Newton

Quem se prepara, não para.

Estrutura de Dados

Professora: Michelle Hanne Soares de Andrade
michelle.andrade@newtonpaiva.br

Newton
Quem se prepara, não para.

Sumário:

- ✓ Pilha (LIFO) – baseado em arranjos
- ✓ Fila (FIFO)

Implementação de Pilha baseada em Arranjos

Newton
Quem se prepara, não para.

Pode-se implementar uma pilha armazenando-se seus elementos em um arranjo. Mais especificamente, a pilha desta implementação consiste em um arranjo *S* de *N elementos* mais uma ***variável inteira t*** que fornece **o índice do elemento topo no arranjo S.**

Fonte: GOODRICH, 2013.

Implementação de Pilha baseada em Arranjos

Figura 5.2 Implementação de uma pilha por meio de um arranjo S. O elemento do topo de S está armazenado na célula $S[t]$.

Fonte: GOODRICH, 2013.

Implementação de Pilha baseada em Arranjos

Newton
Quem se prepara, não para.

```
Algoritmo size( ):
    retorna t + 1
Algoritmo isEmpty( ):
    retorna (t < 0)
Algoritmo top( ):
    se isEmpty( ) então
      lançar uma EmptyStackException
    retorna S[t]
Algoritmo push(e):
    se size( ) = N então
      lançar uma FullStackException
    t ← t + 1
    S[t] ← e
Algoritmo pop( ):
    se isEmpty( ) então
      lançar uma EmptyStackException
    e ← S[t]
    S[t] ← null
    t ← t − 1
    retorna e
```

Introduz-se um novo tipo de exceção chamada de **FullStackException**, que sinalizará uma condição de erro ao se tentar inserir um novo elemento em um arranjo cheio. **A exceção FullStackException** é específica para essa implementação e não está definida no TAD pilha

Fonte: GOODRICH, 2013.

Implementando uma pilha usando uma lista encadeada genérica

Newton
Quem se prepara, não para.

Nesta implementação será necessário decidir se o topo da pilha estará localizado na cabeça ou na cauda da lista. A melhor escolha, e mais eficiente, é na cabeça da lista, uma vez que se pode inserir e remover elementos em tempo constante apenas na cabeça.

Optou-se neste caso por implementar uma pilha genérica usando uma lista encadeada genérica.

Fonte: GOODRICH, 2013.

Implementando uma pilha usando uma lista encadeada genérica

```
public class Node<E> {
private E element; // Variáveis de instância
private Node<E> next;
/** Cria um nodo com referências nulas para os seus
elementos e o próximo nodo */
public Node() {
this(null, null);
}
/** Cria um nodo com um dado elemento e o próximo nodo */
public Node(E e, Node<E> n) {
element = e;
next = n;
}
// Métodos de acesso:
public E getElement() {
return element;
}
public Node<E> getNext() {
return next ;
}
// Métodos modificadores:
public void setElement(E newElem) {
element = newElem;
}
public void setNext(Node<E> newNext) {
next = newNext;
}
}
```

Newton
Quem se prepara, não para.

Fonte: GOODRICH, 2013.

Uma implementação Java de uma pilha, usando uma lista simplesmente encadeada genérica

Newton
Quem se prepara, não para.

O Algoritmo é eficiente em relação ao tempo, cuja necessidade de memória é O(n), em que *n* é o **número de elementos na pilha**. Assim, esta implementação não requer que uma nova exceção seja criada para lidar com **o problema de estouro do tamanho**. Usa-se uma variável de **instância, top,** para referenciar a cabeça da lista (que irá apontar para o objeto *null* se a lista estiver vazia).

Quando se insere um novo elemento e na pilha, simplesmente cria-se um novo *nodo v* para *e*, referencia-se e a partir de *v*, e insere-se **v** na cabeça da lista.

Da mesma forma, quando se retira um elemento da pilha, simplesmente remove-se o nodo da cabeça da lista e retorna-se seu elemento. Assim, executam-se todas as inserções e remoções de elementos na cabeça da lista.

Fonte: GOODRICH, 2013.

Deque

Newton
Quem se prepara, não para.

Considere-se agora uma estrutura de dados similar a uma fila que suporta inserção e remoção tanto em seu final, quanto em seu início. Essa extensão das filas é chamada de fila com dois finais ou deque.

O tipo abstrato de dados deque é mais rico do que os tipos **TAD pilha e fila**. Os métodos fundamentais para o TAD deque são os que seguem:

Fonte: GOODRICH, 2013.

Deque

Newton
Quem se prepara, não para.

addFirst(e): Insere um novo elemento e no começo do deque.
addLast(e): Insere um novo elemento e no final do deque.
removeFirst( ): Remove e retorna o primeiro elemento do deque; ocorre um erro se o deque estiver vazio.
removeLast( ): Remove e retorna o último elemento do deque; ocorre um erro se o deque estiver vazio.

Fonte: GOODRICH, 2013.

Deque

Newton
Quem se prepara, não para.

Adicionalmente, o TAD deque pode incluir os seguintes métodos auxiliares:

getfirst( ): Retorna o primeiro elemento do deque; ocorre um erro se o deque estiver vazio.

getlast( ): Retorna o último elemento do deque; ocorre um erro se o deque estiver vazio.

size( ): Retorna o número de elementos do deque

isEmpty( ): Determina se o deque está vazio.

Fonte: GOODRICH, 2013.

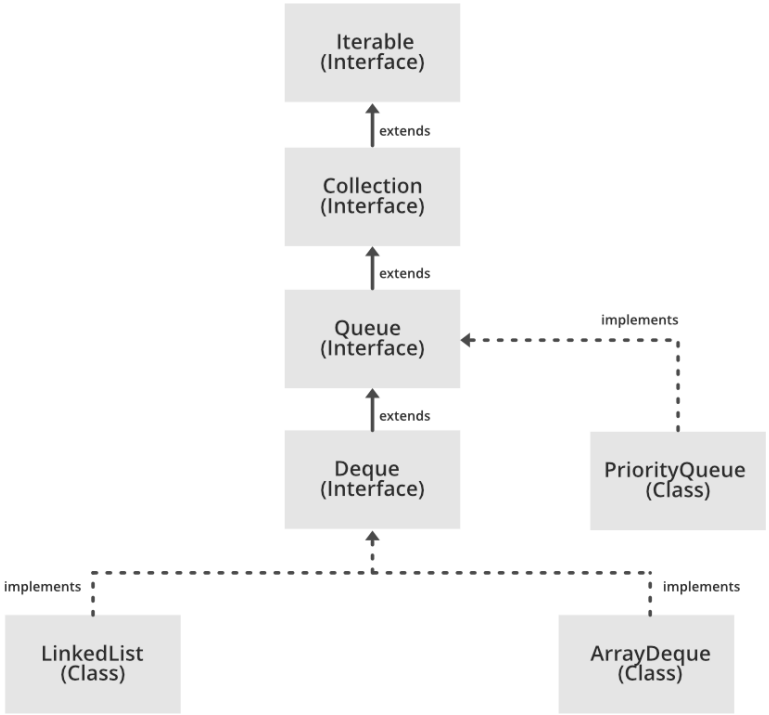
Deque

Exemplo 5.5 *A tabela a seguir mostra uma série de operações e seus efeitos em um deque D, inicialmente vazio, de objetos inteiros. Para simplificar, usam-se inteiros em vez de objetos inteiros como argumentos das operações.*

Operação	*Saída*	*D*
addFirst(3)	–	(3)
addFirst(5)	–	(5,3)
removeFirst( )	*5*	(3)
addLast(7)	–	(3,7)
removeFirst( )	*3*	(3)
removeLast( )	*7*	( )
removeFirst( )	*"error"*	( )
isEmpty( )	*true*	( )

Fonte: GOODRICH, 2013.

Deque



Declaração: A interface deque é declarada como:
interface pública Deque estende Queue

Como o Deque é uma interface , os objetos não podem ser criados do tipo deque. Sempre precisamos de uma classe que estenda essa lista para criar um objeto.

Fonte: https://acervolima.com/interface-deque-em-java-com-exemplo/

Referências

PINTO, Rafael A.; PRESTES, Lucas P.; SERPA, Matheus da S.; et al. Estrutura de dados. Editora SAGAH, 2020. ISBN 9786581492953. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786581492953

RODRIGUES, Thiago N.; LEOPOLDINO, Fabrício L.; PESSUTTO, Lucas Rafael C.; et al. Estrutura de Dados em Java. Editora SAGAH. 2021. ISBN 9786556901282. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786556901282

GOODRICH, Michael T.; TAMASSIA, Roberto. Estruturas de Dados e Algoritmos em Java. [Digite o Local da Editora]: Grupo A, 2013. E-book. ISBN 9788582600191. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582600191/. Acesso em: 13 set. 2022.