

```
> javac Estruturas_De_Dados.java
```

Fundamentos de POO e Java para Estruturas de Dados

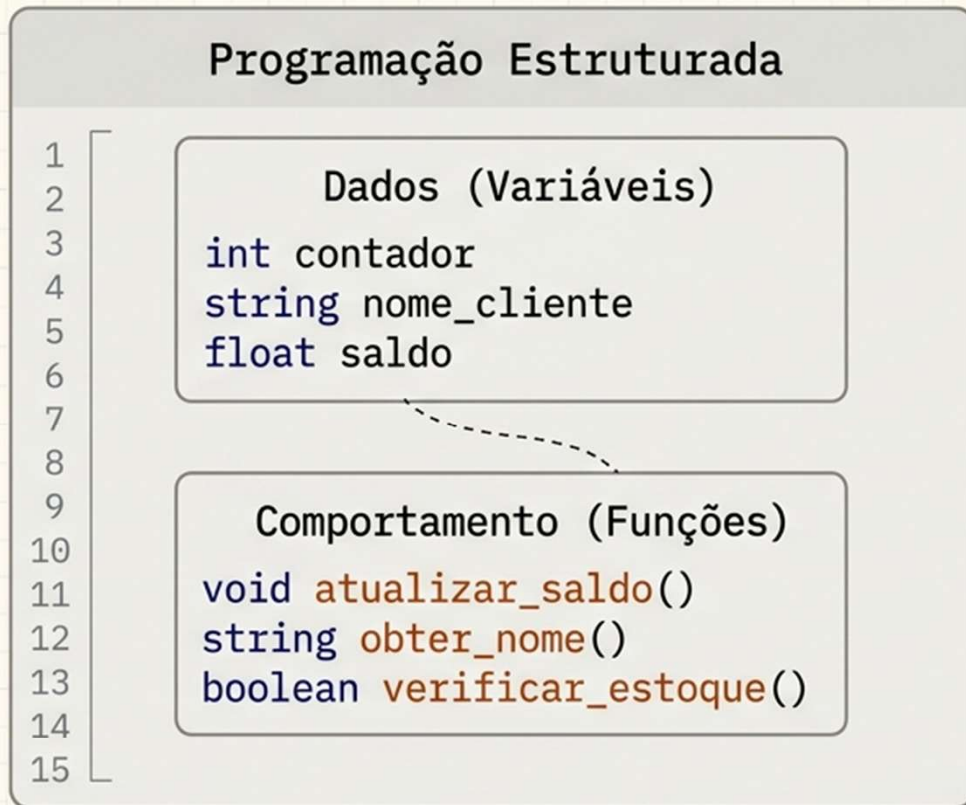
Do **mundo real** para a JVM: Um **guia visual e prático** para
estudantes de **Engenharia** e **Ciência da Computação**.

```
{ }
```

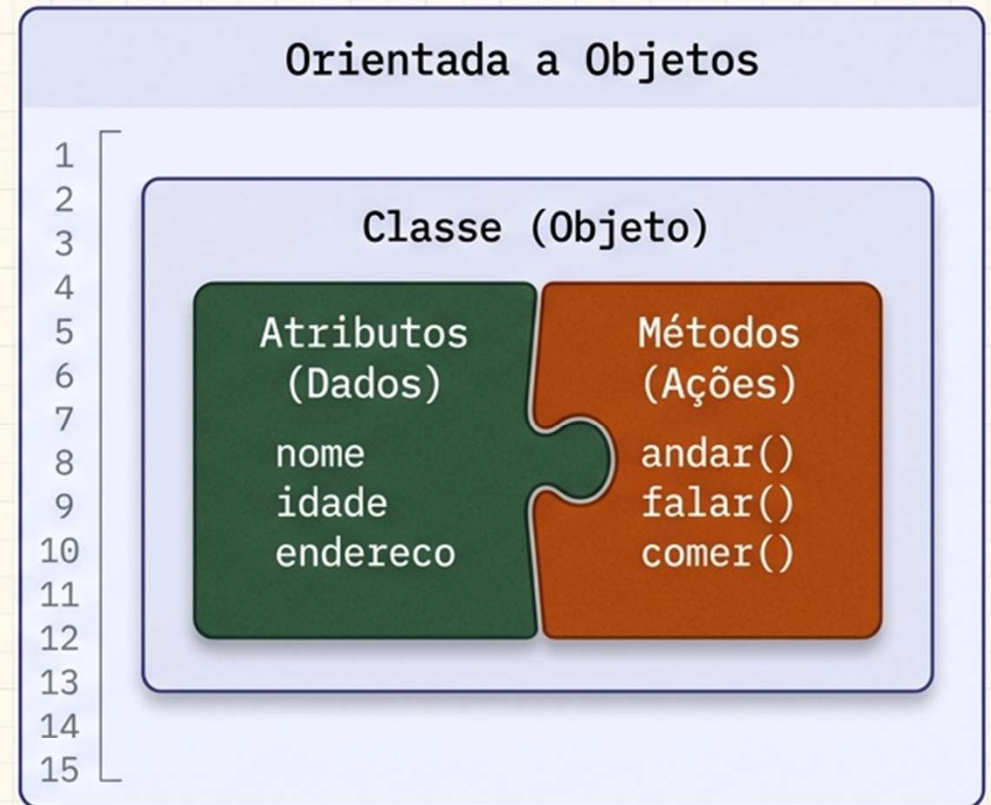
```
}
```

Baseado no material do Prof. Rafael Muniz (BRAESTD).

O Paradigma: Estruturada vs. Orientada a Objetos



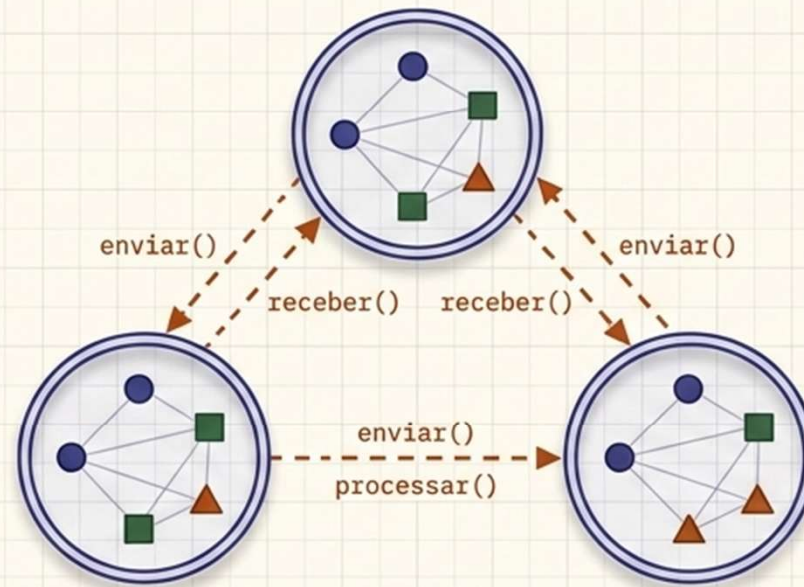
Dados e funções separados. O programador faz a ponte.



Dados e função unidos em uma classe.

Objetivo: Trazer o desenvolvimento de software mais próximo do mundo real.

A Inspiração Biológica (Alan Kay, Prêmio Turing 2003)



“O computador ideal deveria funcionar como um organismo vivo... cada ‘**célula**’ comportar-se-ia relacionando-se com outras células a fim de alcançar um objetivo, entretanto, funcionando de forma **autônoma**.”

(GOOGLE,2026)

*Takeaway: **Objetos** são entidades independentes que se comunicam através de **mensagens**.*

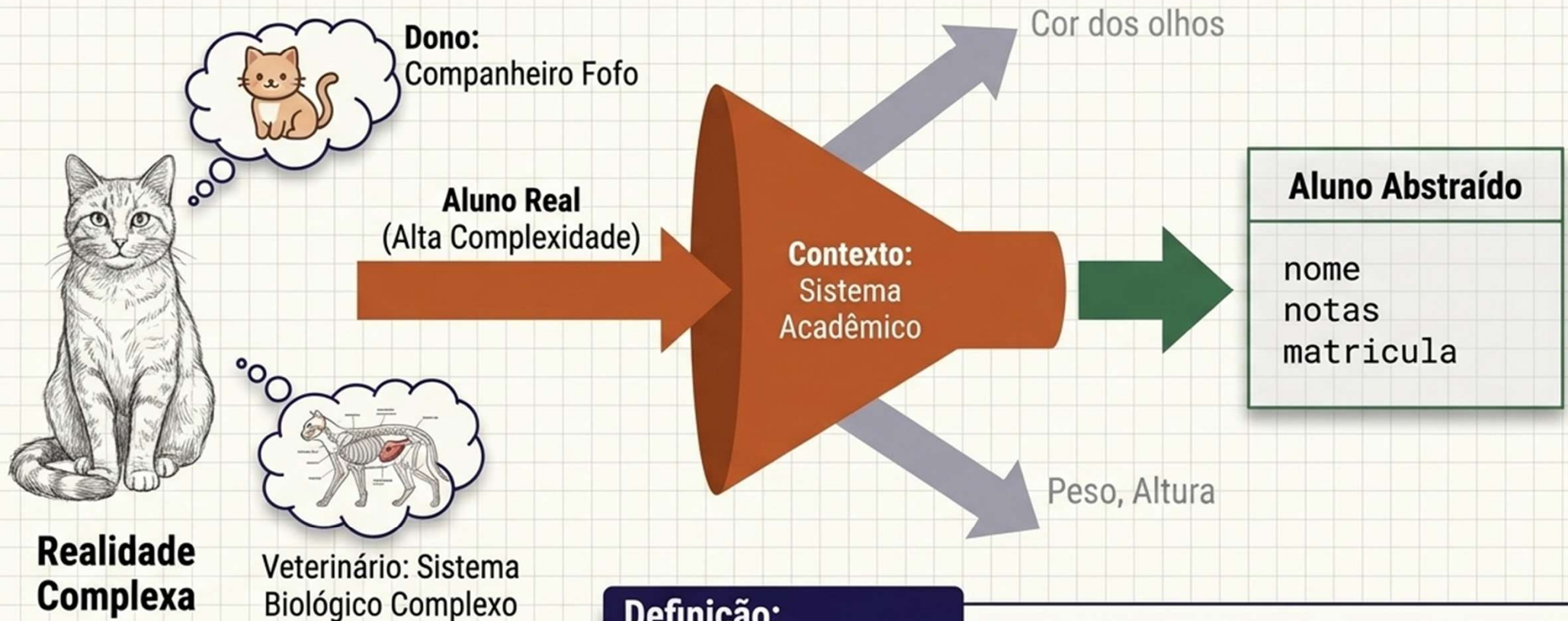
Os Pilares da POO



Os Fundamentos da Orientação a Objetos.

Hoje, focaremos na base (**Abstração**) e no topo (**Classes** e **Objetos**) para construir nossas estruturas de dados.

Abstração na Prática: Filtrando a Realidade

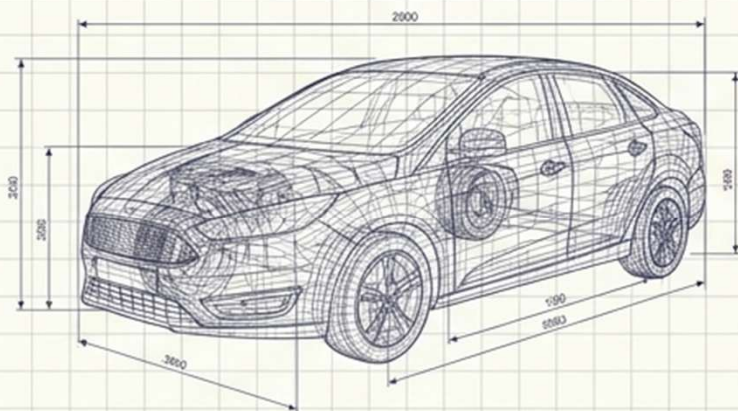


Definição:

Processo de focar apenas no que é **necessário e relevante** para o sistema, desconsiderando outras características.

Classes e Objetos: A Planta e a Construção

A PLANTA
(Classe)



Define o conceito abstrato.

Atributos: Marca, Ano, Cor.

A CONSTRUÇÃO
(Objetos)

Instanciação



Objeto A
(Ford, 2020, Prata)

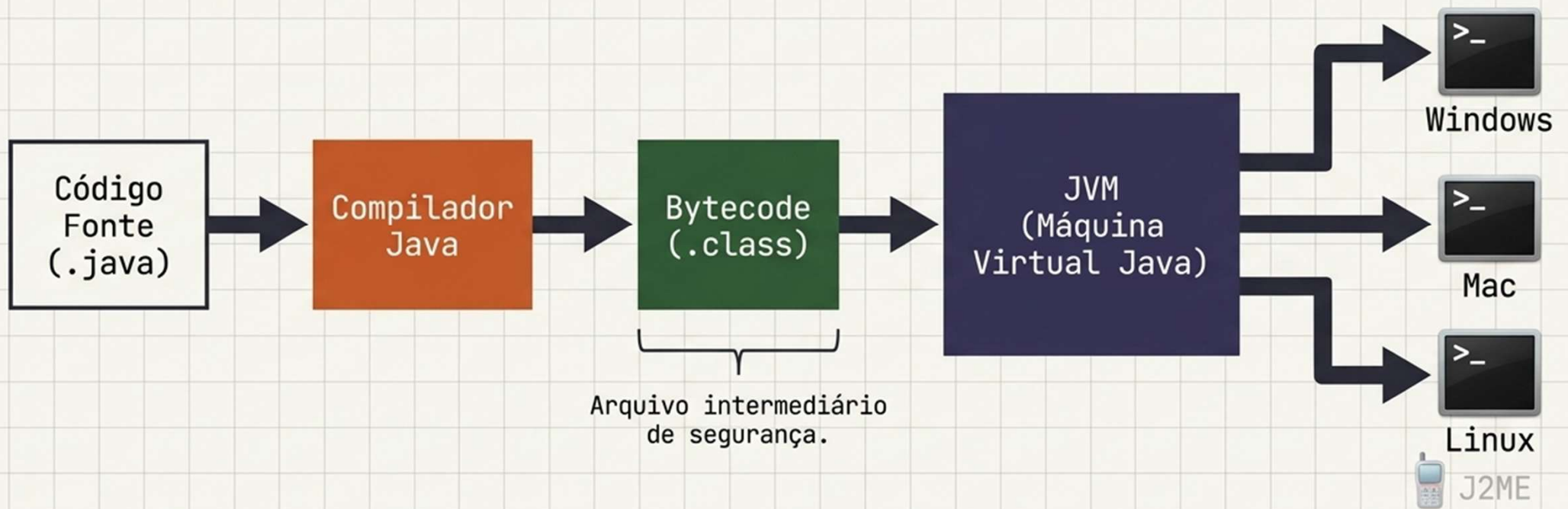


Objeto B
(Fiat, 2015, Vermelho)

A classe é a **abstração do mundo real**. Os **objetos** são as **instâncias reais e independentes** dessa classe.

0 Motor Java:

Escreva uma vez, rode em qualquer lugar



Em geral, programas em C são compilados para código nativo específico da plataforma, enquanto programas Java são compilados para bytecode, executado pela JVM.

Regras de Sintaxe e Boas Práticas

Atenção: O Java é **Case Sensitive** (diferencia maiúsculas de minúsculas).

	EVITE (Erro/Má Prática)	USE (Padrão Oficial)
1.	Nomes Significativos <code>String n;</code> Impossível de dar manutenção.	<code>String nomePaciente;</code> Fácil leitura.
2.	Padrão de Escrita <code>snake_case_proibido</code>	<code>camelCase</code> A primeira letra minúscula, novas palavras em maiúscula.
3.	Classes vs Variáveis <code>NomeDaClasse</code>	<code>nomeDaVariavel</code>

[Anatomia de uma Classe Java]

```
1 package sistema
2
3 public class Aluno {
4     String nome;
5     int idade;
6     double notaFinal;
7     boolean matriculado;
8 }
```

Pacote lógico (o Java cria automaticamente).

Atributos da classe.
Mistura de tipos primitivos (int, double, boolean) e classes embutidas (String).

O nome da classe DEVE ser idêntico ao nome do arquivo .java. Note o 'A' maiúsculo.

[O Ponto de Entrada: O Método Main]

Linguagem C (O Passado)

```
int main() {  
    printf("Hello World");  
    return 0;  
}
```

O ponto de partida. É aqui que a JVM começa a executar o seu programa.

Linguagem Java (O Presente)

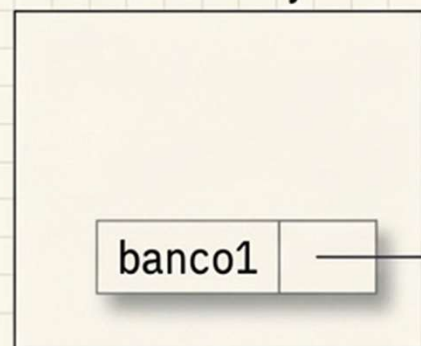
```
1 public static void main(String[] args) {  
2     System.out.println("Hello World!");  
3 }
```

Nosso novo printf. Exige exatidão sintática e ponto-e-vírgula obrigatório.

[A Criação: Entendendo o Operador new]

```
1 Banco banco1 = new Banco();
```

Stack Memory

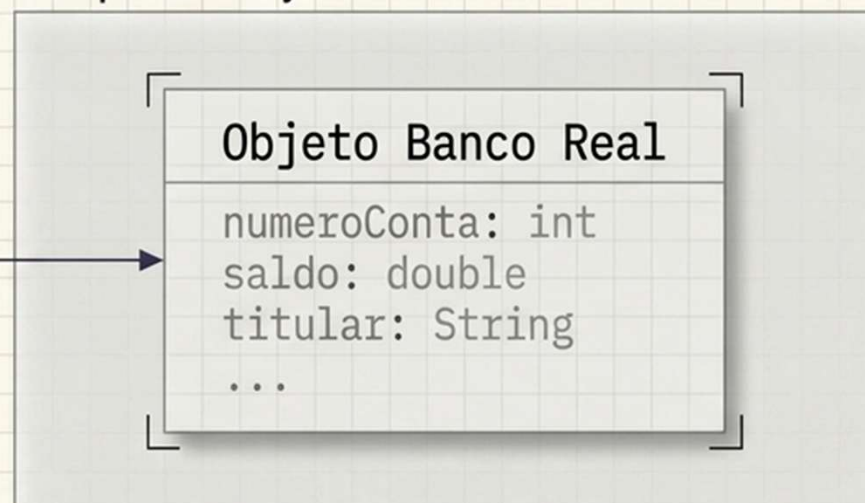


Apenas um
ponteiro/referência.
O objeto ainda não
existe fisicamente.

new

A palavra mágica
que aloca a
memória.

Heap Memory



O objeto físico recém-construído.

Sem o **new**, você tem apenas o mapa. Com o **new**, você constrói a casa. O ato de criar é chamado de INSTANCIAÇÃO.

[Construtores: O Nascimento do Objeto]

O construtor é executado uma vez para cada objeto criado.

Se você não criar um, o Java fornece um vazio de graça.

Opção A: O Padrão Oculto

```
1 public Banco() {  
2     // Gerado automaticamente se nenhum outro existir  
3     }  
    uma vez para cada objeto criado
```



Opção B: Injeção de Dados na Criação

```
1 public Banco(String nome) {  
2     this.nome = nome;  
3 }
```

Crucial: O prefixo 'this.' garante que estamos salvando o dado no atributo da CLASSE, e não na variável local recebida no parâmetro. Ele resolve a ambiguidade de nomes.

Sintetizando: Do Mundo Real para a JVM

Arquivo: Produto.java (A Planta)

```
1 public class Produto {  
2     String nome;  
3  
4     public Produto(String nome) {  
5         this.nome = nome;  
6     }  
7 }
```

Arquivo: Main.java (A Execução)

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Produto p1 = new Produto("Projektor");  
4     }  
5 }
```



O fluxo exato de execução
no momento da instanciação.

Checklist do Desenvolvedor de Estruturas de Dados

- ☒ **Tudo é Objeto:** Mapeou seu nó/lista conceitual como uma classe isolada?
- ☒ **Abstração Focada:** Removeu atributos inúteis para manter apenas o necessário?
- ☒ **Sintaxe Padrão:** Respeitou o camelCase e as letras maiúsculas em NomesDeClasse?
- ☒ **Nomes Significativos:** Abandonou o 'String n' pelo 'String nomePaciente'?
- ☒ **Operador new:** Lembrou de instanciar fisicamente o objeto na memória antes de usar?

```
> Sistema inicializado.  
Boa codificação na disciplina BRAESTD!
```