetos possuirão. Como a linguagem Java é estaticamente tipada, os tipos dos atributos são definidos no código. Os atributos saldo e limite são do tipo double, que permite armazenar números com casas decimais, e o atributo numero é do tipo int, que permite armazenar números inteiros.

Por convenção, os nomes das classes na linguagem Java devem seguir o padrão “Pascal Case” e “Camel Case” como vimos:

```
Conta.java x
Source History
1 package com.mycompany.exemplosparaaulas;
2
3 public class Conta {
4     double saldo;
5     double limite;
6     int numero;
7 }
```

10. Criando objetos em Java

Após definir a classe Conta, podemos criar objetos a partir dela. Esses objetos devem ser alocados na memória RAM do computador. Felizmente, todo o processo de alocação do objeto na memória é gerenciado pela máquina virtual. O gerenciamento da memória é um dos recursos mais importantes oferecidos pela máquina virtual.

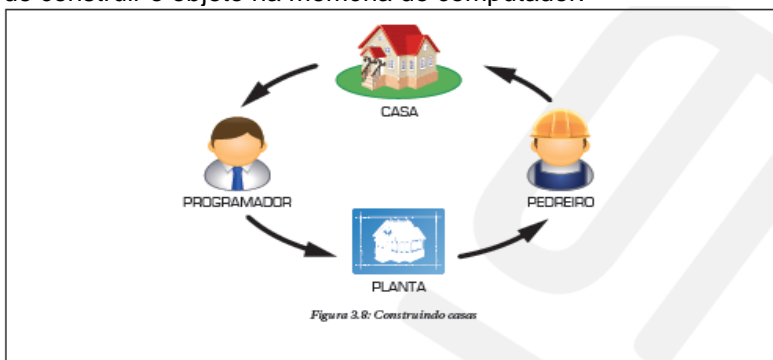
Do ponto de vista da aplicação, basta utilizar um comando especial para criar objetos e a máquina virtual se encarrega do resto. O comando para criar objetos é o **new**.

```
Conta.java x TestaConta.java x
Source History
1 package com.mycompany.exemplosparaaulas;
2
3 public class TestaConta {
4     public static void main(String[] args) {
5         // TODO code application logic here
6         new Conta();
7     }
8 }
```

A linha com o comando **new** poderia ser repetida cada vez que desejássemos criar (instanciar) um objeto da classe Conta. A classe TestaConta serve apenas para colocarmos o método **main**, que é o ponto de partida da aplicação.

```
Conta.java x TestaConta.java x
Source History
1 package com.mycompany.exemplosparaaulas;
2
3 public class TestaConta {
4     public static void main(String[] args) {
5         // TODO code application logic here
6         new Conta();
7         new Conta();
8         new Conta();
9     }
10 }
```

Chamar o comando **new** passando uma classe Java é como se estivéssemos contratando uma construtora passando a planta da casa que queremos construir. A construtora se encarrega de construir a casa para nós de acordo com a planta. Assim como a máquina virtual se encarrega de construir o objeto na memória do computador.

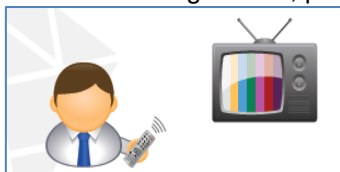


11. Referências

Todo objeto possui uma referência. A referência de um objeto é a única maneira de acessar os seus atributos e métodos. Dessa forma, devemos guardar as referências dos objetos que desejamos utilizar.

A princípio, podemos comparar a referência de um objeto com o endereço de memória desse objeto. De fato, essa comparação simplifica o aprendizado. Contudo, o conceito de

referência é mais amplo. Uma referência é o elemento que permite que um determinado objeto seja acessado. Uma referência está para um objeto assim como um controle remoto está para um aparelho de TV. Através do controle remoto de uma TV você pode aumentar o volume ou trocar de canal. Analogamente, podemos controlar um objeto através da referência do mesmo.



12. Referências em Java

Ao utilizar o comando `new`, um objeto é alocado em algum lugar da memória. Para que possamos acessar esse objeto, precisamos de sua referência. O comando `new` devolve a referência do objeto que foi criado.

Para guardar as referências devolvidas pelo comando `new`, devemos utilizar variáveis não primitivas.

```
1 package com.mycompany.exemplosparaaulas;
2
3 public class TestaConta {
4     public static void main(String[] args) {
5         // TODO code application logic here
6         Conta referencia = new Conta();
7     }
8 }
```

No código Java acima, a variável **referencia** receberá a referência do objeto criado pelo comando `new`. Essa variável é do tipo `Conta`. Isso significa que ela só pode armazenar referências de objetos do tipo `Conta`.

13. Manipulando Atributos

Podemos alterar ou acessar os valores guardados nos atributos de um objeto se tivermos a referência a esse objeto. Os atributos são acessados pelo nome. No caso específico da linguagem Java, a sintaxe para acessar um atributo utiliza o operador `"."`.

```
1 package com.mycompany.exemplosparaaulas;
2
3 public class TestaConta {
4     public static void main(String[] args) {
5         //Instância do objeto conta em variáveis não
6         //primitivas
7         Conta referencia = new Conta();
8
9         //Adicionar os valores aos atributos
10        referencia.saldo = 1000;
11        referencia.limite = 500;
12        referencia.numero = 1;
13
14        //Mostrar no terminal os valores
15        System.out.println(referencia.saldo);
16        System.out.println(referencia.limite);
17        System.out.println(referencia.numero);
18    }
19 }
```

Output - Run (TestaConta) x

```
Recompiling the module because of changed source code.
Compiling 3 source files with javac [debug release 22] to target\classes
--EXECUTING: java -cp target\classes com.mycompany.exemplosparaaulas.TestaConta
1000.0
500.0
1
BUILD SUCCESS
```


No código acima, o atributo saldo recebe o valor 1000. O atributo limite recebe o valor 500 e o numero recebe o valor 1. Depois, os valores são impressos na tela através da função `System.out.println()`.

14. Valores Padrão

Poderíamos instanciar um objeto e utilizar seus atributos sem inicializá-los explicitamente, pois os atributos são inicializados com valores padrão. Os atributos de tipos numéricos são inicializados com 0, os atributos do tipo *boolean* são inicializados com false e os demais atributos com *null* (referência vazia).

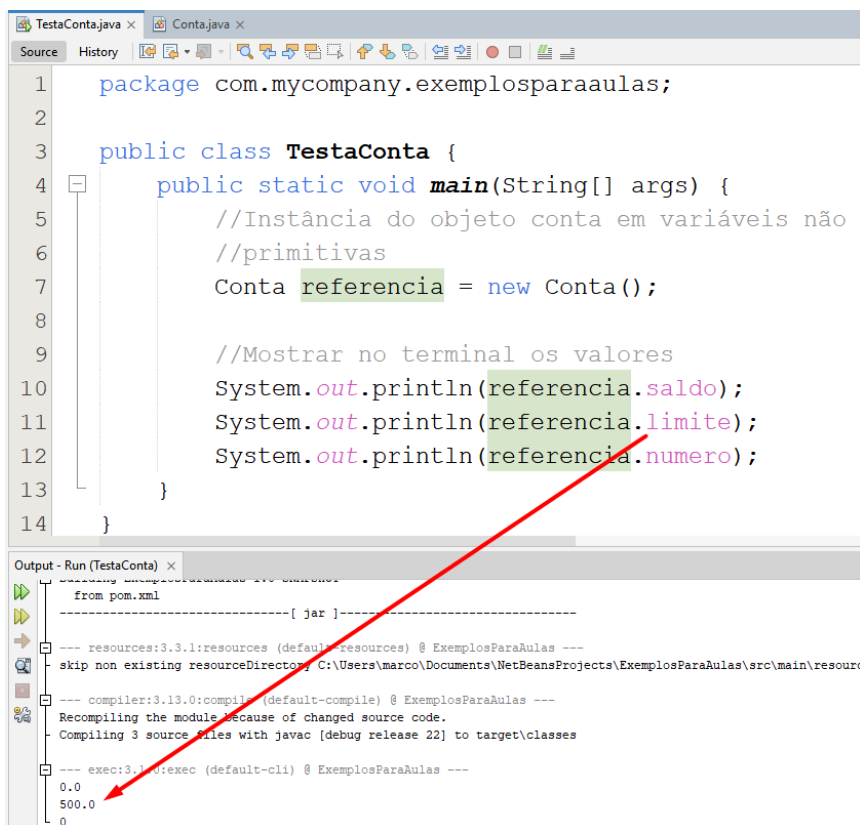
```
Conta.java x
Source History
1 package com.mycompany.exemplosparaaulas;
2
3 public class Conta {
4     double saldo;
5     double limite;
6     int numero;
7 }
```

```
TestaConta.java x
Source History
1 package com.mycompany.exemplosparaaulas;
2
3 public class TestaConta {
4     public static void main(String[] args) {
5         //Instância do objeto conta em variáveis não
6         //primitivas
7         Conta referencia = new Conta();
8
9         //Mostrar no terminal os valores
10        System.out.println(referencia.saldo);
11        System.out.println(referencia.limite);
12        System.out.println(referencia.numero);
13    }
14 }
```

```
Output - Run (TestaConta) x
from pom.xml
[ jar ]-----
--- resources:3.3.1:resources (default-resources) @ ExemplosParaAulas ---
skip non existing resourceDirectory C:\Users\marco\Documents\NetBeansProjects\ExemplosParaAulas\src\main\resou
--- compiler:3.13.0:compile (default-compile) @ ExemplosParaAulas ---
Recompiling the module because of changed source code.
Compiling 3 source files with javac [debug release 22] to target\classes
--- exec:3.1.0:exec (default-cli) @ ExemplosParaAulas ---
0.0
0.0
0
```

A inicialização dos atributos com os valores padrão ocorre na instanciação, ou seja, quando o comando `new` é utilizado. Dessa forma, todo objeto “nasce” com os valores padrão. Em alguns casos, é necessário trocar esses valores. Para trocar o valor padrão de um atributo, devemos inicializá-lo na declaração. Por exemplo, suponha que o limite padrão das contas de um banco seja R\$ 500. Nesse caso, seria interessante definir esse valor como padrão para o atributo `limite`.

```
TestaConta.java x Conta.java x
Source History
1 package com.mycompany.exemplosparaaulas;
2
3 public class Conta {
4     double saldo;
5     double limite = 500;
6     int numero;
7 }
```



```
1 package com.mycompany.exemplosparaaulas;
2
3 public class TestaConta {
4     public static void main(String[] args) {
5         //Instância do objeto conta em variáveis não
6         //primitivas
7         Conta referencia = new Conta();
8
9         //Mostrar no terminal os valores
10        System.out.println(referencia.saldo);
11        System.out.println(referencia.limite);
12        System.out.println(referencia.numero);
13    }
14 }
```

Output - Run (TestaConta) x

```
from pom.xml
[ jar ]-----
resources:3.3.1:resources (default-resources) @ ExemplosParaAulas ---
skip non existing resourceDirectory C:\Users\marco\Documents\NetBeansProjects\ExemplosParaAulas\src\main\resourc
compiler:3.13.0:compile (default-compile) @ ExemplosParaAulas ---
Recompiling the module because of changed source code.
Compiling 3 source files with javac [debug release 22] to target\classes
exec:3.13.0:exec (default-cli) @ ExemplosParaAulas ---
0.0
500.0
0
```

15. Algumas categorias de projetos que podemos trabalhar em Java

Podemos categorizar nossos projetos em Java e podemos falar sobre Java com Maven, Java com Gradle e Java com Ant, como algumas categorias que aparecem quando criamos um projeto novo no Apache Netbeans por exemplo, nestes casos, estamos tratando de ferramentas de *build automation* (automação de compilação) que são usadas para compilar, empacotar, testar, e implantar aplicativos Java de forma eficiente. Cada uma dessas ferramentas tem uma abordagem distinta, mas todas têm o objetivo de facilitar o processo de desenvolvimento e tornar a construção de projetos mais ágil e menos propensa a erros, para nossas aulas e práticas, a ideia é utilizarmos a Maven ou Ant. Vamos ver cada uma delas em detalhes:

15.1. Java com Maven

Maven é uma das ferramentas de build mais populares no ecossistema Java. Ele usa um modelo baseado em XML para descrever o ciclo de vida de construção do projeto.

15.1.1. Principais Características:

- **Dependências:** Maven facilita o gerenciamento de dependências. Ele se conecta a repositórios (como o Maven Central) e baixa automaticamente as bibliotecas necessárias para o seu projeto. Isso significa que você não precisa gerenciar manualmente os arquivos JAR e suas versões.
- **Arquitetura baseada em Plugins:** O Maven executa tarefas por meio de plugins (como compilação, testes e empacotamento) que são configurados no arquivo pom.xml (*Project Object Model*).
- **Ciclo de Vida:** O Maven segue um ciclo de vida definido, que inclui fases como *compile*, *test*, *package*, *install*, *deploy*. Cada uma dessas fases pode ser configurada para realizar tarefas específicas.

15.1.2. Vantagens do Maven:

- Gerenciamento de dependências automático.
- Repositórios centralizados (Maven Central).
- Integração fácil com servidores de build e CI/CD.

- Documentação e convenção forte (se você seguir as convenções do Maven, o projeto é fácil de entender).

15.1.3. Desvantagens:

- Complexidade em grandes projetos.
- XML pode ser verboso.

15.2. Java com Gradle

Gradle é uma ferramenta de automação de build moderna que tem ganhado popularidade nos últimos anos, principalmente por ser mais flexível e rápida que o Maven.

15.2.1. Principais Características:

- **Baseado em Groovy ou Kotlin:** Gradle usa uma linguagem de script (Groovy ou Kotlin DSL) para descrever o build, o que torna a configuração mais flexível e programática.
- **Desempenho:** Gradle é conhecido por ser mais rápido que o Maven e o Ant devido ao seu build incremental (onde ele apenas recompila partes do código que foram alteradas).
- **Gerenciamento de Dependências:** Assim como o Maven, Gradle também oferece fácil integração com repositórios (como Maven Central), mas com maior flexibilidade.
- **Configuração:** Gradle permite configurações mais complexas e personalizadas, sendo ideal para projetos mais complexos.

15.2.2. Vantagens do Gradle:

- Desempenho superior (builds incrementais e paralelismo).
- Flexibilidade: Configurações personalizadas e lógica programática.
- Suporte a múltiplas linguagens: Além de Java, Gradle pode ser usado com outros tipos de projeto (Android, Groovy, Scala, etc.).
- DSL mais conciso: Arquivos de configuração mais simples e expressivos que o Maven (usando Groovy ou Kotlin).

15.2.3. Desvantagens:

- Curva de aprendizado: A flexibilidade de Gradle pode tornar o processo de configuração um pouco mais complexo para iniciantes.
- Menos documentação tradicional (se comparado ao Maven, por exemplo).

15.3. Java com Ant

Ant é uma das ferramentas de build mais antigas para Java. Ao contrário do Maven e do Gradle, o Ant não segue uma convenção padrão para construir o projeto e não tem uma definição de ciclo de vida, o que significa que você tem que definir manualmente como o projeto deve ser construído.

15.3.1. Principais Características:

- **Arquivo build.xml:** O Ant usa um arquivo XML chamado build.xml para definir as tarefas de construção, como compilar código, empacotar JARs, executar testes e mais.
- **Sem Convenção:** Diferente do Maven, o Ant não segue convenções. O desenvolvedor tem total liberdade para definir como o build deve ocorrer.
- **Extensível:** Ant permite adicionar novos tipos de tarefas personalizadas com facilidade.

15.3.2. Vantagens do Ant:

- Extremamente flexível: Você tem controle total sobre o processo de build.
- Pode ser usado em projetos não-Java: Ant pode ser usado para outras linguagens também.
- Facilidade em adicionar tarefas personalizadas.

15.3.3. Desvantagens:

- Não tem ciclo de vida predefinido: O desenvolvedor deve definir manualmente como o processo de construção será.
- Mais difícil de gerenciar dependências: Ao contrário do Maven e Gradle, você precisa gerenciar dependências manualmente ou com plugins adicionais.
- Configuração verbosa: Arquivos build.xml podem ser grandes e difíceis de manter à medida que o projeto cresce.

15.4. Comparação resumida

Característica	Maven	Gradle	Ant
Configuração	Baseado em XML (pom.xml)	Baseado em Groovy ou Kotlin (build.gradle)	Baseado em XML (build.xml)
Gerenciamento de dependências	Sim, integrado com repositórios Maven	Sim, mas mais flexível que o Maven	Necessário configuração manual
Desempenho	Médio (builds não incrementais)	Rápido (builds incrementais e paralelismo)	Médio, depende de como é configurado
Facilidade de uso	Fácil para iniciantes, com convenções fortes	Maior curva de aprendizado, mas flexível	Flexível, mas sem ciclo de vida predefinido
Popularidade	Muito popular, especialmente em grandes empresas	Crescendo rapidamente, especialmente para Android	Menos usado atualmente, mas ainda popular em projetos antigos

Tabela de comparação entre categorias