Fundamentos de Listas Lineares

Estrutura de Dados

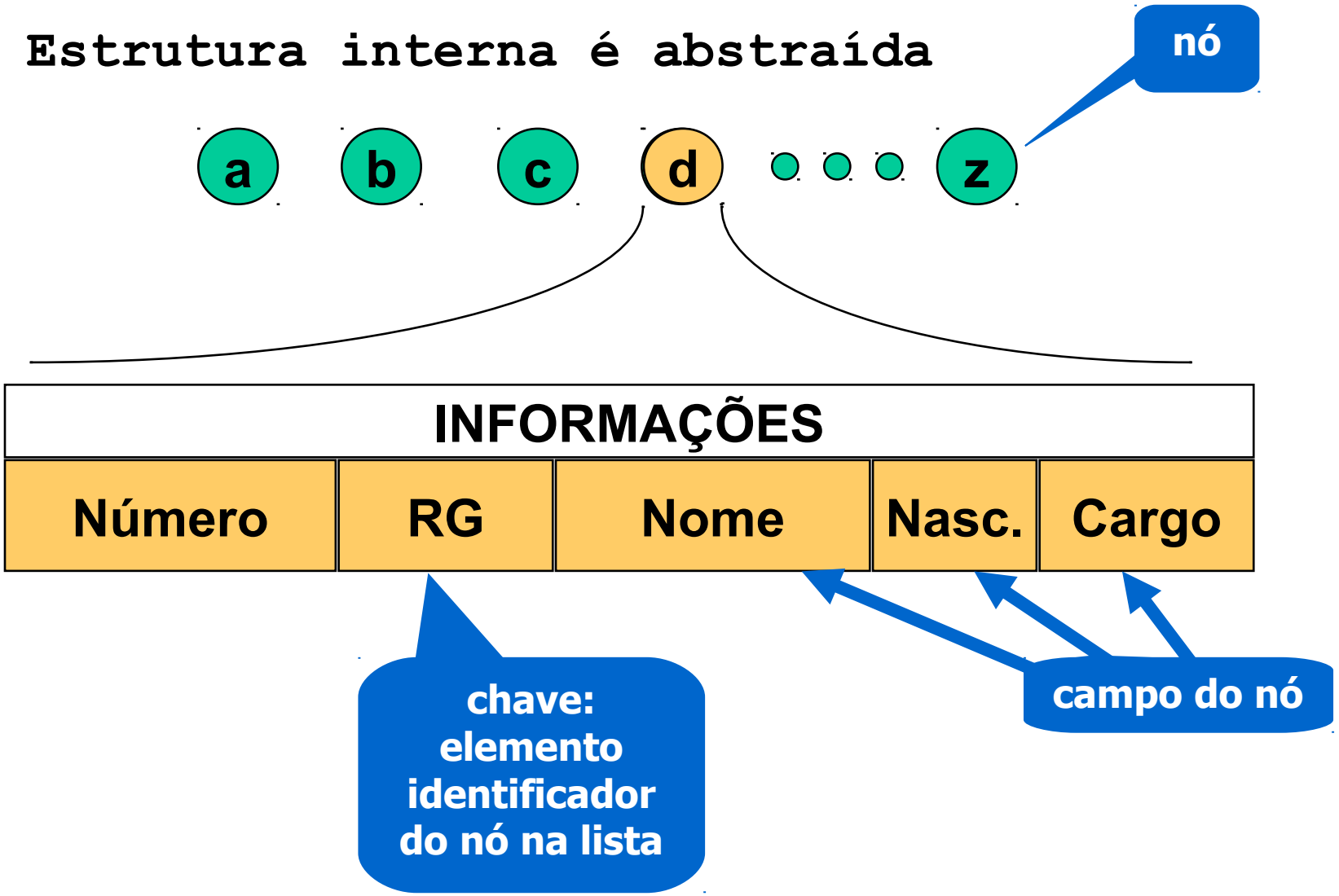
- Para implementar um TAD, numa linguagem de programação, é necessário encontrar uma forma de representá-los nessa linguagem utilizando tipos e operações suportadas pelo computador.
- Estrutura de Dados (ED) = materialização do TAD
  - **Em Java, ED é modelado por uma classe.**
    - **Dados em TAD: representado por variáveis na classe**
    - **Operações em TAD: representados por método na classe.**
- Assim, um mesmo tipo abstrato de dados pode ser concretizado (ou implementado) de diversas formas.
- TADs são materializados pela estruturas de dados.
  - **Lista Encadeada (implementação com referências)**
  - **Lista com alocação estática (implementação usando array)**

Listas Lineares

- A Lista Linear é a estrutura que permite representar um conjunto de dados de forma a preservar a relação de ordem existente entre eles.
- Uma lista linear, ou tabela, é um conjunto de n>= 0 nós L[1], L[2], ..., L[n] , onde:
  - **Se n>0, L[1] é o primeiro nó,**
  - **Para 1 < k <= n, o nó L[k] é precedido por L[k-1].**
- Sequência Linear de dados

Estrutura dos nós

- Estrutura interna é abstraída



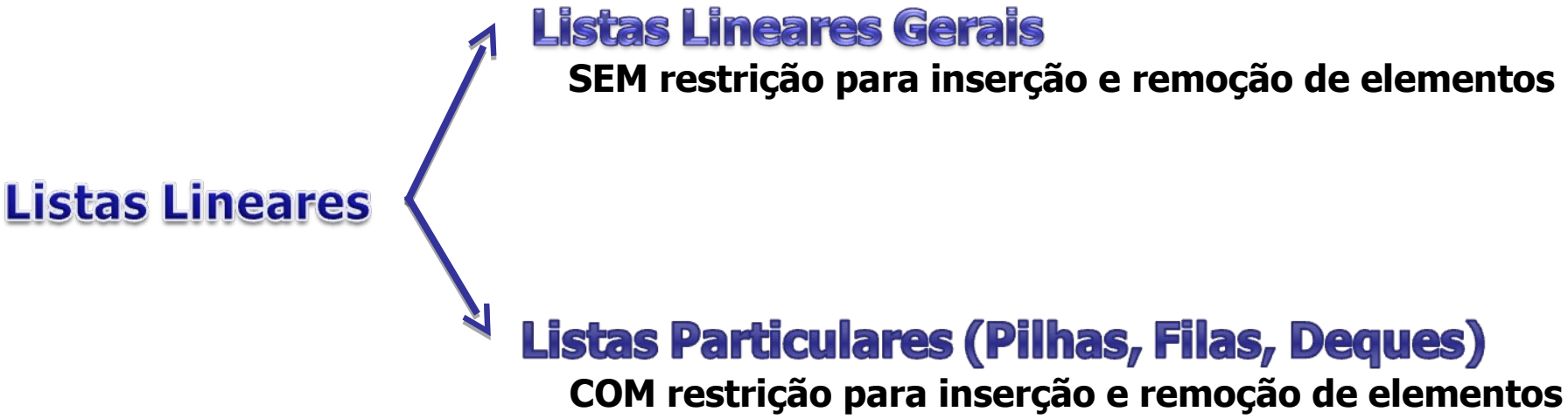
Exemplos de aplicações com listas

- notas de alunos
- cadastro de funcionários de uma empresa
- itens em estoque em uma empresa
- dias da semana
- vagões de um trem
- letras de uma palavra
- pessoas esperando ônibus
- cartas de baralho
- precipitações pluviométricas em um mês / dia

Listas: Tipo de Armazenamento

- O tipo de armazenamento de uma lista linear pode ser classificado de acordo com a posição relativa (sempre contígua ou não) na memória de dois nós consecutivos na lista.
  - **Lista linear com alocação estática de memória**
    - **Também chamada de Lista Sequencial**
    - **Nós em posições contíguas de memória**
    - **Geralmente representado por arrays**
    - **Útil para implementar filas e pilhas (variáveis para controlar fim e início)**
  - **Lista linear com alocação dinâmica de memória**
    - **Também chamada de Lista Encadeada**
    - **Posições de memória são alocadas a medida que são necessárias**
    - **Nós encontram-se aleatoriamente dispostos na memória e são interligados por ponteiros, que indicam a próxima posição da tabela**
      - **Nós precisam de um campo a mais: campo que indica o endereço do próximo nó.**

Listas Lineares: Classificação



Listas Lineares Gerais

- Listas lineares gerais:
  - **a inclusão e remoção de elementos pode ser realizada em qualquer posição da lista.**
- Essas listas não apresentam nenhuma restrição de acesso.
  - **podendo sofrer inserções ou retiradas em qualquer lugar, inclusive no meio da lista.**
  - **Ex: lista de chamada dos alunos.**
- As listas gerais podem ser ordenadas ou não ordenadas.
  - **Lista Ordenada: os nós encontram-se ordenados segundo os valores de suas chaves**
  - **Lista Não Ordenada: os nós não estão ordenados**

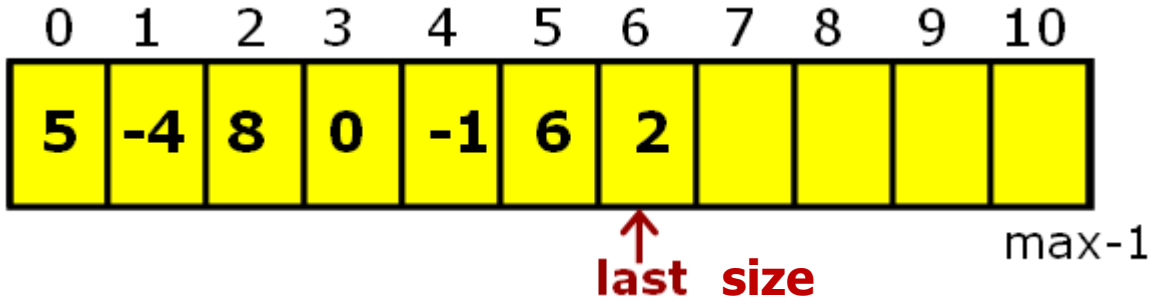
Listas Lineares Sequenciais

Lista linear geral com alocação estática

- Também chamada de Lista Sequencial
- Suponhamos uma lista geral de números inteiros L que, em certo momento, possui os seguintes 7 elementos:

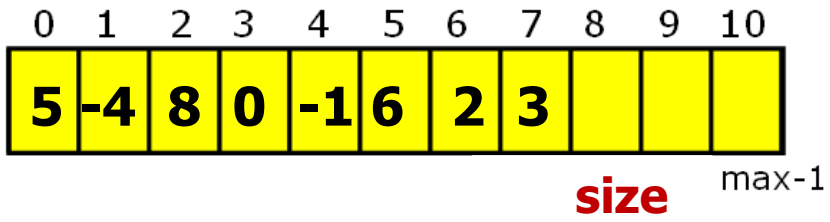
  5 -4 8 0 -1 6 2

- Declarando um vetor de inteiros de nome L[0..Max-1], disporemos a lista nas primeiras posições do vetor, de modo que o primeiro nó da lista ocupe a posição 0 do vetor, e indicaremos seu término por uma variável inteira de nome size que indica a posição do último elemento. Teremos

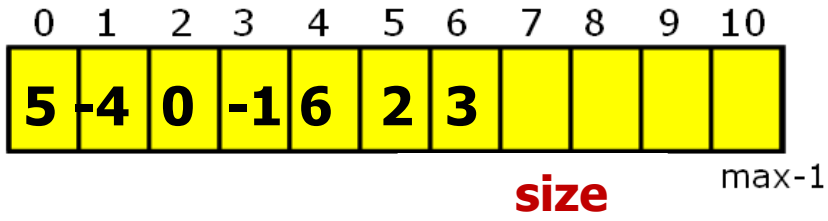


Exercício: Lista com alocação estática

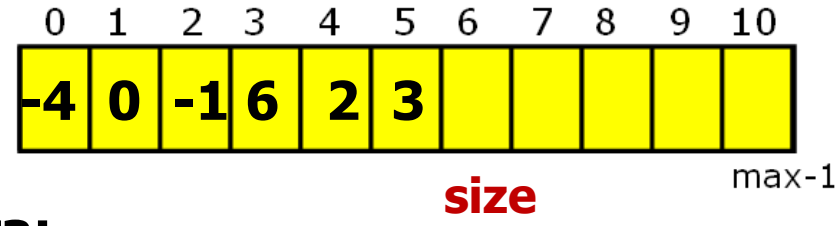
- **Inserir 3 no fim da lista:**



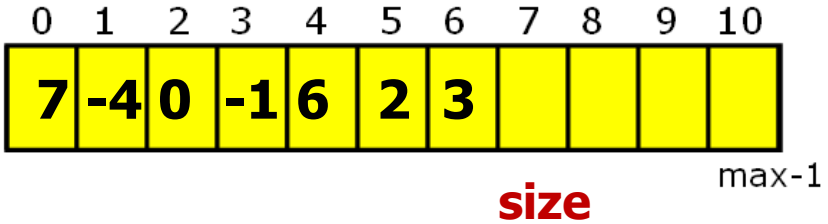
- **Retirar 8:**



- **Retirar o 1º elemento:**



- **Inserir 7 no início da lista:**



Operações implementadas sobre listas lineares gerais

```
public interface IndexListObject {
  /** Returns the number of elements in this list. */
  public int size();
  /** Returns whether the list is empty. */
  public boolean isEmpty();
  /** Inserts an element e to be at index i, shifting all elements
   after this. */
  public void add (int i, Object e)
              throws IndexOutOfBoundsException;
  /** Returns the element at index i, without removing it. */
  public Object get (int i)
               throws IndexOutOfBoundsException;
  /** Removes and returns the element at index i, shifting the
   elements after this. */
  public Object remove (int i)
               throws IndexOutOfBoundsException;
  /** Replaces the element at index i with e, returning the
   previous element at i. */
  public Object set (int i, Object e)
               throws IndexOutOfBoundsException;
}
```

Listas Sequenciais

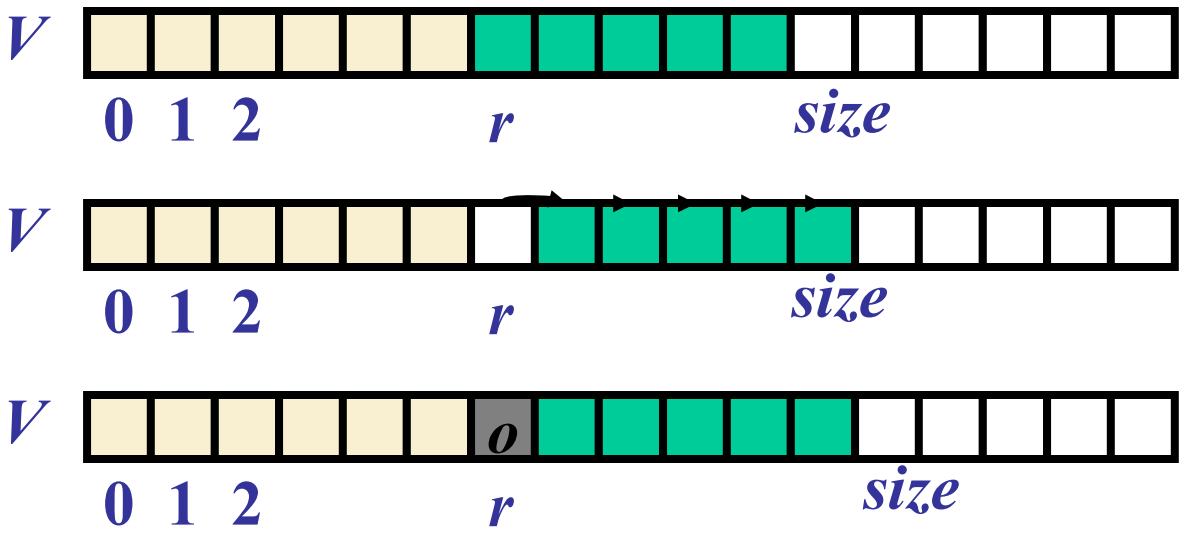
```
public class ArrayIndexListObject implements IndexListObject {
  private Object[] A;  // array storing the elements of the indexed list
  private int capacity = 16;  // initial length of array A
  private int size = 0; // number of elements stored in the indexed list

  /** Creates the indexed list with initial capacity 16. */
  public ArrayIndexListObject() {
    A = new Object[capacity];
  }
  /** Returns the number of elements in the indexed list. */
  public int size() {
    return size;
  }
  /** Returns whether the indexed list is empty. */
  public boolean isEmpty() {
    return size() == 0;
  }
```

Lista geral com alocação estática

```
/** Returns the element stored at the given index. */
 public Object get(int r) throws IndexOutOfBoundsException {
   checkIndex(r, size());
   return A[r];
 }
 /** Replaces the element stored at the given index. */
 public Object set(int r, Object e)
         throws IndexOutOfBoundsException {
   checkIndex(r, size());
   Object temp = A[r];
   A[r] = e;
   return temp;
 }
```

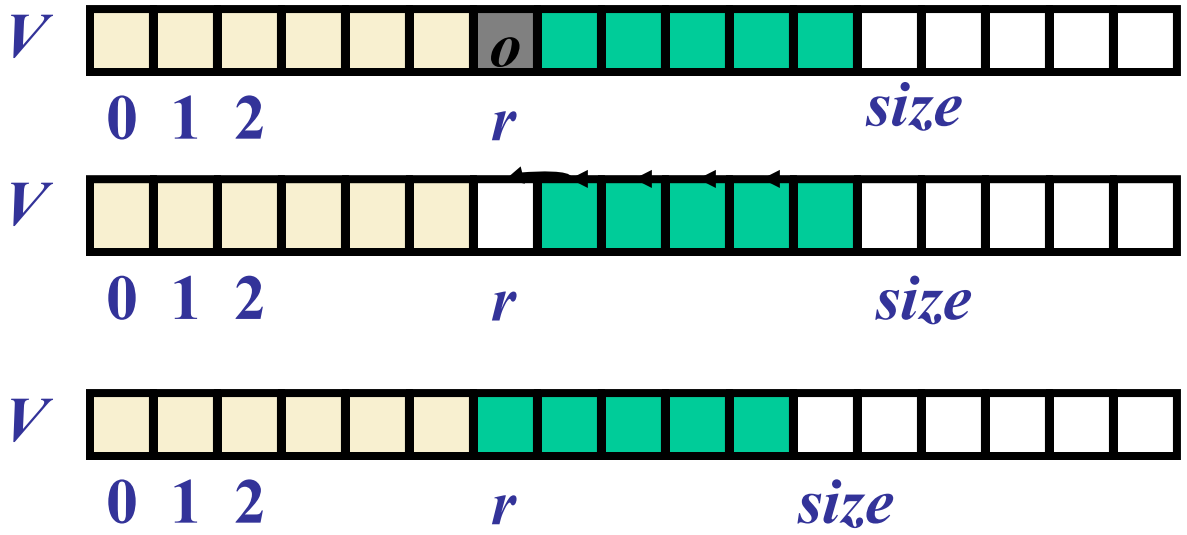
Inserção



Lista geral com alocação estática

```
  /** Inserts an element at the given index. */
public void add(int r, Object e)
  throws IndexOutOfBoundsException {
  checkIndex(r, size() + 1);
  if (size == capacity) {// an overflow
    capacity *= 2;
    Object[] B = new Object[capacity];
    for (int i=0; i<size; i++)
        B[i] = A[i];
    A = B;
  }
  for (int i=size-1; i>=r; i--)// shift elements up
    A[i+1] = A[i];
  A[r] = e;
  size++;
}
```

Remoção



Lista geral com alocação estática

```
/** Removes the element stored at the given
 index. */
 public Object remove(int r)
   throws IndexOutOfBoundsException {
   checkIndex(r, size());
   Object temp = A[r];
   for (int i=r; i<size-1; i++)// shift elements down
     A[i] = A[i+1];
   size--;
   return temp;
 }
```

Lista geral com alocação estática

```
  /** Checks whether the given index is in the range
  [0, n - 1] */
  protected void checkIndex(int r, int n) //
            throws IndexOutOfBoundsException {//
    if (r < 0 || r >= n)
      throw new IndexOutOfBoundsException("Illegal
  index: " + r);
  }
}
```

Exercícios

- Exercício 1. Crie a classe Aluno, representando um aluno que conterá um nome (String) e uma nota (double).
  - **Adicione métodos de acesso e modificação para os atributos da classe.**
  - **Sobrescreva nesta classe o método toString da classe Object.**
- No método main de uma classe qualquer, crie uma lista linear geral (implementação dada em aula). Crie vários objetos alunos e o insira na lista.
  - **Exercício 2. Na classe ArrayIndexListObject, sobrescreva nesta classe o método toString da classe Object.**

Estouro das Listas

- `Estouro de listas:`
  - **Estouro negativo (underflow): lista vazia sofre operação de extração**
  - **Estouro positivo (overflow): quando a inserção de um elemento excede a capacidade total da lista.**

Lista geral com alocação estática

```
  /** Inserts an element at the given index. */
public void add(int r, Object e)
  throws IndexOutOfBoundsException {
  checkIndex(r, size() + 1);
  if (size == capacity) {// an overflow
    capacity *= 2;
    Object[] B = new Object[capacity];
    for (int i=0; i<size; i++)
        B[i] = A[i];
    A = B;
  }
  for (int i=size-1; i>=r; i--)// shift elements up
    A[i+1] = A[i];
  A[r] = e;
  size++;
}
```

Lista Linear Geral: Interface

```
public interface IndexList<E> {
  /** Returns the number of elements in this list. */
  public int size();
  /** Returns whether the list is empty. */
  public boolean isEmpty();
  /** Inserts an element e to be at index i, shifting all elements
   after this. */
  public void add(int i, E e)
    throws IndexOutOfBoundsException;
  /** Returns the element at index i, without removing it. */
  public E get(int i)
    throws IndexOutOfBoundsException;
  /** Removes and returns the element at index i, shifting the elements
   after this. */
  public E remove(int i)
    throws IndexOutOfBoundsException;
  /** Replaces the element at index i with e, returning the previous
   element at i. */
  public E set(int i, E e)
    throws IndexOutOfBoundsException;
}
```

Lista Linear Geral com Tipos Genéricos (1)

```
public class ArrayIndexList<E> implements IndexList<E> {
  private E[] A;       // array storing the elements of the
   indexed list
  private int capacity = 16;  // initial length of array A
  private int size = 0;       // number of elements stored in
   the indexed list
  /** Creates the indexed list with initial capacity 16. */
  public ArrayIndexList() {
    A = (E[]) new Object[capacity];
  }
  /** Returns the number of elements in the indexed list. */
  public int size() {
    return size;
  }
  /** Returns whether the indexed list is empty. */
  public boolean isEmpty() {
    return size() == 0;
  }
```

Lista Linear Geral com Tipos Genéricos (2)

```
/** Returns the element stored at the given index. */
  public E get(int r)
    throws IndexOutOfBoundsException {
    checkIndex(r, size());
    return A[r];
  }
  /** Replaces the element stored at the given index. */
  public E set(int r, E e)
    throws IndexOutOfBoundsException {
    checkIndex(r, size());
    E temp = A[r];
    A[r] = e;
    return temp;
  }
```

Lista Linear Geral com Tipos Genéricos (3)

```
/** Inserts an element at the given index. */
 public void add(int r, E e)
   throws IndexOutOfBoundsException {
   checkIndex(r, size() + 1);
   if (size == capacity) {// an overflow
     capacity *= 2;
     E[] B =(E[]) new Object[capacity];
     for (int i=0; i<size; i++)
         B[i] = A[i];
     A = B;
   }
   for (int i=size-1; i>=r; i--)// shift elements up
     A[i+1] = A[i];
   A[r] = e;
   size++;
 }
```

Lista Linear Geral com Tipos Genéricos (4)

```
 /** Removes the element stored at the given index. */
  public E remove(int r)
    throws IndexOutOfBoundsException {
    checkIndex(r, size());
    E temp = A[r];
    for (int i=r; i<size-1; i++)// shift elements down
      A[i] = A[i+1];
    size--;
    return temp;
  }
/** Checks whether the given index is in the range [0, n - 1] */
  protected void checkIndex(int r, int n) //
    throws IndexOutOfBoundsException {//
    if (r < 0 || r >= n)
      throw new IndexOutOfBoundsException("Illegal index: " +
   r);
  }
}
```

Exercícios

- **Exercício 3. Repetir exercício 1, mas usando a classe ArrayIndexList com tipos genéricos.**

Referências Bibliográficas

- GOODRICH, MICHAEL T. ; TAMASSIA, ROBERTO. Estruturas de dados e algoritmos em Java. 4.ed. Bookman. 2007.