Desenvolvimento de Sistemas

Aula 05 - Paradigmas e Sistemas de Tipagem

Lorrany B A Marim

SENAI

Paradigmas Estruturado e Imperativo

A base da programação clássica

- **Foco no "Como":** O programador dita passo a passo o que o computador deve fazer. É uma sequência de instruções que alteram o estado do programa

Paradigmas Estruturado e Imperativo

- **Fluxo de Execução:** A execução começa na primeira instrução e segue de cima para baixo, podendo ser alterada por desvios (funções) e repetições.
- **Estrutura:** O código é dividido em blocos lógicos usando sequência, decisão (if/else) e iteração (loops).

Paradigmas Estruturado e Imperativo

- **Variáveis:** O uso de variáveis é central, funcionando como nomes que se referem a valores na memória que podem ser atualizados.

Paradigma Orientado a Objetos (POO)

Modelando o mundo real

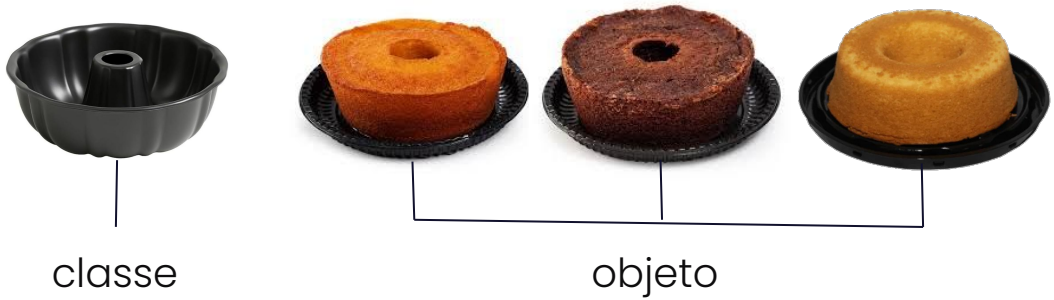
- **Objetos como agentes:** Em vez de focar apenas em funções, a POO foca nos dados (objetos) e nas operações que eles podem realizar.

- **Classes e Instâncias:**
  - **Classe:** É a fábrica ou molde. Define os atributos (dados) e métodos (comportamentos).

Paradigma Orientado a Objetos (POO)

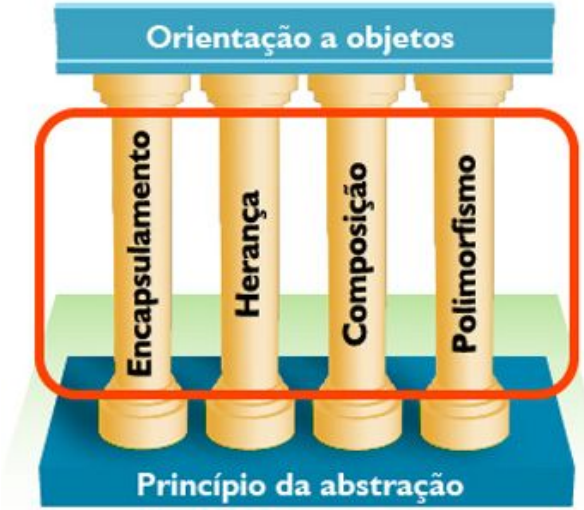
Modelando o mundo real

- **Classes e Instâncias:**
  - **Instância/Objeto:** É a concretização da classe. Se Carro é a classe, o Fusca na memória é o objeto.



Paradigma Orientado a Objetos (POO)

- **Encapsulamento:** Os dados internos do objeto e sua representação ficam ocultos; o acesso é feito através de operações (métodos).
- **Herança:** Permite definir uma nova classe como uma versão modificada de uma existente, promovendo reutilização de código.
- **Polimorfismo:** Capacidade de substituir objetos com interfaces compatíveis em tempo de execução, permitindo que um mesmo comando atue de forma diferente dependendo do objeto.



Paradigma Funcional

Matemática e Imutabilidade

- **Funções Puras:** São funções que não têm efeitos colaterais. Elas recebem uma entrada e geram uma saída sem alterar variáveis globais ou o estado de outros objetos.
- **Imutabilidade:** Evita-se a mudança de estado. Em vez de modificar uma lista, cria-se uma nova lista com as alterações desejadas.

- **Vantagens:**
  - **Testabilidade:** Como a função sempre retorna o mesmo resultado para a mesma entrada, é muito fácil de testar.

Paradigma Funcional

Matemática e Imutabilidade

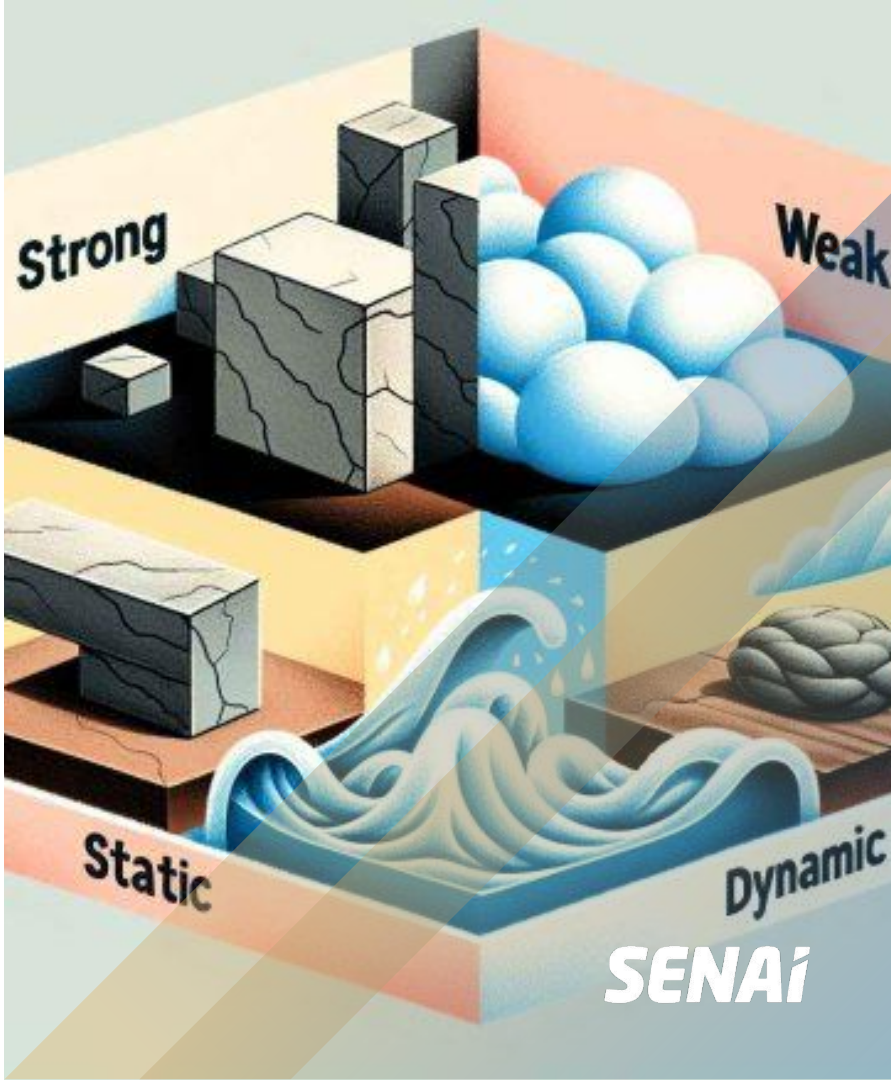
- **Vantagens:**
  - **Concorrência:** Facilita a execução paralela (várias threads), pois não há risco de uma função alterar o dado que outra está usando.

- **Ferramentas:** Uso intensivo de recursos como map, filter e compreensões de lista.

Tipagem Estática

Verificação em Tempo de Compilação

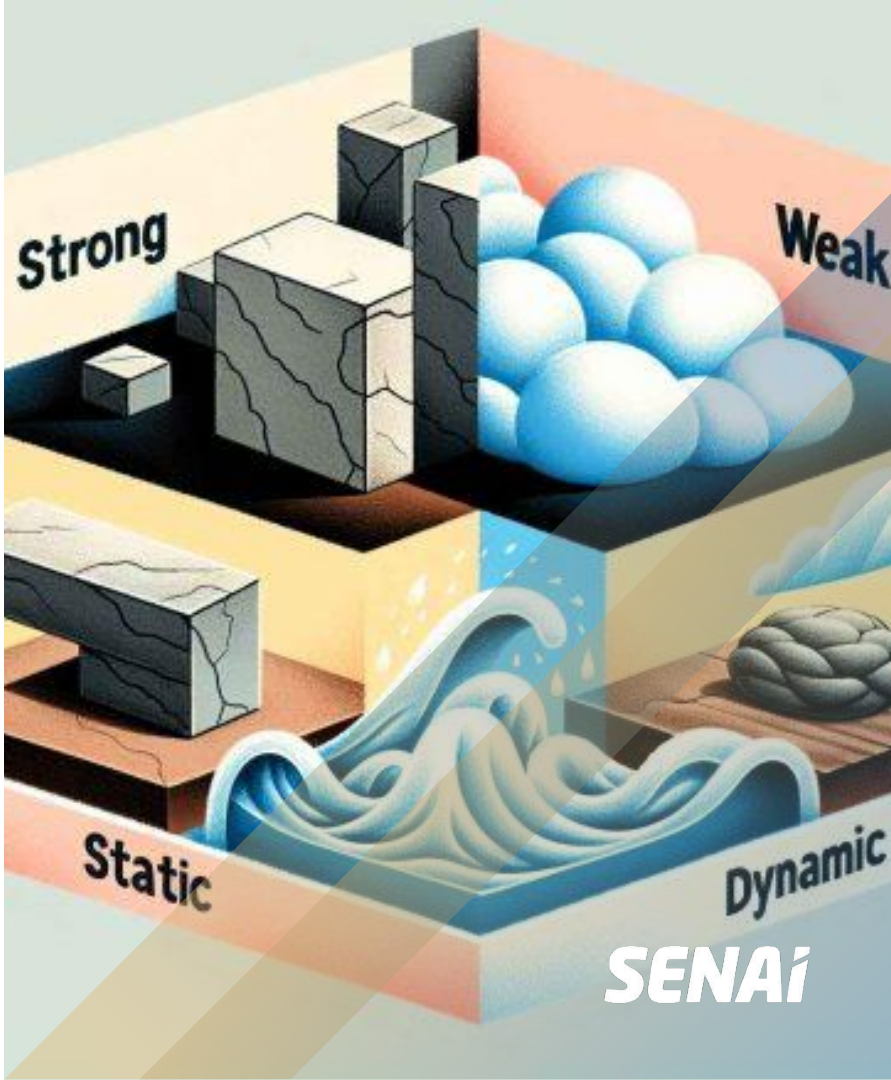
- **Definição Explícita:** Em linguagens estáticas, as variáveis têm tipos definidos na declaração e isso é verificado antes do programa rodar (compilação).



Tipagem Estática

Verificação em Tempo de Compilação

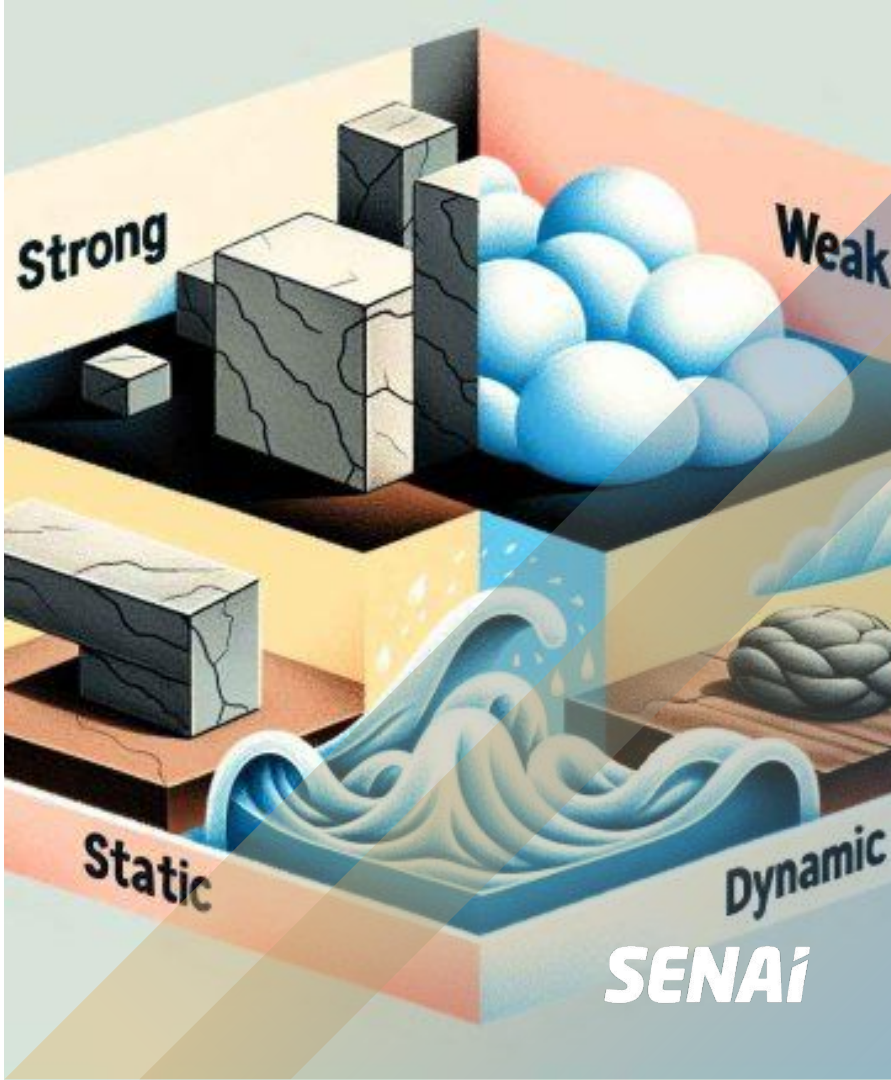
- **Segurança:** O compilador garante que você não está tentando usar um método que não existe naquele tipo de objeto [Gamma, 240].
- **Rigidez:** Uma vez declarada como int, a variável não pode receber uma string.



Tipagem Estática

Verificação em Tempo de Compilação

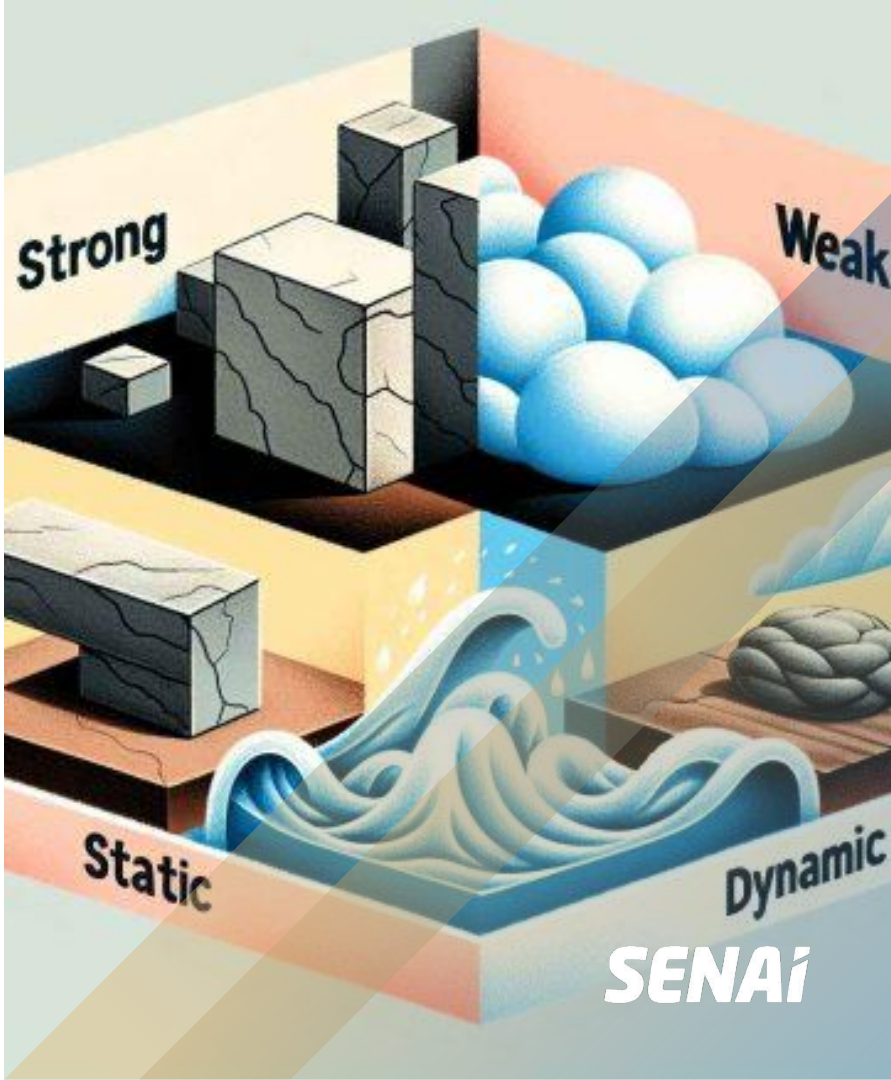
- **Interface:** A herança é essencial para permitir que objetos diferentes sejam tratados da mesma forma (polimorfismo), já que o tipo precisa ser compatível.



Tipagem Dinâmica

Verificação em Tempo de Execução

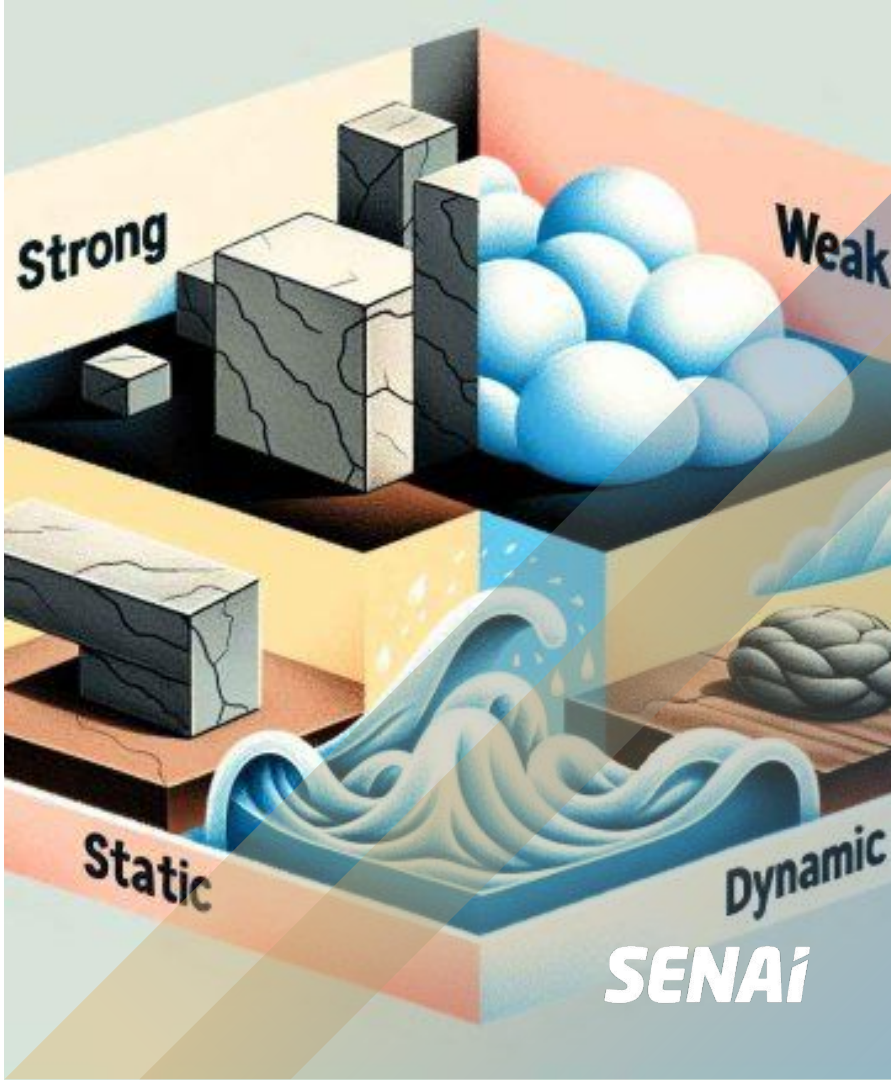
- **Flexibilidade:** As variáveis não têm tipo fixo; os valores (objetos) é que têm tipos. Uma variável pode referenciar um inteiro agora e uma string depois.



Tipagem Dinâmica

Verificação em Tempo de Execução

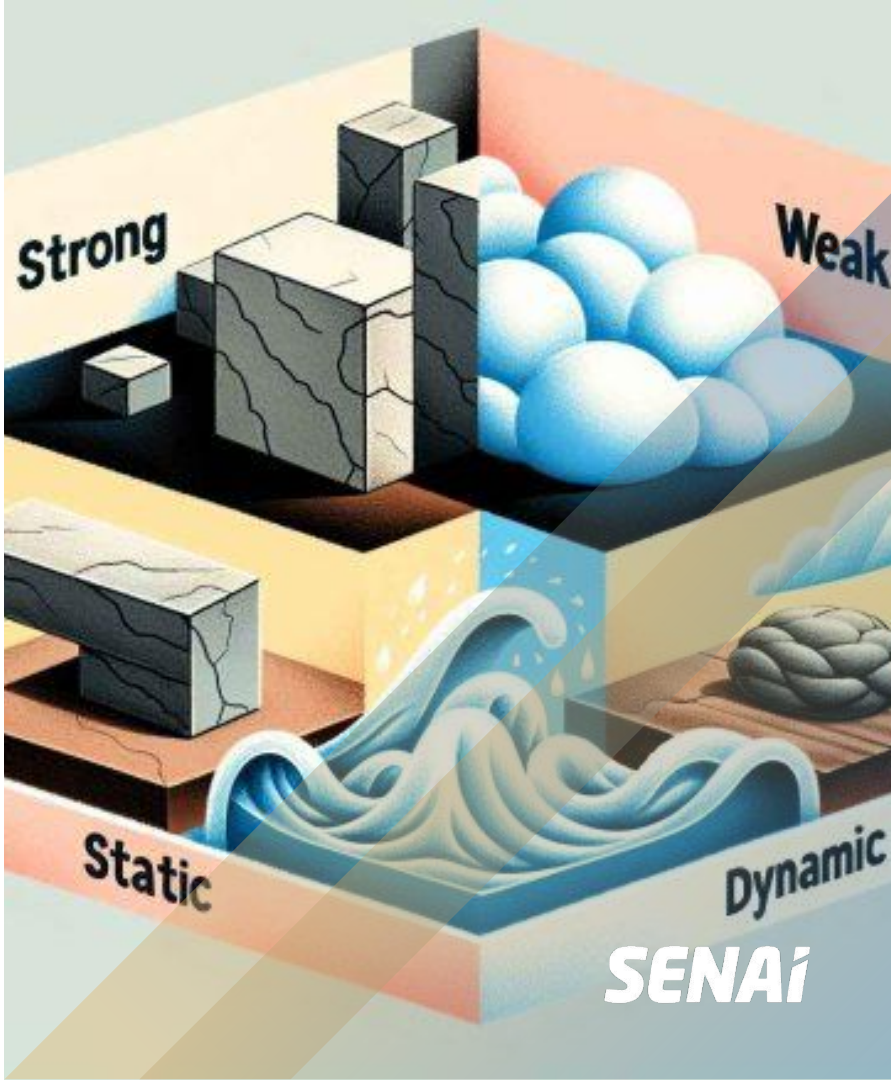
- **Verificação Tardia:** O interpretador verifica se a operação é válida apenas quando o código é executado.
- **Duck Typing:** "Se anda como pato e grasna como pato, é um pato".



Tipagem Dinâmica

Verificação em Tempo de Execução

- **...Duck Typing:** Se o objeto possui o método chamado, ele funciona, independente da sua classe base.
- **Menos Código:** Não é necessário declarar tipos explicitamente, tornando o código mais conciso.



Vantagens e Desvantagens

Qual escolher?

- **Orientado a Objetos vs. Funcional:**
  - *OO* é excelente para modelar sistemas complexos e gerenciar estado, mas o estado compartilhado pode gerar bugs de concorrência.
  - *Funcional* é mais conciso e seguro para processamento de dados e paralelismo, mas pode ser menos intuitivo para quem está acostumado a pensar em "passo a passo".

Vantagens e Desvantagens

Qual escolher?

- **Estática vs. Dinâmica:**
  - *Estática* oferece segurança e ajuda a detectar erros cedo (na compilação), mas é mais verbosa.
  - *Dinâmica* permite desenvolvimento rápido e flexibilidade (como criar tipos genéricos facilmente), mas erros de tipo podem aparecer só na mão do usuário (Runtime Error).

A Realidade do Mercado

- **Multiparadigma:** Linguagens modernas (como Python e Java atual) misturam paradigmas. Podemos usar objetos para estrutura e funções puras para processamento.

- **Não existe bala de prata:** Cada abordagem tem custos e benefícios. A escolha depende do problema a ser resolvido.

- **Boas Práticas:** Independente da tipagem, manter interfaces claras e separar a implementação da interface é fundamental para a manutenção do software.

Fontes Utilizadas

- *Downey, A. B. "Pense em Python".*
- *Gamma, E. et al. "Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos".*
- *Danjou, J. "Python Levado a Sério".*
- *Valente, M. T. "Engenharia de Software Moderna".*


Obrigado!
SENAI