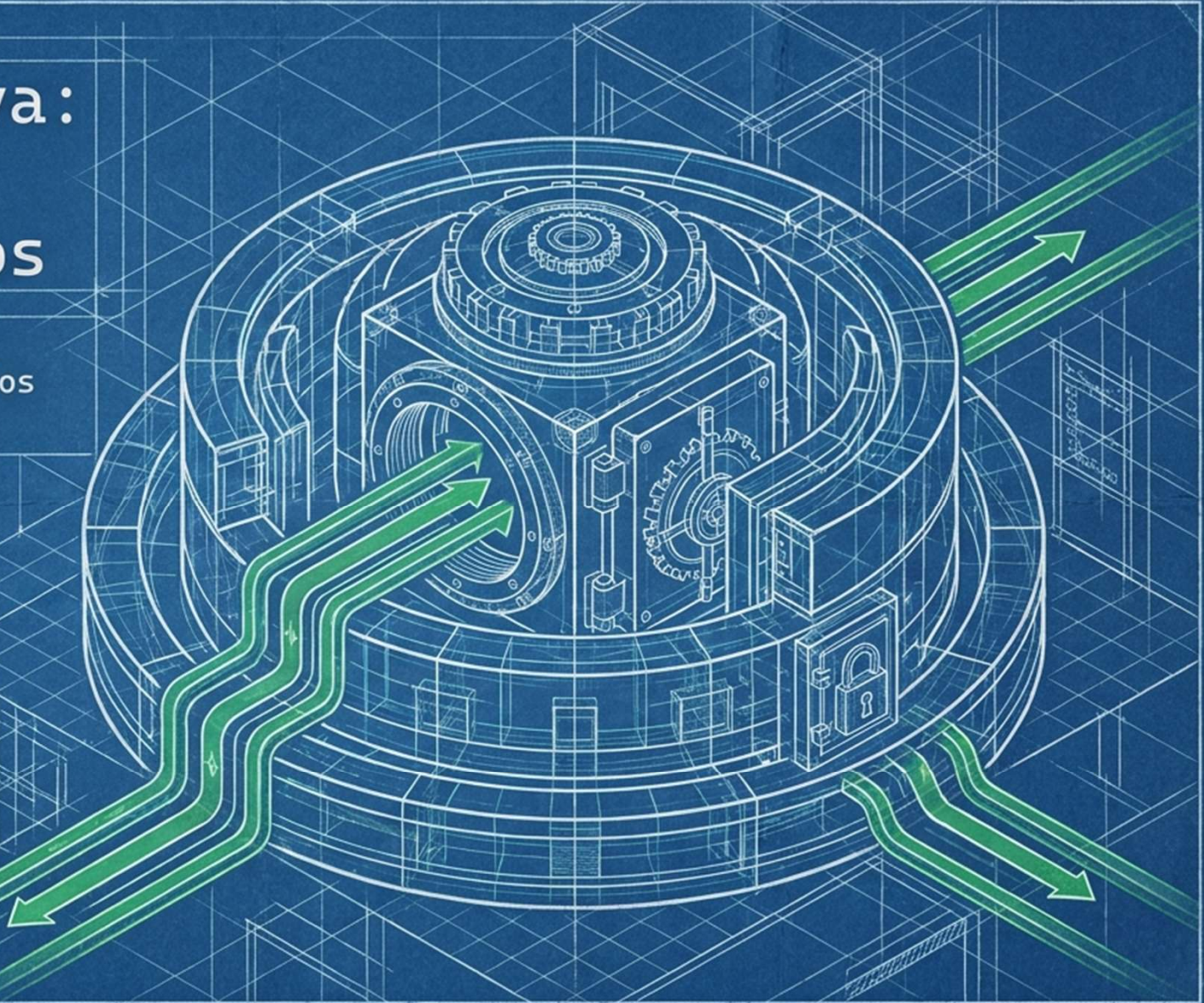


Arquitetura Java: A Anatomia de Objetos Robustos

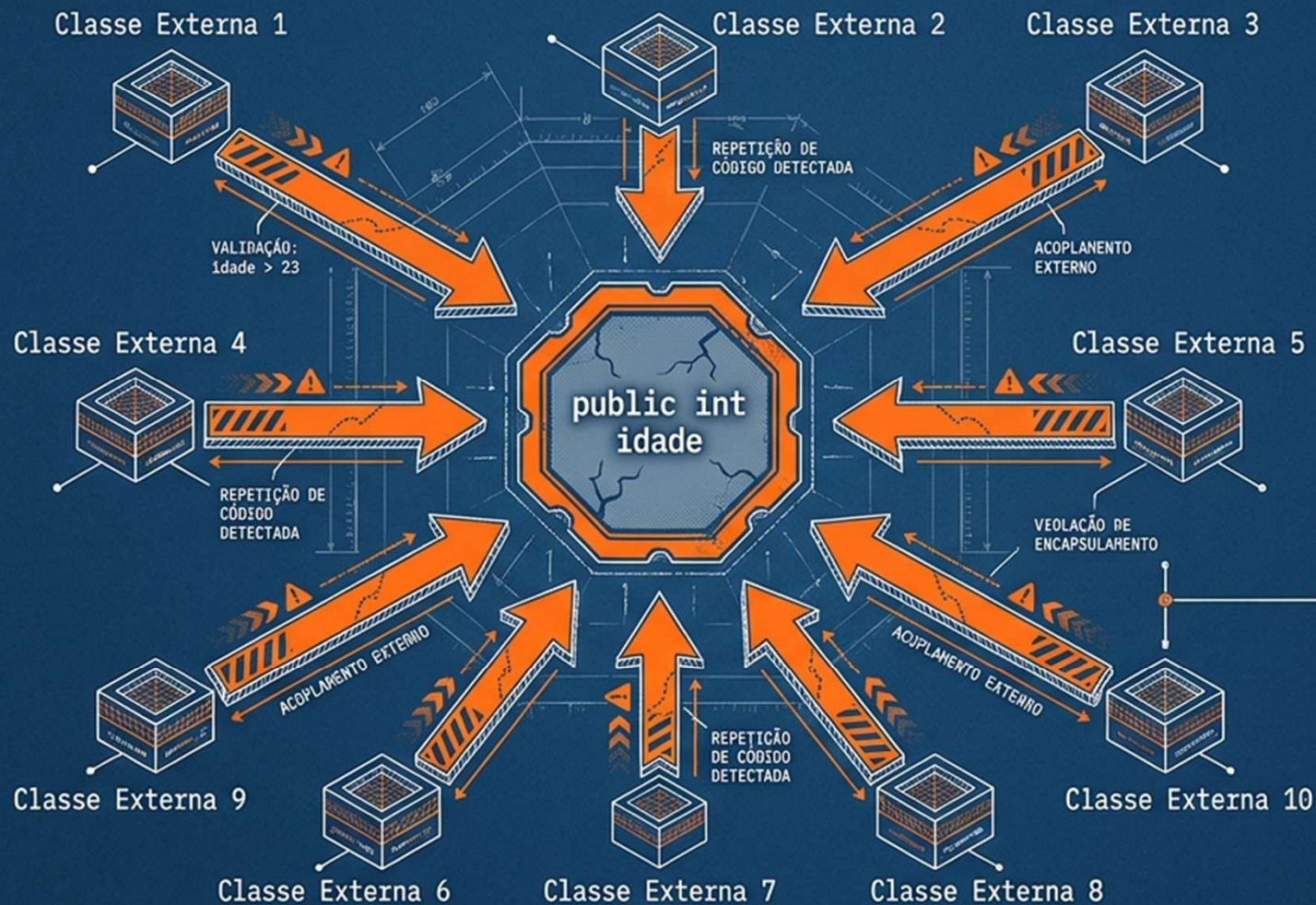
Do encapsulamento à escala:
dominando modificadores, métodos
e associações seguras.



Gerado com a IA Gemini



O Perigo da Baixa Coesão



Se 10 classes precisam validar a regra de que a idade > 23, o mesmo trecho de código precisará ser repetido 10 vezes.

A sua arquitetura falhou. A responsabilidade pelas regras de um dado deve pertencer estritamente ao próprio objeto.

Matriz de Modificadores de Acesso

		Mesma Classe	Mesmo Pacote / Herança	Qualquer Classe
	public (+): Acesso total. Visível dentro e fora da classe, sem restrições.			
	protected (#): Acesso restrito. Visível para a classe, pacote e classes herdeiras/derivadas.			
	default (): Acesso de pacote (padrão quando não declarado).			
	private (-): Isolamento total. Acessado apenas pelos métodos da própria classe.			

 **Padrão Ouro para Atributos**

A Anatomia de um Método (Comportamento)

```
public double calculaSoma(double a, double b) {  
    ...  
    return c;  
}
```

[modificador]: Define quem pode acionar este comportamento.

<tipo_retorno>: O que o método devolve após a execução. Pode ser void se não retornar nada.

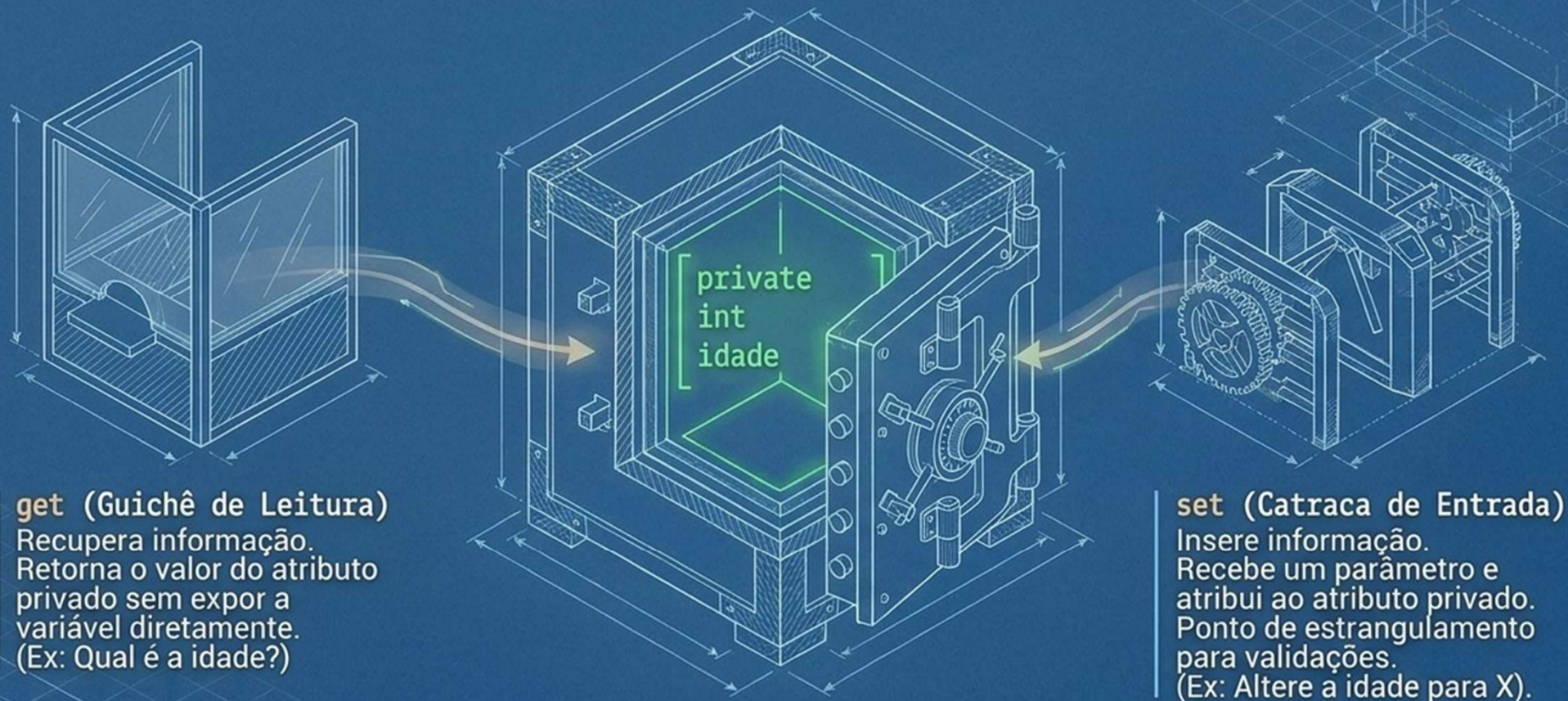
nome_metodo: A assinatura da ação.

([argumentos]): Os dados injetados necessários para o comportamento.

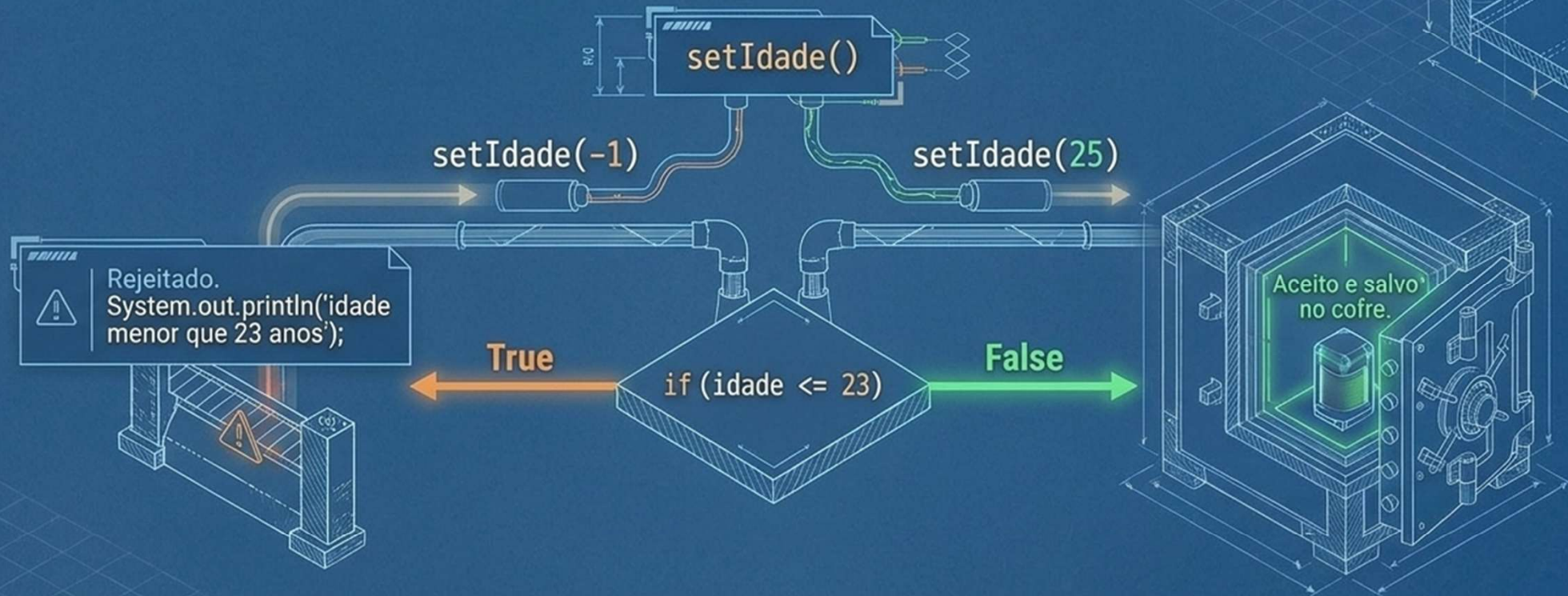


Nota UML: Na representação, os métodos ocupam o último retângulo da classe.

Getters e Setters como Porteiros da Classe

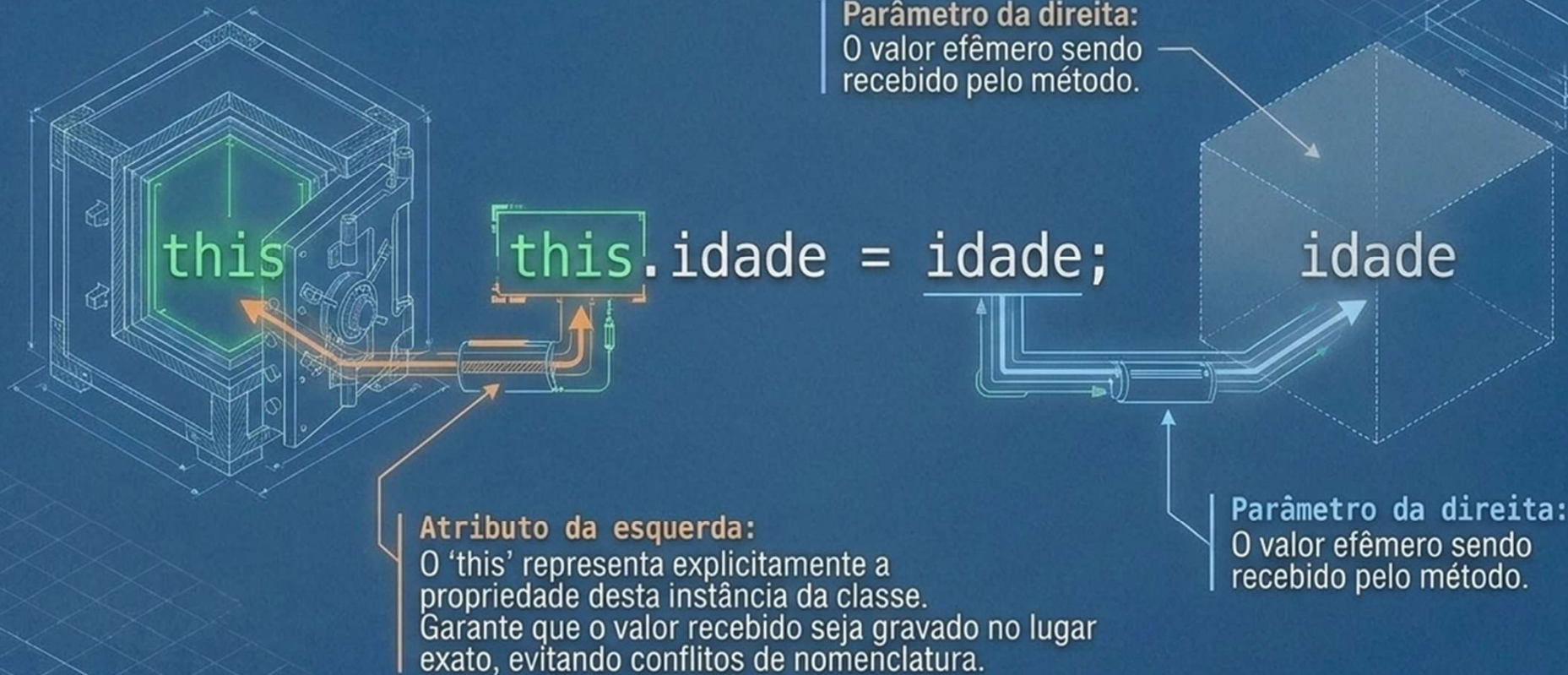


Injetando Regras de Negócio

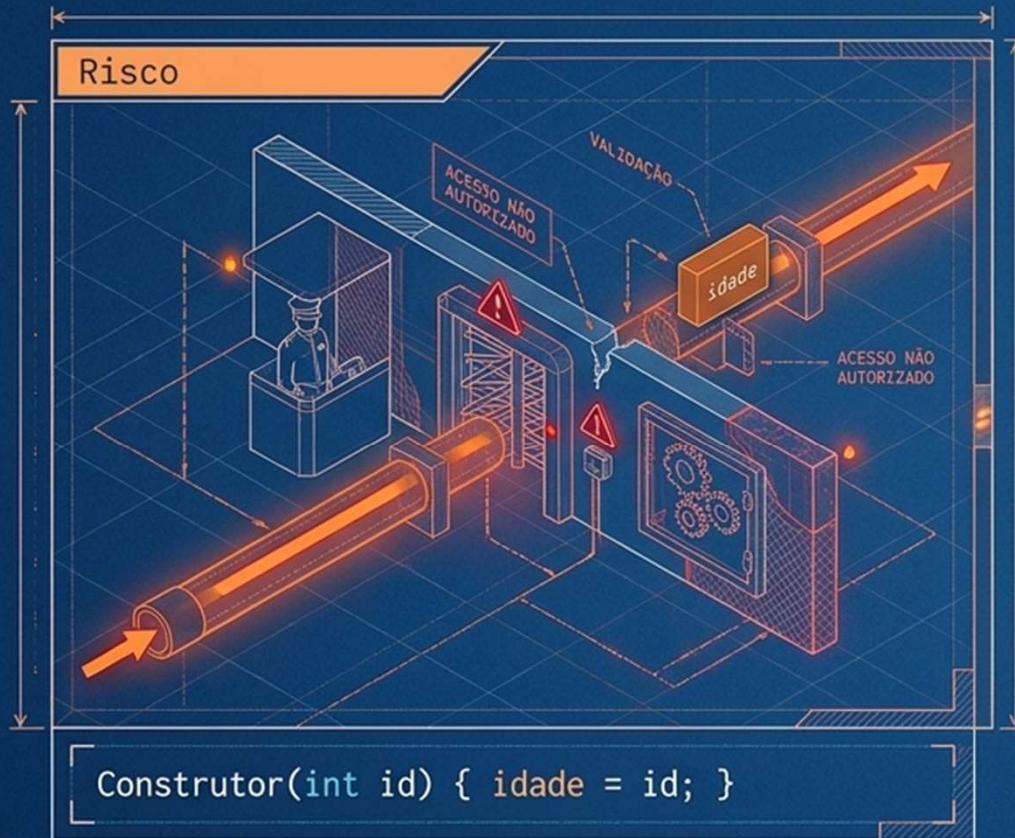


Com a regra centralizada no Setter, 1.000 classes diferentes podem chamar o método, e todas passarão pela mesma barreira de segurança. Alta coesão atingida.

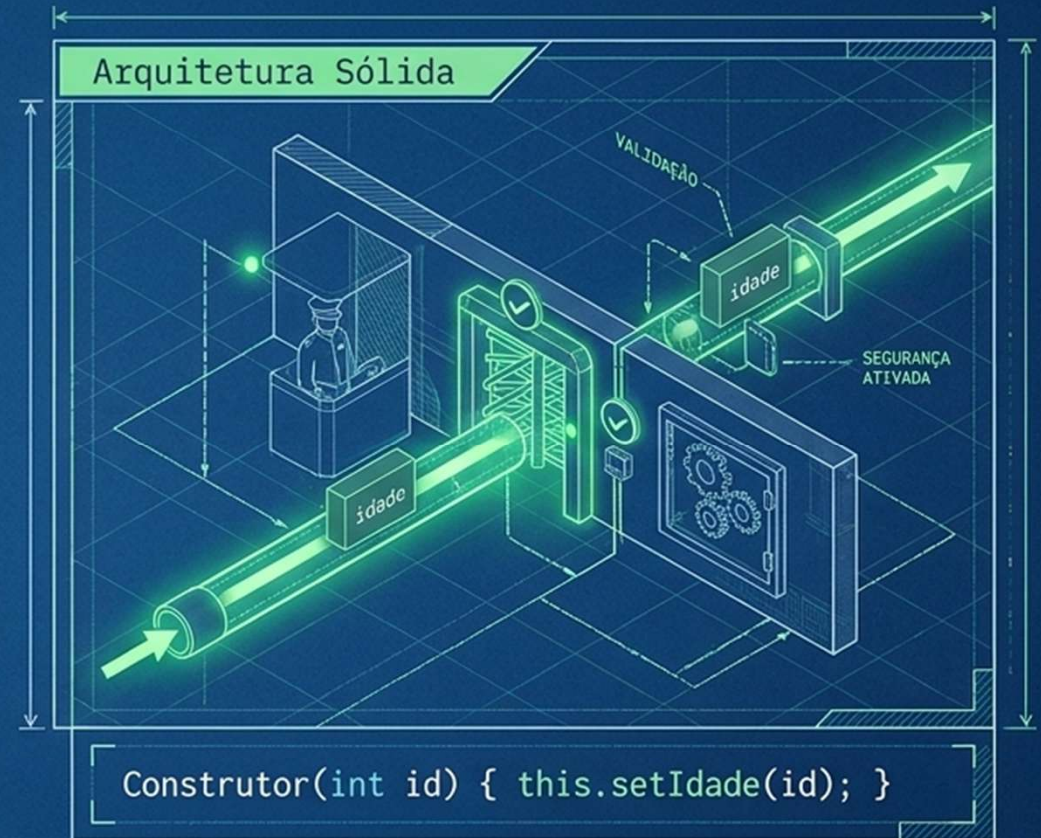
Eliminando Ambiguidades com this



O Construtor Seguro (Evitando Contrabando de Dados)

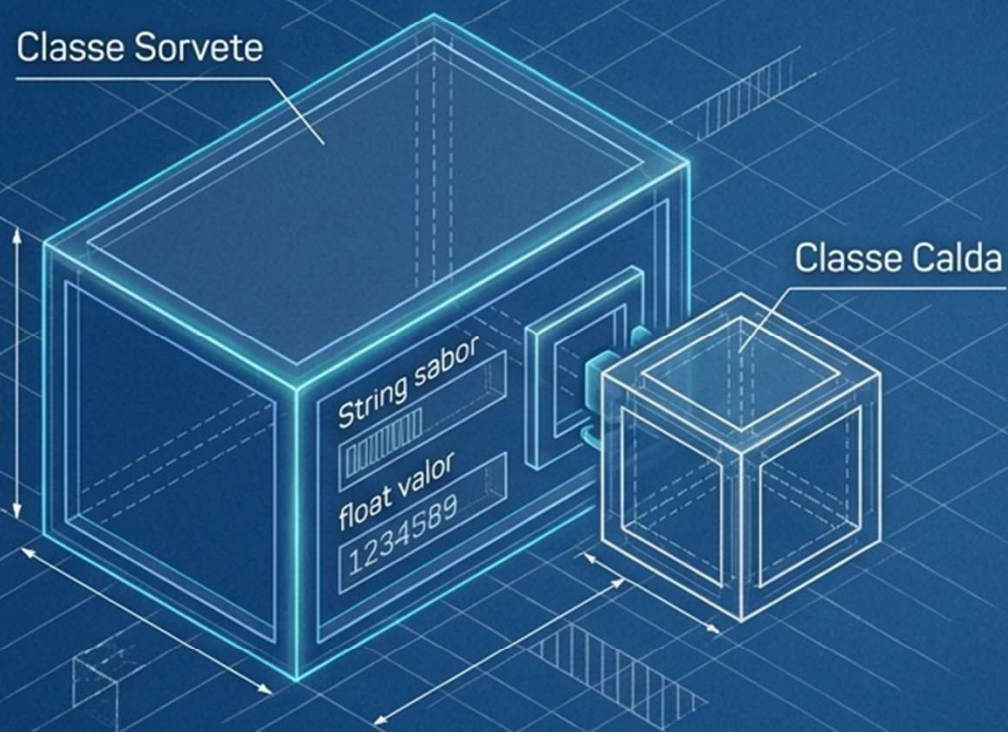


Problema: Inicializar o atributo diretamente no construtor burla a regra de validação ($idade > 23$). Ele aceitaria valores inválidos na criação do objeto.



Solução: O construtor invoca o próprio método Set. A regra de negócio é respeitada incondicionalmente desde o exato momento do nascimento do objeto.

Construindo Ecossistemas (Associação 1:1)

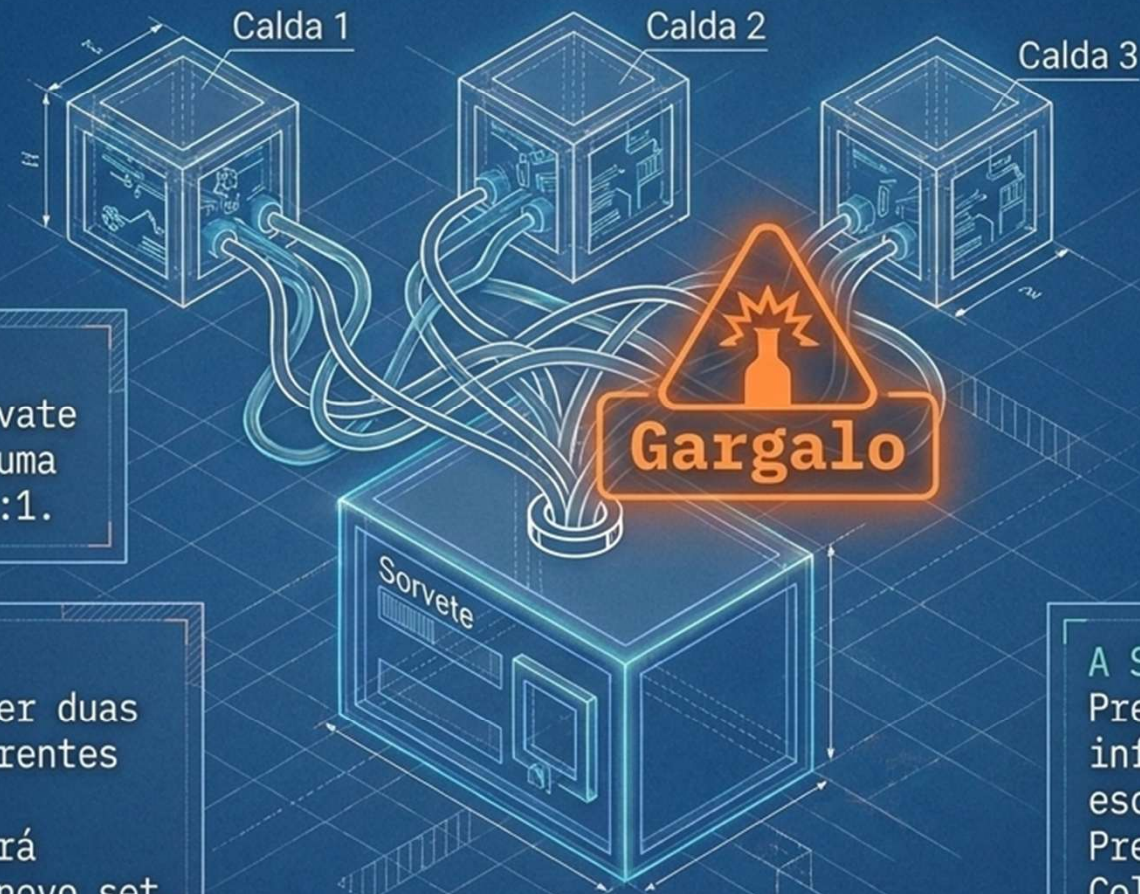


Uma nova classe criada representa um **novo tipo de dado**. Para criar uma associação, basta declarar um objeto de uma classe como atributo de outra.

```
public class Sorvete {  
    private String sabor;  
    private float valor;  
    private Calda calda; // Associação Criada  
}
```

No programa principal, instanciamos o Sorvete, instanciamos a Calda, e unimos os dois através do método setCalda().

O Limite do Relacionamento Único



O Dilema:
O design atual (private Calda calda;) cria uma ligação rígida de 1:1.

A Pergunta:
E se o usuário quiser duas ou três caldas diferentes no mesmo sorvete?
O atributo único será sobrescrito a cada novo set.

A Solução:
Precisamos de uma infraestrutura que escale dinamicamente.
Precisamos do Java Collections.

O Motor de Escala: ArrayList

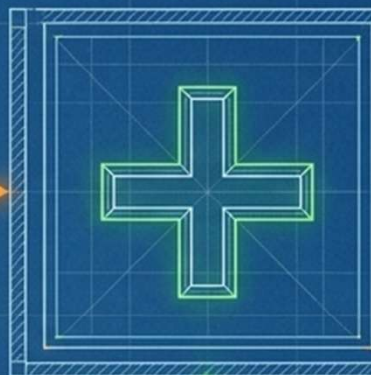
O ArrayList permite agrupar múltiplos objetos em uma lista redimensionável.

motor



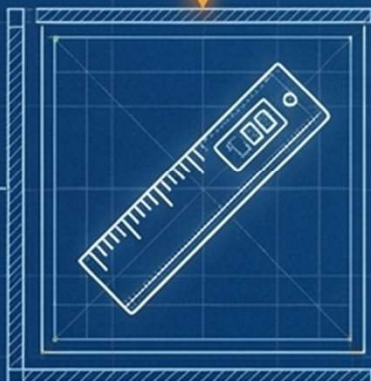
Instanciação:

```
ArrayList lista = new ArrayList();  
(Inicia o motor)
```



Inserção:

```
lista.add(obj);  
(Adiciona um novo item ao final)
```



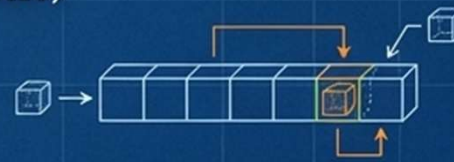
Tamanho:

```
lista.size();  
(Retorna a capacidade atual)
```



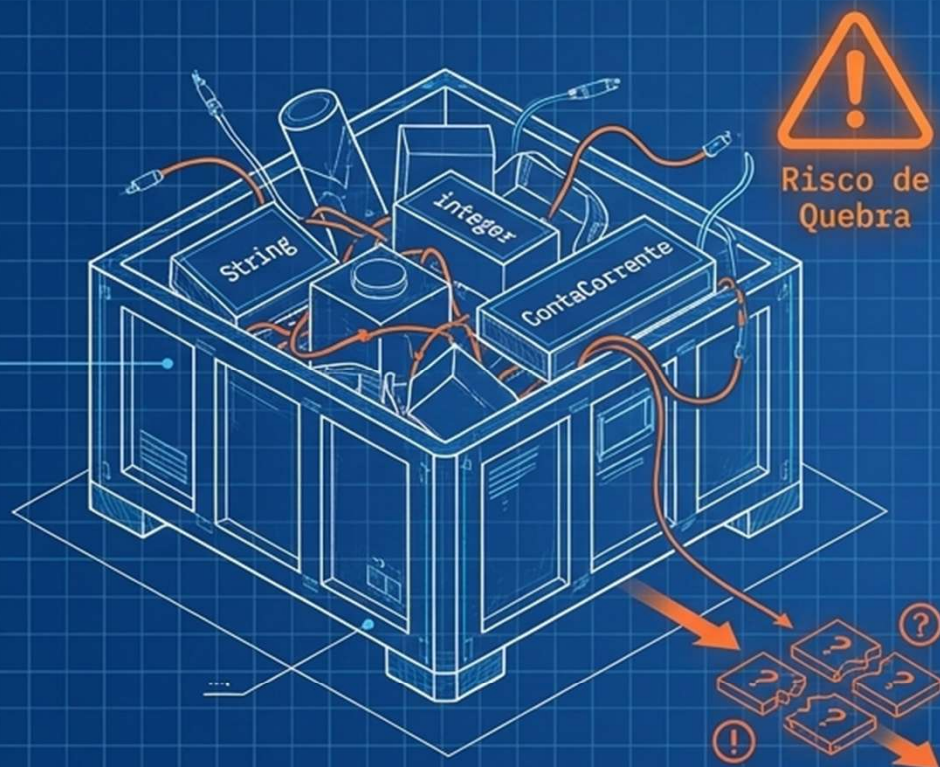
Verificação:

```
lista.contains(obj);  
(Confere se o item já existe lá dentro)
```



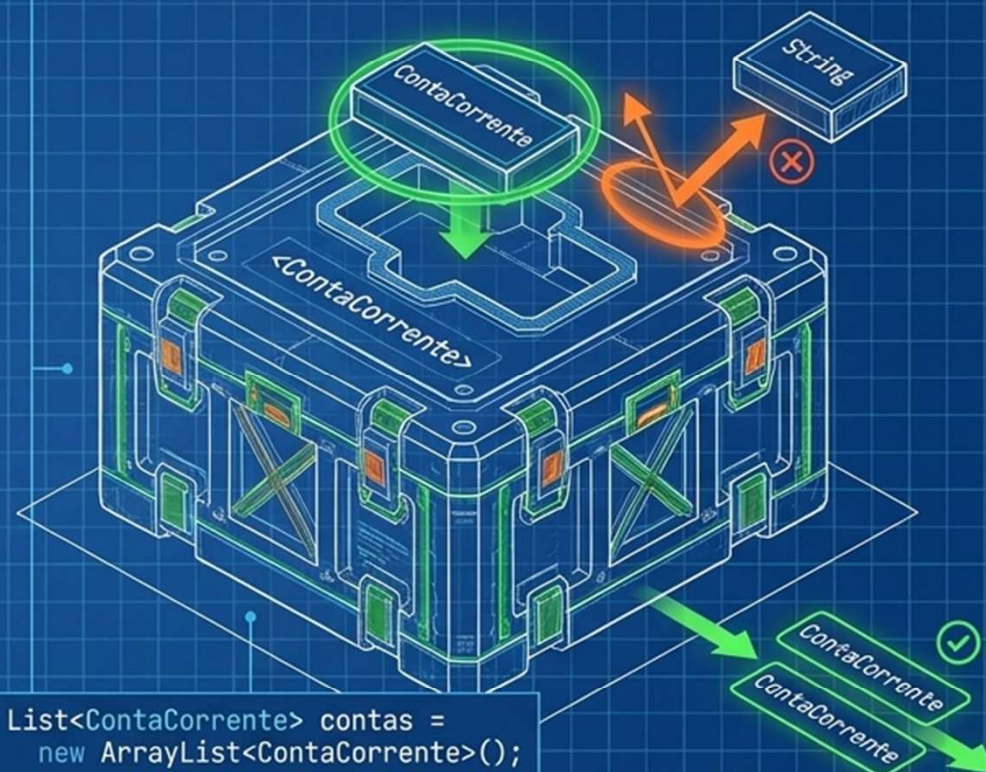
Generics: A Blindagem de Tipagem

O Problema (Sem Generics)



Uma lista comum aceita qualquer Object. Misturar tipos exige conversão forçada (casting) no resgate, podendo causar erros ocultos que quebram o código.

A Solução (Com Generics)



```
List<ContaCorrente> contas =  
    new ArrayList<ContaCorrente>();
```

O Efeito: A lista é selada. Trabalha exclusivamente com objetos ContaCorrente. Tentar injetar uma String não passará nem mesmo pelo compilador.

Iteração Inteligente (Enhanced For)

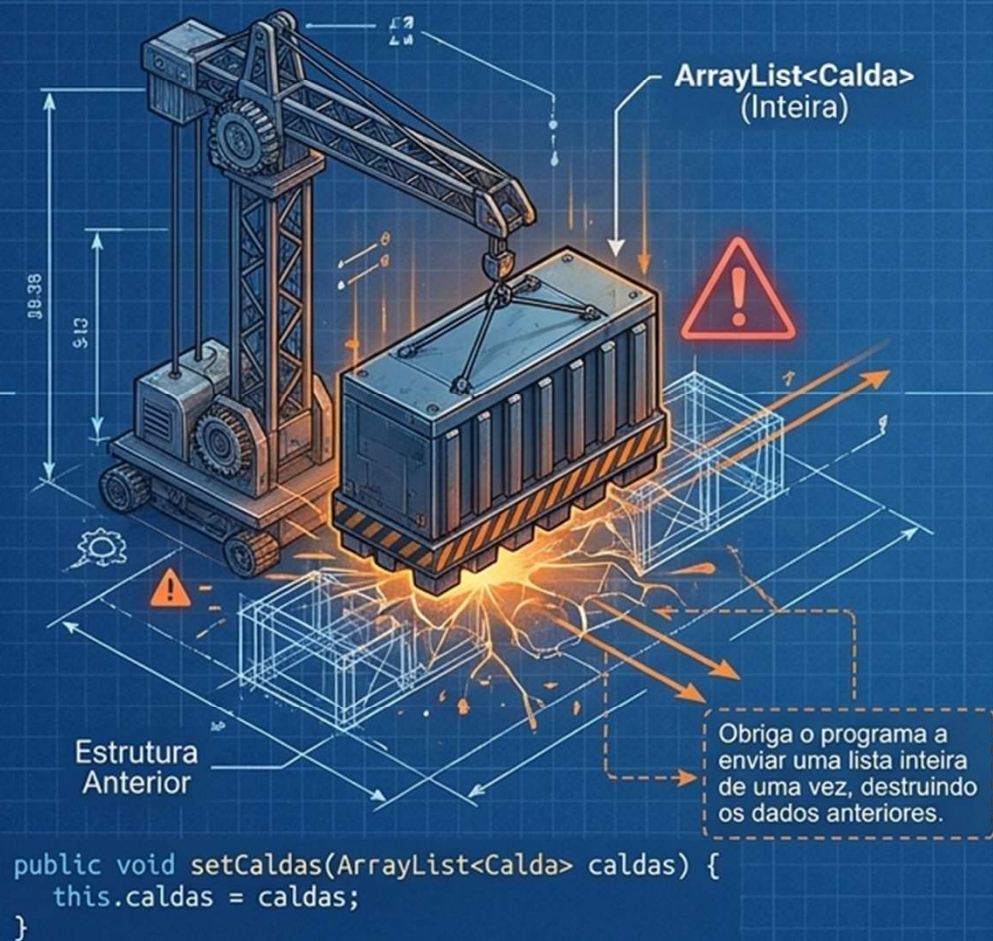
Como percorrer os objetos do Array selado sem complexidade com índices matemáticos.



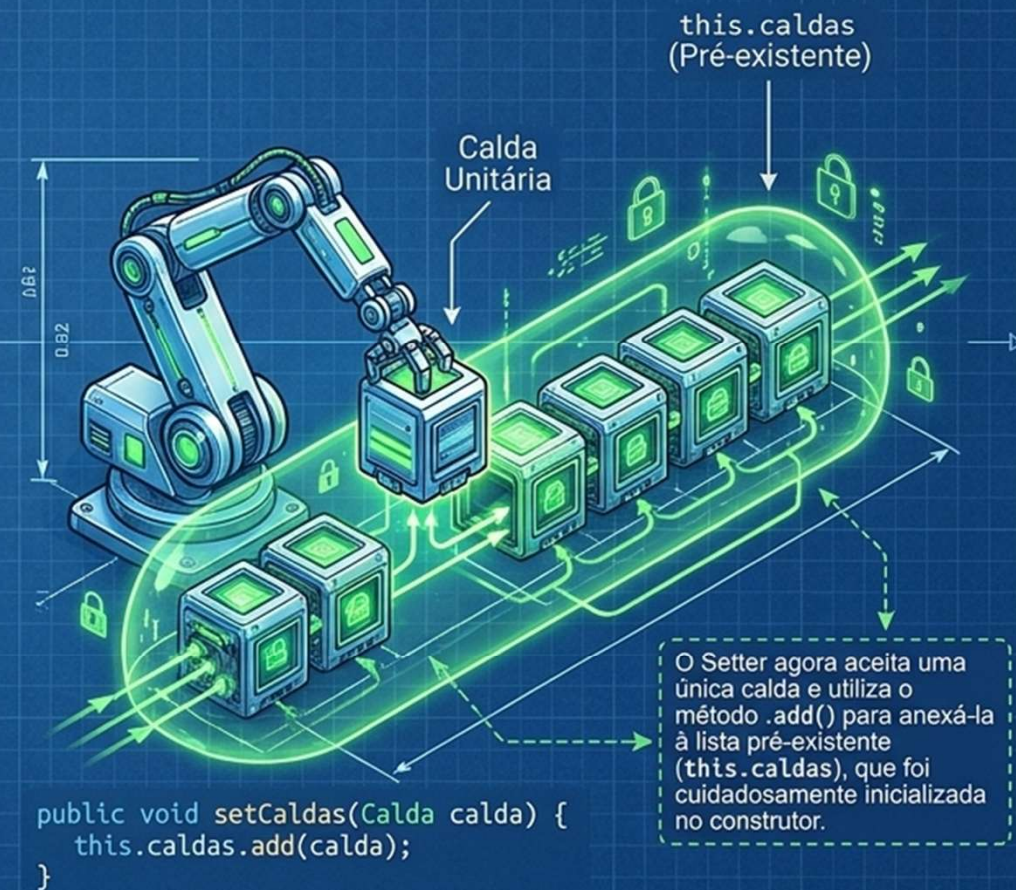
Resultado: Acesso direto e limpo aos métodos sem necessidade de casting.

Setters Inteligentes para Coleções (1:N)

O Erro Comum - Sobreescrita Total



Refatoração Arquitetônica - Injeção Unitária



O Blueprint da Arquitetura Perfeita

1. Proteção de Núcleo:

Todos os atributos operam sob a blindagem do modificador `private`.



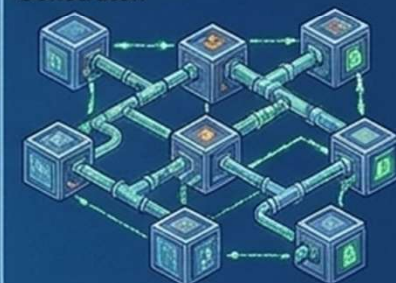
3. Tipagem Inquebrável:

Uso de `<Generics>` para banir dados invasores em tempo de compilação.



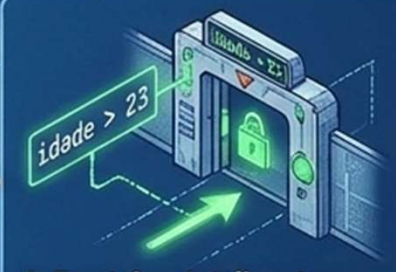
2. Relacionamentos Seguros:

Associações complexas geridas por `ArrayLists` inicializados no Construtor.



4. Fronteiras Inteligentes:

Métodos `Setters` não são meros repassadores; eles filtram regras de negócio (idade > 23) e orquestram inclusões seguras (`.add`).



Um design altamente coeso. A classe se defende sozinha, escala sob demanda e confia em sua própria arquitetura.