Programação Orientada a Objetos

TIPOS ABSTRATOS DE DADOS

em C++

Introdução

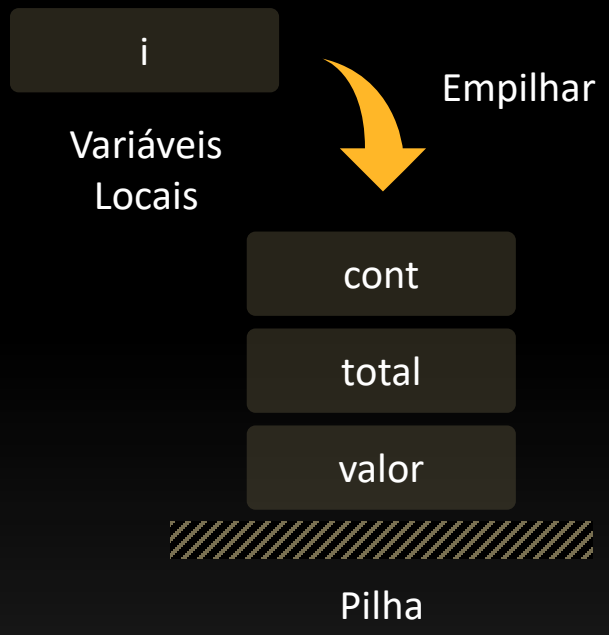
- **Classes** são usadas para representar objetos concretos
  - Temos feito isso em nossos exemplos:
    - Carros
    - Pessoas
    - Jogos
    - Atletas
    - Pontos
    - Retângulos

```
class Carro
{
private:
   int cor;
   string tipo;
   float velocidade;
   ...
};
```

```
class Jogo
{
private:
   string nome;
   float preco;
   int horas;
   ...
};
```

Introdução

- Mas elas são também frequentemente usadas para representar conceitos mais gerais chamados de:
  - Tipos Abstratos de Dados (TAD)
    - Armazenam dados
    - Fornecem operações
    - Abstraem detalhes de implementação
    - Podem ser usados com qualquer tipo de dado

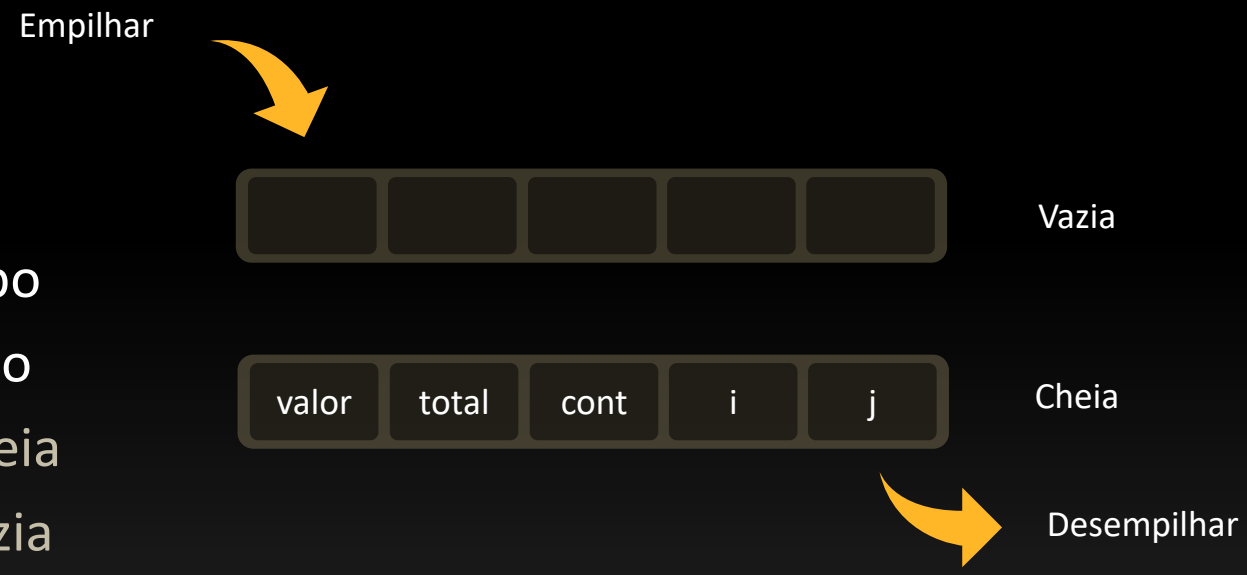


Pilha

- A **pilha** armazena dados
  - Ficam encapsulados
    - Atributos privados
    - Implementado através:
      - Vetor estático
      - Vetor dinâmico
      - Lista encadeada
      - Etc.

Pilha

- Uma **pilha** implementa operações
  - Fornecem uma interface pública
  - Modificam atributos
    - Métodos:
      - Criar pilha vazia
      - Adicionar elemento ao topo
      - Remover elemento do topo
      - Verificar se a pilha está cheia
      - Verificar se a pilha está vazia



Pilha

- Um TAD pode ser criado com uma **classe**

```
class Pilha
{
private:
   enum { MAX = 10 };                  // limite de itens
   Item itens[MAX];                    // armazenamento de itens
   int topo;                           // índice do item no topo

public:
   Pilha();                            // construtor

   bool Vazia() const;                 // verifica se a pilha está vazia
   bool Cheia() const;                 // verifica se a pilha está cheia

   bool Empilhar(const Item & item);   // adiciona item na pilha
                                       // retorna falso se a pilha estiver cheia
   bool Desempilhar(Item & item);      // remove item da pilha
                                       // retorna falso se a pilha estiver vazia
};
```

Constantes em Classes

- Por que essa **enumeração** dentro da classe?
  - Ela está criando um nome simbólico para um valor
  - Mas, por que não usar uma constante?
    - Valores são armazenados em objetos
    - Ainda não existe um objeto

```
class Pilha
{
private:
   enum { MAX = 10 }; ✓
   Item itens[MAX];
   ...
};
```

```
class Pilha
{
private:
   const int Max = 10; ✗
   Item itens[Max];
   ...
};
```

A **classe** representa um modelo para a criação de um objeto. Ela não guarda valores.

† C++11 permite a criação de constantes em classes mas não para fornecer valor para a declaração de outros membros

Constantes em Classes

- Por que a enumeração funciona?
  - A enumeração não define um membro da classe
  - Ela é armazenada de forma independente dos objetos
  - Defini-la na classe apenas limita seu escopo

```
class Pilha
{
public:
   enum { MAX = 10 }; ✓
   Item itens[MAX];
   ...
};
```

```
int main()
{
   int a = Pilha::MAX; ✓
   int b = MAX;        ✗
}
```

MAX existe mesmo sem a criação de objetos.

Constantes em Classes

- Outra forma de criar constantes em uma **classe**
  - Membros estáticos são criados fora dos objetos
  - Seu valor é o mesmo para todos os objetos

```
class Pilha
{
private:
   enum { MAX = 10 }; ✓
   Item itens[MAX];
   ...
};
```

```
class Pilha
{
private:
   static const int Max = 10; ✓
   Item itens[Max];
   ...
};
```

Membros estáticos não são armazenados nos objetos.

A Classe Pilha

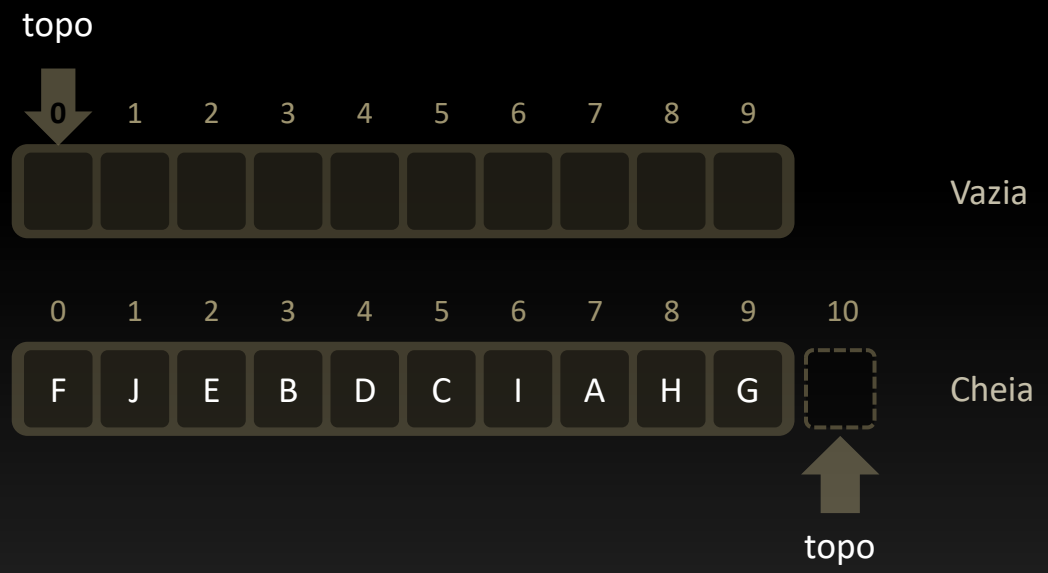
```
// definição do tipo Item
using Item = char;

// declaração da classe Pilha
class Pilha
{
private:
   enum { MAX = 10 };                  // limite de itens
   Item itens[MAX];                    // armazenamento de itens
   int topo;                           // índice do item no topo

public:
   Pilha();                            // construtor

   bool Vazia() const;                 // verifica se a pilha está vazia
   bool Cheia() const;                 // verifica se a pilha está cheia

   bool Empilhar(const Item & item);   // adiciona item na pilha
   bool Desempilhar(Item & item);      // remove item da pilha
};
```

» Pilha.h

A Classe Pilha

```cpp
// definição da classe Pilha
#include "Pilha.h"

Pilha::Pilha()
{
    topo = 0;
}

bool Pilha::Vazia() const
{
    return topo == 0;
}

bool Pilha::Cheia() const
{
    return topo == MAX;
}

...
```

```cpp
class Pilha
{
private:
   enum { MAX = 10 };
   Item itens[MAX];
   int topo;
   ...
};
```



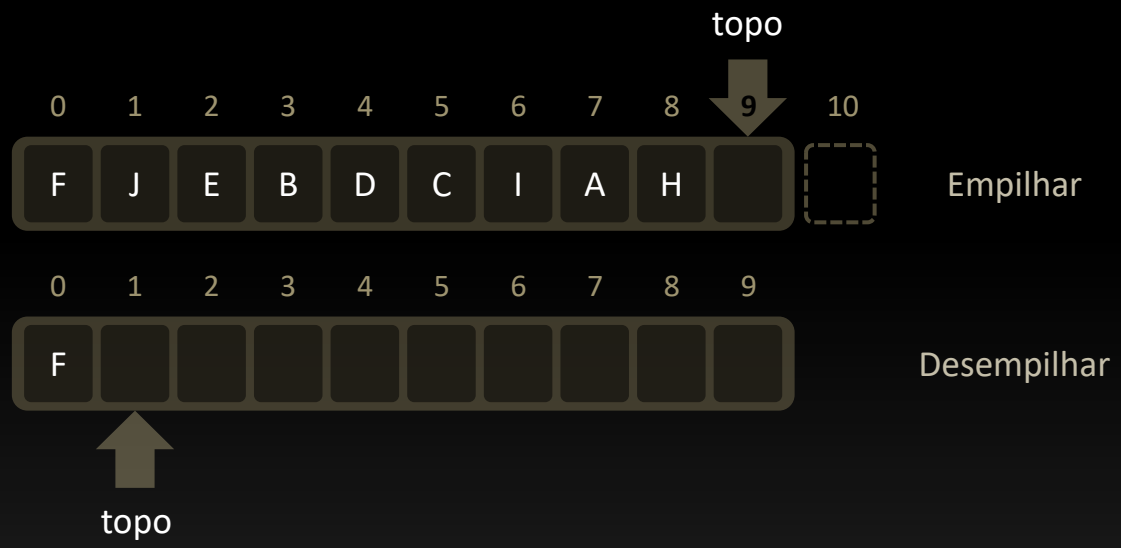
» Pilha.cpp

A Classe Pilha

```
bool Pilha::Empilhar(const Item& item)
{
    if (topo < MAX) {
        itens[topo++] = item;
        return true;
    }
    return false;
}
```

```
bool Pilha::Desempilhar(Item& item)
{
    if (topo > 0) {
        item = itens[--topo];
        return true;
    }
    return false;
}
```

```
class Pilha
{
private:
    enum { MAX = 10 };
    Item itens[MAX];
    int topo;
    ...
};
```



» Pilha.cpp

A Classe Pilha

```cpp
#include <iostream>
#include "Pilha.h"
using namespace std;

int main()
{
    Pilha pilha;

    pilha.Empilhar('A');
    pilha.Empilhar('H');
    pilha.Empilhar('L');
    pilha.Empilhar('I');
    pilha.Empilhar('P');

    while (!pilha.Vazia())
    {
        Item i;
        pilha.Desempilhar(i);
        cout << i << endl;
    }
}
```

```cpp
class Pilha
{
private:
   enum { MAX = 10 };
   Item itens[MAX];
   int topo;
   ...
};
```

» Empilhando.cpp

A Classe Pilha

- Saída do programa:

```
P
I
L
H
A
```

  - O desempilhamento pode ser diferente:

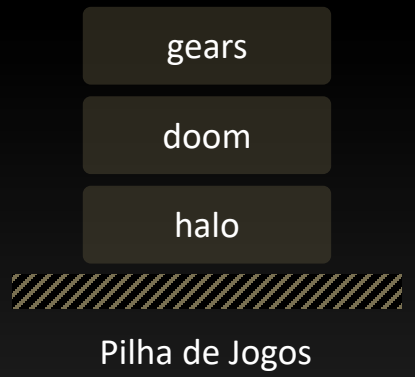
```
Item i;
while (pilha.Desempilhar(i))
    cout << i << endl;
```

```
class Pilha
{
private:
   enum { MAX = 10 };
   Item itens[MAX];
   int topo;

public:
   Pilha();
   bool Vazia() const;
   bool Cheia() const;
   bool Empilhar(const Item &);
   bool Desempilhar(Item &);
};
```

Resumo

- Classes podem ser usadas para implementar
  - Tipos concretos
    - Jogo
    - Carro
    - Pessoa
  - Tipos abstratos
    - Lista
    - Pilha
    - Fila

Um **TAD** abstrai o tipo de dado armazenado e fornece operações para manipulação dos dados.



Resumo

- A **orientação a objetos** fornece maior segurança
  - Pelo encapsulamento dos dados
  - E uso de uma interface pública

    - A classe Pilha é um exemplo:
      - O construtor garante a construção de uma pilha vazia
      - O método empilhar garante a existência de espaço
      - Desempilhar garante que a pilha não está vazia
      - Os métodos gerenciam o topo da pilha