Universidade Cruzeiro do Sul

Estruturas de dados

Conteúdo:

- Tipos de dados.
- Tipo abstrato de dados (TAD)
- Estrutura de dados pilha.
  - Pilhas estáticas sequenciais.

Autores

Prof. Manuel F. Paradela Ledón
Profª Cristiane P. Camilo Hernandez
Prof. Amilton Souza Martha
Prof. Daniel Calife

Tipos de Dados

- Tipo de dados um **conjunto** ou coleção **de valores** que uma constante, variável ou expressão pode assumir (domínio) e um grupo de **operações** que podem ser efetuadas para manipular esses dados.

```
Exemplo:   int  idade;          //a variável 'idade' poderá armazenar
                                //valores inteiros, como 4, 22 ou 76
Exemplo:   x * 2.0 + 3.1        //a expressão  x * 2.0 + 3.1 assumirá um
                                //valor dentro do conjunto de números reais
```

- Um *conjunto de valores* que possam ser gerados por uma função:

```
Exemplo:   int  quad(int v) {   //a função 'quad' retornará um valor
               return(v*v);     //dentro do conjunto de números inteiros
           }
```

- Os tipos de dados diferem conforme o sistema operacional e a linguagem utilizada. Exemplo: o tipo de dado int em Java utiliza 4 bytes e permite guardar valores entre -2^{31} e 2^{31}. Em outras linguagens e S.O. poderia ser diferente.

Tipos de dados

- Exemplos genéricos (independentes da linguagem):

Tipo	Valores
Inteiro	-20, 0, 47, 99
Real	23.4, 68.2, -15.0
String	"Dados", "árvore", "pilha"
Meses	Janeiro, Fevereiro, Março

- Os operadores poderão ter significados (operações) diferentes dependendo dos tipos de dados dos operandos:

```
x + 2.0                 //soma dois valores reais (adição)
"Uma" + " casa"         //concatena, junta, duas strings
```

Tipos Primitivos

- São os tipos de dados básicos que, além de depender das características do sistema, dependem da linguagem de programação utilizada.

- São aqueles a partir dos quais podemos definir os demais tipos ou organizações de informações, quase sempre mais complexas.

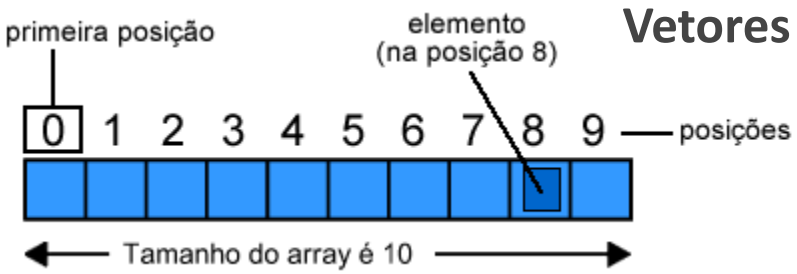
- Possuem operações que podem ser aplicadas a dados desse tipo, como a adição, subtração, multiplicação, divisão etc.

Exemplo: tipos de dados primitivos em Java

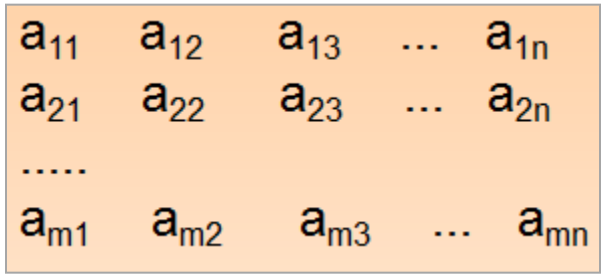
Tipo	Descrição
boolean	Pode assumir o valor true ou o valor false
char	Serve para a armazenagem de um valor alfanumérico. Também pode ser usado como um dado inteiro com valores na faixa entre 0 e 65535.
byte	Inteiro de 8 bits (1 byte) em notação de complemento de dois. Pode assumir valores entre -2^7=-128 e 2^7-1=127.
short	Inteiro de 16 bits em notação de complemento de dois. Os valores possíves cobrem a faixa de -2^{15}=-32.768 a 2^{15}-1=32.767
int	Inteiro de 32 bits (4 bytes) em notação de complemento de dois. Pode assumir valores entre -2^{31} = -2.147.483.648 e 2^{31}-1 = 2.147.483.647.
long	Inteiro de 64 bits (8 bytes) em notação de complemento de dois. Pode assumir valores entre -2^{63} e 2^{63}-1.
float	Representa números em notação de ponto flutuante normalizada em precisão simples de 32 bits em conformidade com a norma IEEE 754-1985. O menor valor positivo representável por esse tipo é 1.40239846e-46 e o maior é 3.40282347e+38
double	Representa números em notação de ponto flutuante normalizada em precisão dupla de 64 bits em conformidade com a norma IEEE 754-1985. O menor valor positivo representável é 4.94065645841246544e-324 e o maior é 1.7976931348623157e+308

Tipos de dados estruturados

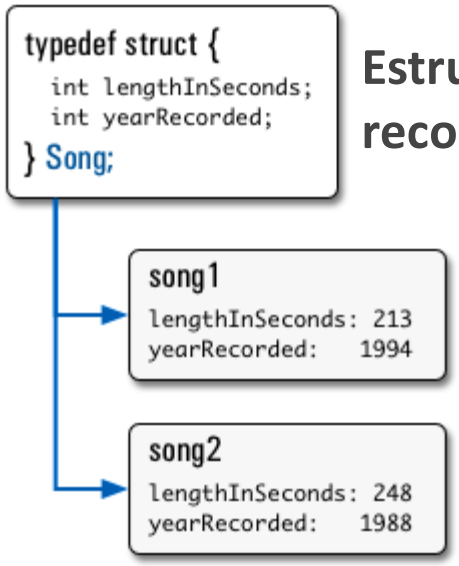
- São organizações de dados obtidas normalmente a partir dos tipos de dados primitivos.
- A maioria das linguagens de programação fornecem alguns tipos estruturados para facilitar a organização de dados. Os mais frequentes são:



Vetores



Matrizes



Estruturas, records (registros)

Listas Python

lista = [4.0, 9.2, 3.1, 9.1, 1.5, 3.2, 2.6]

Abstração (antes de ver o conceito de TAD)

- Um computador ainda tem sérias limitações físicas para representar a complexidade do mundo real. Nosso mundo contêm ricos detalhes que não podem ser inseridos ou representados diretamente no computador.

- É **abstraindo** nossa realidade que **podemos capturar o que existe de mais relevante** em uma situação real, tornando possível a construção de **modelos** que podem ser implementados nos computadores por meio de uma linguagem de programação.

abstração

substantivo feminino

ato ou efeito de abstrair(-se); abstraimento

- processo mental que consiste em isolar um aspecto determinado de um estado de coisas relativamente complexo, a fim de simplificar a sua avaliação, classificação ou para permitir a comunicação do mesmo

Dicionário Houaiss Eletrônico, 2009.

- ato de separar mentalmente um ou mais elementos de uma totalidade complexa (coisa, representação, fato), os quais só mentalmente podem subsistir fora dessa totalidade.

Novo Dicionário Eletrônico Aurélio versão 6.0, 2009.

Tipo Abstrato de Dados (TAD)

- É um conjunto de valores e uma série de operações que atuam sobre esses valores.
- As operações devem ser consistentes com os tipos de valores.
- Frequentemente definimos pré-condições e pós-condições para as diferentes operações de um TAD.
- Podemos considerar tipo como um conjunto de possibilidades ou domínio válido para os dados. Os elementos dos conjuntos são entendidos como os valores daquele tipo.
- Neste material vamos observar as diferenças entre os conceitos: Tipo Abstrato de Dados (TAD) e Estrutura de Dados (ED).

Tipo Abstrato de Dados (TAD)

- Para declarar um TAD, precisamos especificar não só um tipo de dados (conjunto), mas também operações (métodos) que podem ser feitas sobre os dados e as relações entre seus elementos.
- Associação entre a declaração de um tipo de dados e a declaração das operações que incidem sobre as variáveis deste tipo.
- Vantagens de utilizar TAD:
  - Integridade dos dados;
  - Facilidade de manutenção;
  - Reutilização.
- Um TAD está desvinculado de sua implementação, ou seja, quando definimos um TAD estamos preocupados com o que ele faz e não em como ele faz ou como serão implementados os dados e as operações.

Tipo Abstrato de Dados (TAD)

Segundo **Goodrich e Tamassia** (no livro Estruturas de Dados e Algoritmos em Java):

"**Um TAD é um modelo matemático** de estruturas de dados **que especifica os tipos dos dados armazenados, as operações definidas sobre estes dados e os tipos de parâmetros destas operações**. Um TAD define o que cada operação faz, mas não como o faz. Em Java, um TAD pode ser expresso por uma **interface***, que é uma simples lista de **declarações** de métodos."

"**Um TAD é materializado por uma estrutura de dados** concreta que, em Java, é modelada por uma classe. Uma **classe define** os dados que serão armazenados e as operações suportadas..."

Analisar a diferença entre declarar e definir. Lembrar os conceitos de interfaces, classes abstratas e classes concretas. *Um TAD poderia ser uma *interface* ou uma *classe abstrata*.

Tipo Abstrato de Dados (TAD)

Segundo **Shaffer** (2012) (no livro Data Structures and Algorithm Analysis):

"Um **tipo abstrato de dados** (**TAD**) é a realização de um tipo de dados como um componente de software. A interface do TAD é definida em termos de um tipo e um conjunto de operações sobre esse tipo. O comportamento de cada operação é determinado pelas suas entradas e saídas. Um TAD não especifica como o tipo de dados é implementado.

Uma **estrutura de dados** é a implementação de um TAD. Em uma linguagem orientada a objetos como Java, um TAD e sua implementação juntos compõem uma classe. Cada operação associada ao TAD é implementada por uma função membro ou método."

Tipo Abstrato de Dados (TAD) como contrato

O **TAD** deverá especificar um "**contrato**" para que futuros *implementadores* desenvolvam adequadamente as estruturas de dados com base a esse modelo.

Este contrato (no **TAD**) considera:

- os nomes das operações deste TAD;
- os tipos de dados retornados e tipos de dados dos parâmetros;
- o comportamento geral das operações;
- opcionalmente: as pré-condições e pós-condições das operações do TAD.

Veja que no contrato (TAD), normalmente, **não serão especificados**:

- a representação específica dos dados para implementar este TAD;
- nem os algoritmos específicos para efetuar a implementação.

Exemplo: um TAD chamado TAD_NotaFinal

```
public interface TAD_NotaFinal {

    public float calculaNotaFinal();
        //Calcula a nota final do aluno.
        // pré-condições:
        //    cada nota deverá ser >= 0
        //    a soma das notas deverá ser <= 10
        // pós-condições:
        //    a nota final será a soma de todas as notas

    public float calculaMediaDasNotas();
        //Calcula a média das notas do aluno.
        //pré-condições:
        //    cada nota deverá ser >= 0
        //    a soma das notas deverá ser <= 10
        //pós-condições:
        //    a média será a soma de todas as notas
        //    dividida pela quantidade de notas
}
```

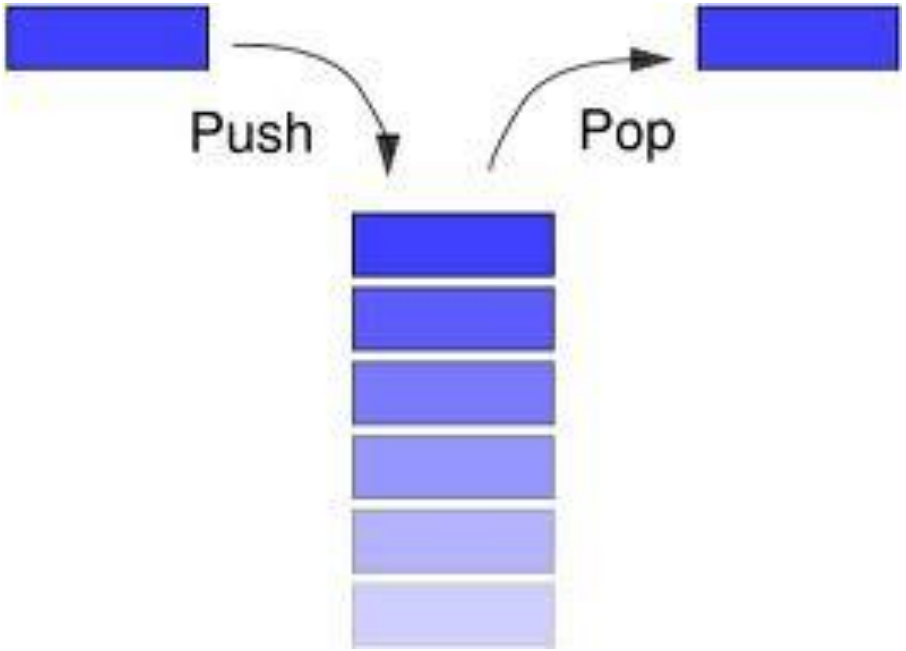
Exemplo: uma classe que implementa o TAD

```
public class EAD_NotaFinal  implements TAD_NotaFinal {

    private float n1 = -1, n2 = -1, n3 = -1;

    public float calculaNotaFinal() {
        if( n1>=0 && n2>=0 && n3>=0 && (n1+n2+n3)<=10 )
            return (n1+n2+n3); else return -1;
    }

    public float calculaMediaDasNotas() {
       if( n1>=0 && n2>=0 && n3>=0 && (n1+n2+n3)<=10 )
            return (n1+n2+n3)/3; else return -1;
    }

    //... construtores e outros métodos

}
```

Pilhas

- Uma pilha é um tipo de dado abstrato para implementação de uma estrutura de dados conhecida pelo critério de inserção/retirada dos elementos: **Last In, First Out** (**LIFO**), em português: o último que entrou será o primeiro a sair;
- Mantém seus elementos em sequência;
- Caracterizada pela restrição de acesso apenas ao último elemento inserido (acessar somente o elemento no topo);
- **Inserções** e **remoções** são feitas numa mesma extremidade denominada **topo**.

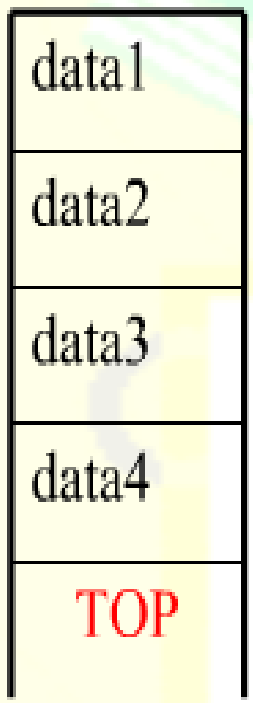
- Resumindo, temos acesso somente ao topo da pilha, ou seja, quando queremos **inserir** um elemento na pilha, colocamos no topo e se quisermos **excluir** (retirar) um elemento da pilha, só podemos excluir aquele que está no topo.

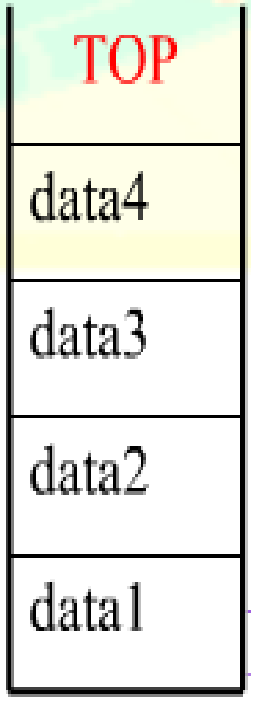


Uma **pilha** possui três operações básicas (depois adicionaremos outras operações):

- **top**: acessa (retorna, sem eliminar) o elemento posicionado no topo;
- **push**: insere um novo elemento no topo da pilha;
- **pop**: remove um elemento do topo da pilha.

Representações gráficas de uma pilha





Os estados de uma pilha (uma pilha em vetor)

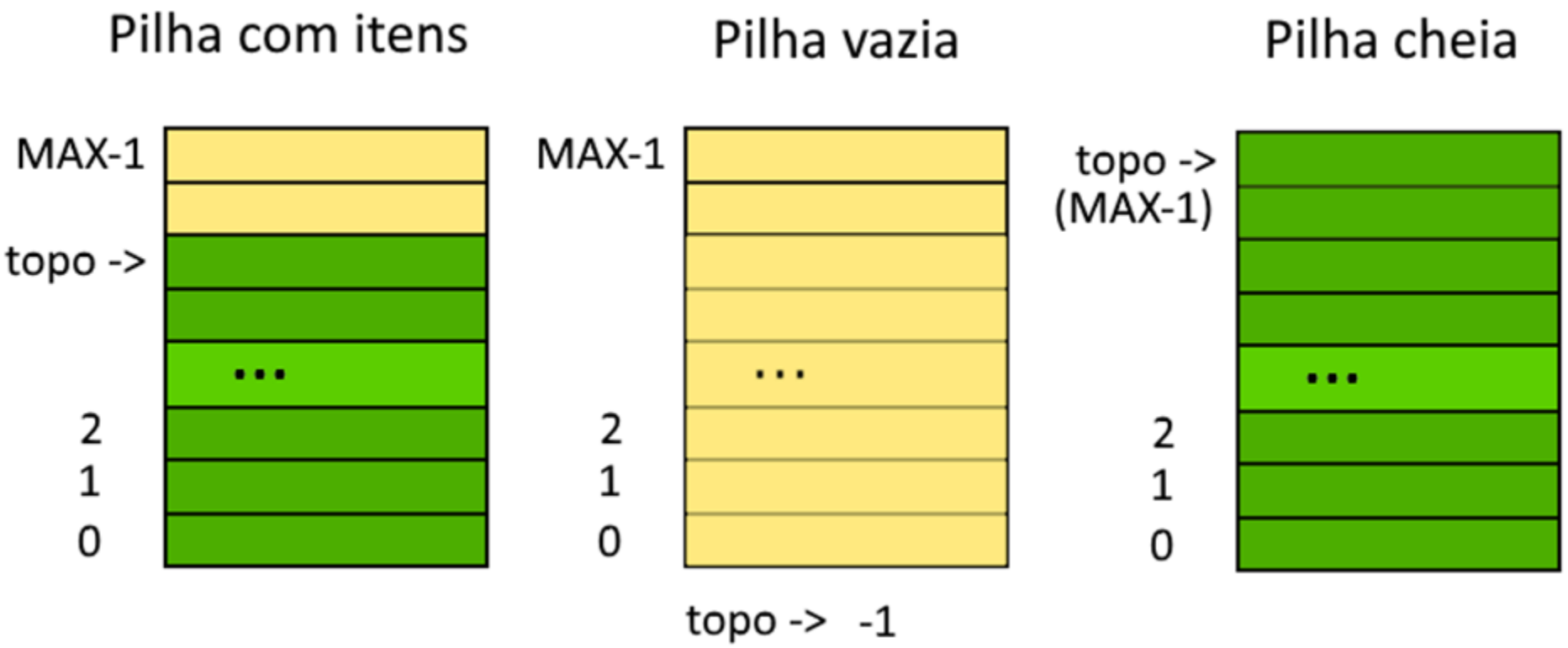


Tabela com exemplos – uma pilha P

--------	P:[ ]	Inicialmente a Pilha está Vazia
push(a)	P:[a]	Insere o elemento a na Pilha P
push(b)	P:[a, b]	Insere o elemento b na Pilha P. Note que agora o topo é b
push(c)	P:[a, b, c]	Insere o elemento c na Pilha P. Agora o topo contém c
pop()	P:[a,b]	Remove o elemento do topo e retorna o conteúdo. No caso c
push(d)	P:[a, b, d]	Insere o elemento d no topo.
top()	P:[a, b, d]	Não altera a Pilha, apenas retorna o valor do topo da Pilha (d)
push(top())	P:[a, b, d, d]	Insere no topo mais uma cópia do topo

Exemplo

- Pense em uma pilha de pratos, onde sempre colocamos o prato no topo da pilha e retiramos também do topo.
- Não é possível inserir pratos indefinidamente, pois existe um limite (no caso o teto) e nem tirar pratos indefinidamente (pilha vazia).
- Implementação computacional → também existem tais limites:
  - a quantidade de elementos não pode exceder a quantidade de memória alocada;
  - não podemos retirar elementos de uma pilha vazia.

Para suprir esses limites, precisamos de mais quatro métodos:

- **construtor** - inicializa a pilha no estado "vazia" e, opcionalmente, aloca memória inicial;
- **isEmpty** - verifica se a pilha está vazia;
- **isFull** - verifica se a pilha está "cheia" (se chegou no limite da capacidade, acabou a memória);
- **toString** - para retornar os itens da pilha.

Exemplos de aplicações de pilhas

- Análise de expressões matemáticas:
  - identifica se os elementos separadores de abertura '[', '{' e '(' são encerrados de forma correta com os respectivos elementos separadores de encerramento ')', '}' e ']'.
- Avaliação de uma expressão pós-fixa:
  - Infixa: **((1 + 2) * 4) + 3**
  - Pós-fixa: **1 2 + 4 * 3 +**
  - Utilizando uma pilha consideramos:
    - empilhar quando encontrar um operando;
    - desempilhar dois operandos e achar o valor quando encontrar uma operação;
    - empilhar o resultado.
- Chamadas de métodos e retornos (nos compiladores).

Tipo Abstrato de Dados (TAD): Pilha

- Uma **Pilha** de elementos do tipo T é uma sequência finita de elementos de T, sobre a qual podemos realizar as operações:
  - **Criar** uma pilha vazia;
  - Testar se a pilha **está vazia;**
  - Testar se a pilha **está cheia;**
  - **Empilhar** (**push**) um elemento no topo de uma pilha não-cheia;
  - **Desempilhar** (**pop**) o elemento do topo de uma pilha não-vazia;
  - Ler o elemento no **topo** de uma pilha não-vazia.
- Todas estas operações possuem tempo de execução constante em relação à entrada: **O(1)**.

Exemplo: TAD chamado TAD_Pilha (1)

```
public interface TAD_Pilha {

    public Object push (Object x);
        //Método para inserir um novo item x no topo da pilha.
        // pré-condições:
        //    a pilha deverá ter espaço para inserir: verificar !isFull()
        //    o objeto a inserir x não poderá ser nulo
        // pós-condições:
        //    retornará diferente de nulo se a inserção foi possível
        //    o topo será modificado, apontando ao novo item

    public Object pop ();
        //Método para retirar o item que se encontra no topo da pilha.
        // pré-condições:
        //    não poderá retirar da pilha se estiver vazia: verificar !isEmpty()
        // pós-condições:
        //    retornará o objeto que se encontra no topo da pilha
        //    retornará nulo se a pila está vazia
        //   o topo será modificado, apontando ao próximo item
```

Exemplo: TAD chamado TAD_Pilha (2)

```
    public boolean isEmpty ();
        //Método que verifica se a pilha está vazia.
        //Exemplo: não poderão ser retirados valores se a pilha estiver vazia.

    public boolean isFull ();
        //Método que verifica se a pilha está cheia, ou seja, não existe
        //memória para adicionar mais elementos.

     public Object top();
       //Método para retornar o item que se encontra no topo da
       //pilha, sem eliminar o mesmo
       // pré-condições:
       //    não poderá retornar o item se a pilha se estiver vazia
       // pós-condições:
       //    retornará o objeto que se encontra no topo da pilha
       //    retornará nulo se a pila está vazia
       //    o topo não será modificado, nem a pilha

      // Falta a descrição do método toString()
}
```

Implementação estática sequencial de pilhas

Estática

A alocação ou reserva de memória pode ser estática ou dinâmica. Neste material estudaremos pilhas com alocação estática de memória.
Entendemos o termo *estática* como uma quantidade máxima fixa de memória reservada, seja na compilação ou na execução do programa.

Sequencial

Os elementos de uma pilha poderiam ser colocados sequencialmente (um depois do outro, na sequência, dentro de um vetor, por exemplo) ou encadeados/enlaçados/ligados na memória interna. Nesta primeira etapa estudaremos pilhas com elementos colocados em forma sequencial.

A seguir mostraremos uma estrutura de dados **Pilha**, como uma implementação **estática** e **sequencial** do **TAD_Pilha** anterior, utilizando a linguagem Java, com um vetor de objetos.

Implementação estática sequencial de pilhas em Java

```
package pilhaestaticasequencial;

public class Pilha implements TAD_Pilha {
        private int topo;                   //topo da pilha
        private int MAX;                    //tamanho máximo da pilha
        private Object memo[];              //elementos da pilha
```

Object é uma classe geral de onde todas as classes herdam características. Portanto, pose-se guardar qualquer objeto dentro desse tipo, como Integer, Float, String etc. ou de outra classe qualquer.
Saiba mais no final deste material.

```
//Método construtor que inicializa a Pilha com um tamanho máximo desejado
public Pilha (int qtde)
{
        topo = -1;   // esta é a convenção que utilizaremos para "pilha vazia"
        MAX = qtde;
        memo = new Object[MAX];
}

//Método que verifica se a Pilha está Vazia
public boolean isEmpty ()
{
        return(topo==-1);
}

//Método que verifica se a Pilha está cheia
public boolean isFull ()
{
        return(topo==MAX-1);
}
```

```java
//Método para inserir um valor na Pilha
public Object push(Object x)
{
        if( !isFull() && x != null ) {
                memo[++topo]=x;
                return x;
        }
        else return null;
}
```

```java
//Método para retornar o topo da Pilha e remove-lo
public Object pop()
{
   if(!isEmpty())
      return memo[topo--];
   else
      return null;
}
```

Obs: os métodos mostrados retornam null nos casos de situações anormais (erradas). Outra abordagem seria que o método lance uma exception.

Observe a utilização correta do operador ++ pré-fixo e também do operador -- pós-fixo.

memo[++topo]=x;

é equivalente a:

topo = topo + 1;
memo[topo] = x;

return memo[topo--];

é equivalente a:

Object temp = memo[topo]
topo = topo - 1;
return temp;

```java
//Método que retorna o topo da pilha sem removê-lo
public Object top() {
    if(!isEmpty()) return memo[topo];
                   else return null;
}


//Método para retornar o conteúdo da Pilha
public String toString () {
        if(!isEmpty()) {
                String msg = "";
                for(int i=0; i<=topo; i++) {
                  msg += memo[i].toString();
                  if(i != topo) msg += ", ";
                }
                return ("P: [ " + msg + " ]");
        }
        else return "Pilha Vazia!";
}
```

Exemplo: operações com uma pilha

Universidade Cruzeiro do Sul

```
public static void main(String[] args) {
    Pilha p = new Pilha(10);
    p.push("Lima");
    p.push("Roma");
    p.push("Brasília");
    p.push("Washington");
    p.push("Havana");
    p.push("Buenos Aires");
    p.push("Bogotá");
    System.out.println("Pilha inicial:");
    System.out.println(p.toString());
    System.out.println("Retiremos dois elementos da pilha:");
    System.out.println(p.pop());
    System.out.println(p.pop());
    System.out.println(p.toString());
    System.out.println("Adicionamos Montevidéu:");
    p.push("Montevidéu");
    System.out.println(p.toString());
    System.out.println("No topo está: \n" + p.top() );
    System.out.println("Retirando e esvaziando a pilha:");
    while ( !p.isEmpty() ) {
       System.out.println(p.pop());
    }
    if ( p.isEmpty() ) System.out.println("Impossível retirar da pilha: Pilha vazia.");
  }
```

```
Pilha inicial:
P: [ Lima, Roma, Brasília, Washington, Havana, Buenos Aires, Bogotá ]
Retiremos dois elementos da pilha:
Bogotá
Buenos Aires
P: [ Lima, Roma, Brasília, Washington, Havana ]
Adicionamos Montevidéu:
P: [ Lima, Roma, Brasília, Washington, Havana, Montevidéu ]
No topo está:
Montevidéu
Retirando e esvaziando a pilha:
Montevidéu
Havana
Washington
Brasília
Roma
Lima
Impossível retirar da pilha. Pilha vazia.
```

Solução completa no projeto NetBeans PilhaEstaticaSequencial

Mais sobre pilhas e tipos de dados

- Uma estrutura de dados pode ser capaz de guardar diferentes tipos de dados;
- Podemos criar pilhas de objetos inteiros, de reais, de pratos, de alunos, de elementos, de veículos, de trabalhadores etc. e inclusive de objetos polimorfos;
- Em Java, todos as classes são herdeiras de uma classe chamada Object, como por exemplo as classes Float, Integer, String etc.;
- Portanto, um Object consegue armazenar uma referência para qualquer objeto (de qualquer classe).

Veja exemplo no projeto NetBeans **PilhaComObjetos**

Sobre conversões

- Podemos converter um objeto que guarda um número como String para número inteiro ou real (em versões mais recentes do JDK). Por exemplo:

```
Object x;
...
int v = (Integer)x;  //casting
```

- Outro exemplo. Um "casting" para a classe Trabalhador:

```
while ( !p.isEmpty()  ) {
        Trabalhador tr = (Trabalhador) p.pop();   //casting (porque pop retorna Object)
        System.out.println(tr.toString());
        if(tr.getSalario() > 4000.00f) {
            System.out.println("  -- O trabalhador anterior ganha mais de R$ 4.000,00");
        }
}
```

Converter objetos para String - exemplo

- Um dos métodos herdados da classe Object é a função **toString()**, que converte o conteúdo do objeto em String.
- Isso é útil para visualizar objetos e também para poder comparar elementos, visto que com o sinal de igual não é possível.

```
Object a, b;

if(a == b)... ;  //errado, porque só compararia as referências

//correto, mas também poderíamos comparar cada atributo:
if(a.toString().equals(b.toString()) {
   ...
}
```

Exercício proposto

Criar uma **pilha** com objetos da classe **Veículo**. Considere os atributos: placa, marca, modelo e ano de fabricação para a classe Veículo.

- Leia ou crie vários objetos de veículos e insira os mesmos na pilha.
- Retire os objetos da pilha e mostre seus dados.

Outro exercício proposto para praticar

Modifique o exercício anterior de forma a utilizar um **ArrayList** para guardar um grupo de objetos da classe Trabalhador em uma pilha, em lugar do vetor estático que foi utilizado na implementação mostrada neste material para a classe **Pilha**.

Bibliografia oficial da disciplina

BIBLIOGRAFIA BÁSICA	BIBLIOGRAFIA COMPLEMENTAR
CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. Algoritmos: teoria e prática. Rio de Janeiro: Elsevier, Campus, 2002. GOODRICH, Michael T.; TAMASSIA, Roberto. Estruturas de dados e algoritmos em Java. 2. ed. Porto Alegre, São Paulo: Bookman, 2002. SZWARCFITER, Jayme Luiz. Estruturas de dados e seus algoritmos. 3. Rio de Janeiro: LTC, 2010, recurso online, ISBN 978-85-216-2995-5.	ASCENCIO, A. F. G.; ARAÚJO, G. S. Estruturas de Dados. São Paulo: Pearson, 2011. [eBook] EDELWEISS, N.; GALANTE, T. Estruturas de Dados. Porto Alegre: Bookman, 2009. [eBook] MORIN, P. Open Data Structures (in Java) Creative Commons, 2011. Disponível em http://opendatastructures.org/ods-java.pdf [eBook] PUGA, S.; RISSETTI, G. Estruturas de Dados com aplicações em Java, 2a ed. São Paulo: Pearson, 2008. [eBook] SHAFFER, C. A.; Data Structures and Algorithm Analysis. Virginia Tech, 2012. Disponível em