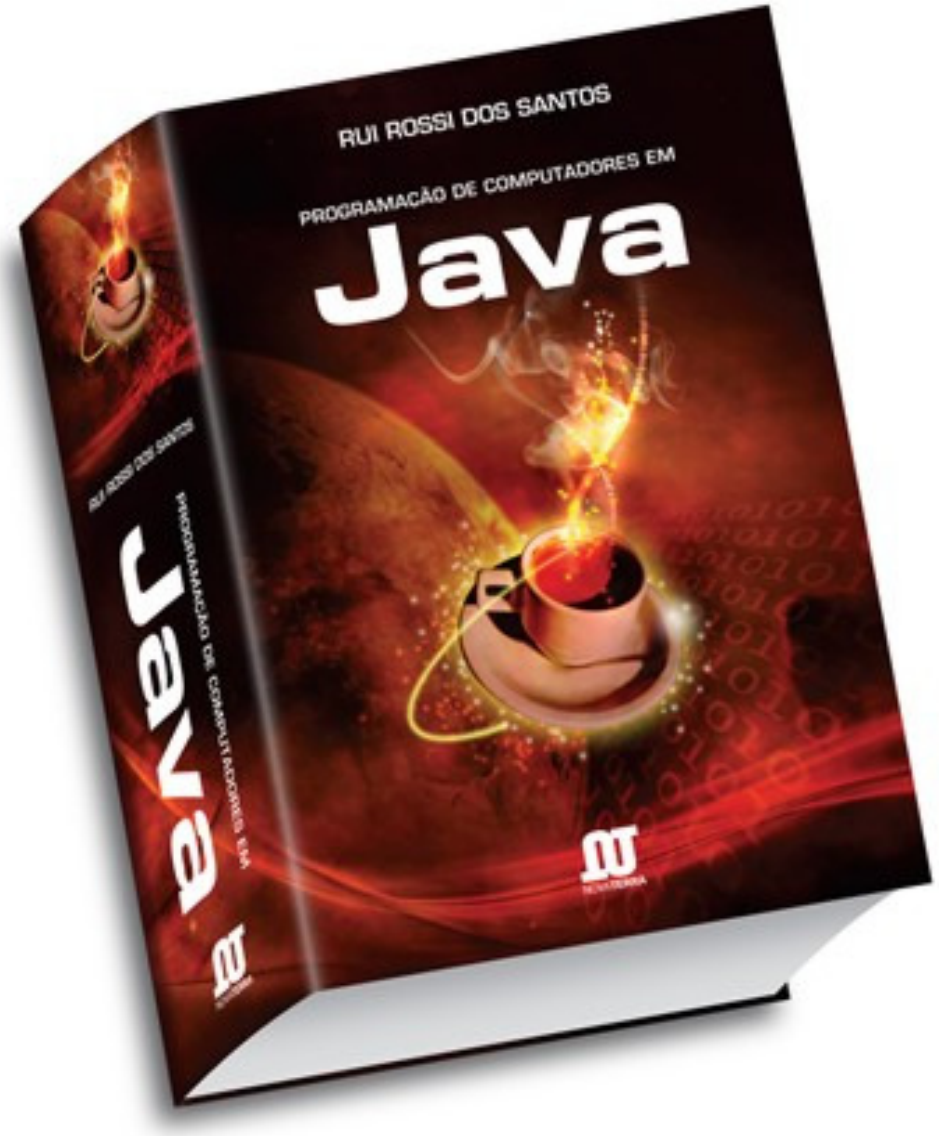


Capítulo 20

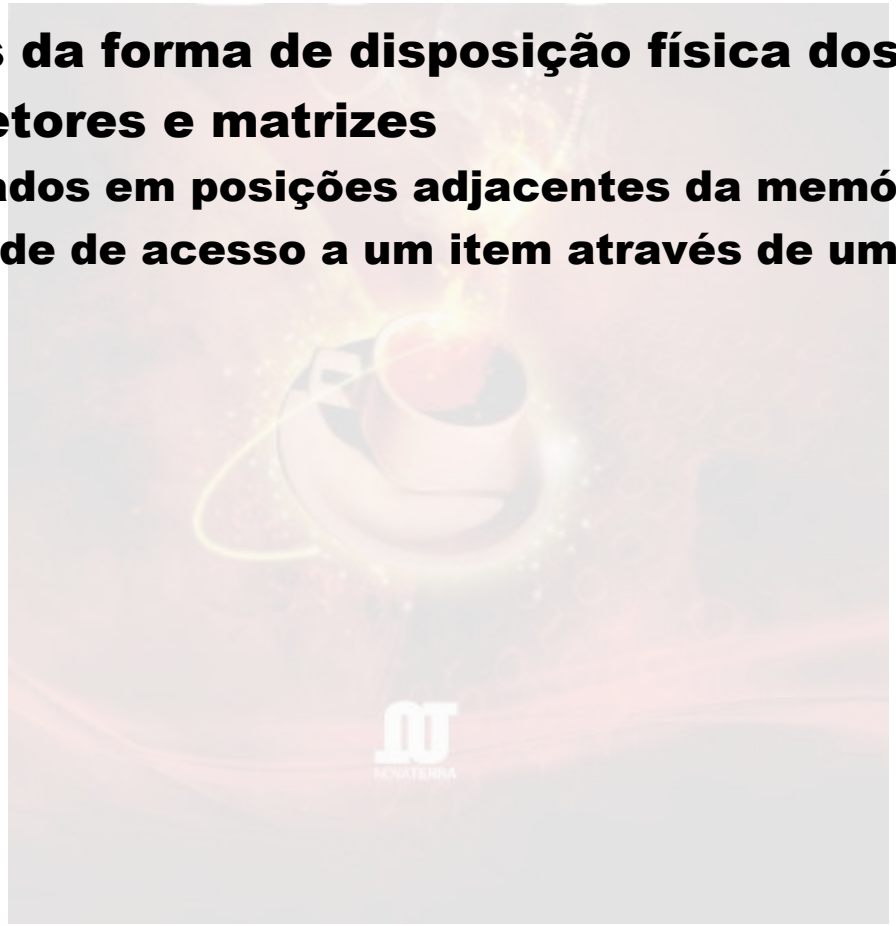
Estruturas de Dados Dinâmicas

Objetivos do Capítulo

- Caracterizar as estruturas de dados dinâmicas.
- Apresentar implementações para três diferentes estruturas de dados: as pilhas, as filas e as listas encadeadas.
- Demonstrar como as classes genéricas podem ser empregadas para a construção de estruturas de dados mais sofisticadas e versáteis.
- Utilizar as listas encadeadas como mecanismo de armazenamento na construção de pilhas e de filas dinâmicas.

Introdução

- **Estruturas de dados estáticas**
  - **Capacidade fixa**
  - **Dependentes da forma de disposição física dos itens**
  - **Exemplos: vetores e matrizes**
    - ✓ **Itens gravados em posições adjacentes da memória**
    - ✓ **Possibilidade de acesso a um item através de um índice.**



Introdução

- **TAD (Tipo Abstrato de Dado)**
  - **Não depende da forma de disposição física dos itens**
    - ✓ **Podem ser construídas de diferentes maneiras**
  - **Exemplos:**
    - ✓ **Pilhas**
    - ✓ **Filas**
    - ✓ **Listas**
  - **As listas são estruturas de dados dinâmicas**
    - ✓ **A alocação de memória para cada item inserido é realizada em tempo de execução.**
  - **As pilhas e filas podem ser estáticas ou dinâmicas**
    - ✓ **Itens de uma pilha ou fila podem ser gravados em um vetor**
      - o **O uso de um vetor as torna estruturas de dados estáticas**
    - ✓ **Itens de uma pilha ou fila podem ser gravados em uma lista**
      - o **O uso de uma lista as torna uma estrutura de dados dinâmica**

Pilhas Estáticas

- **Visão geral**
  - Conjunto de objetos que se encontram uns sobre os outros
  - Exemplos: pratos, livros e correspondências.

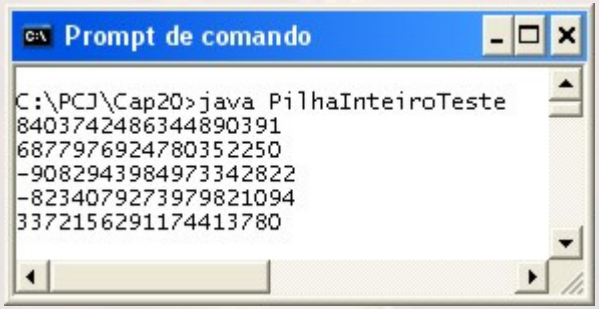
- **Ações:**
  - Empilhar: inserir um item na pilha.
  - Desempilhar: remover um item da pilha.

- **Funcionamento:**
  - Metodologia LIFO (Last In, First Out)
  - Um novo item só pode ser inserido no topo da pilha
  - Somente o item do topo da pilha pode ser removido

Pilhas Estáticas

- **Código 20.1 – PilhaInteiro.java**

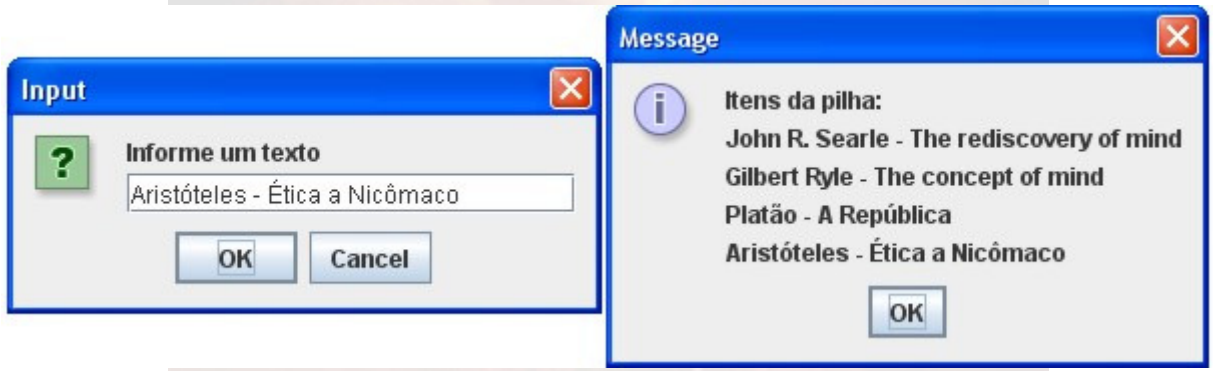
- **Código 20.2 – PilhaInteiroTeste.java**



Pilhas Estáticas

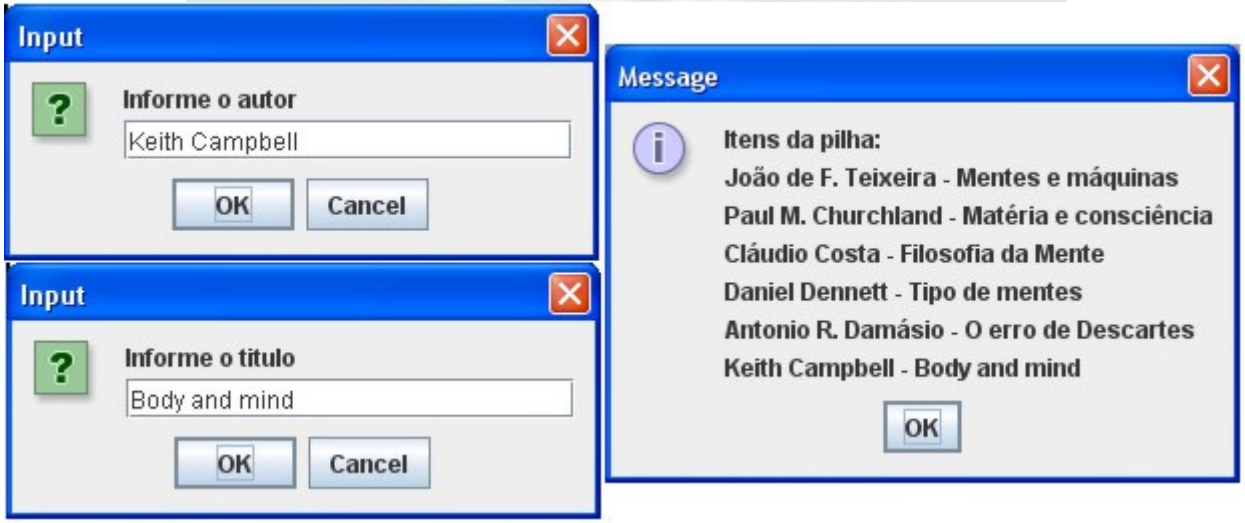
- **Código 20.3 – PilhaGenerica.java**

- **Código 20.4 – PilhaStringTeste.java**



Pilhas Estáticas

- Código 20.3 – PilhaGenerica.java
- Código 20.5 – Livro.java
- Código 20.6 – PilhaLivroTeste.java



Filas Estáticas

- **Visão geral**
  - **Conjunto de objetos que se encontram uns atrás dos outros**
  - **Exemplos: pessoas no banco e carros no semáforo.**

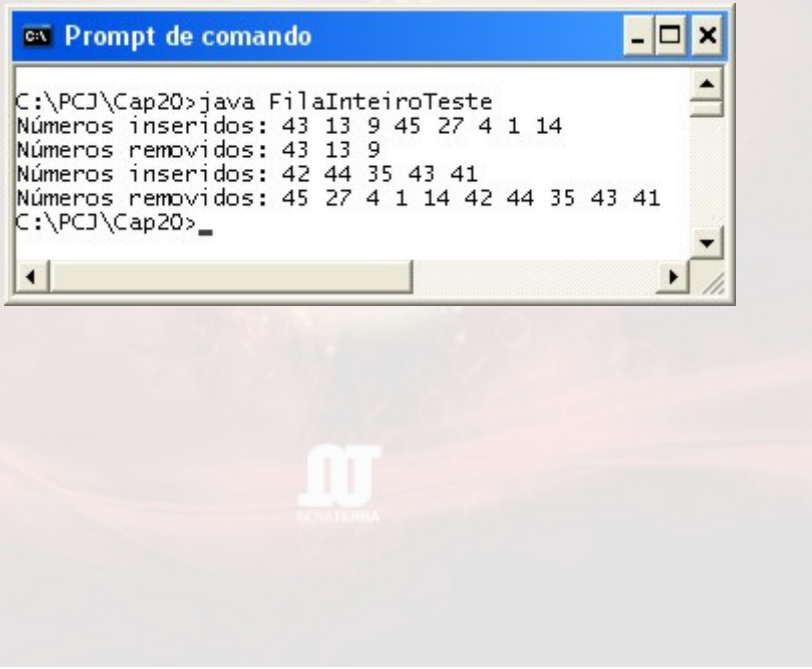
- **Ações:**
  - **Incluir um item na fila.**
  - **Retirar um item da fila.**

- **Funcionamento:**
  - **Metodologia FIFO (First In, First Out)**
  - **Um novo item só pode ser inserido no final da fila**
  - **Somente o item do início da fila pode ser retirado**

Filas Estáticas

- **Código 20.7 – FilaInteiro.java**

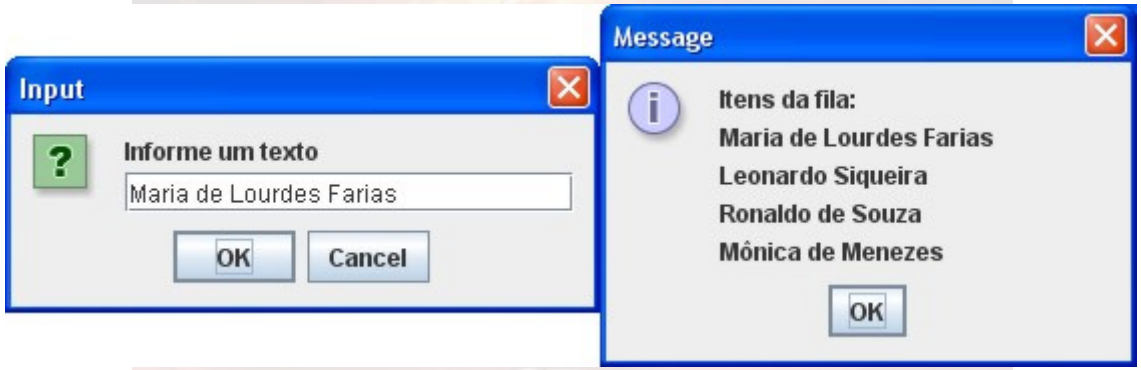
- **Código 20.8 – FilaInteiroTeste.java**



Filas Estáticas

- **Código 20.9 – FilaGenerica.java**

- **Código 20.10 – FilaStringTeste.java**

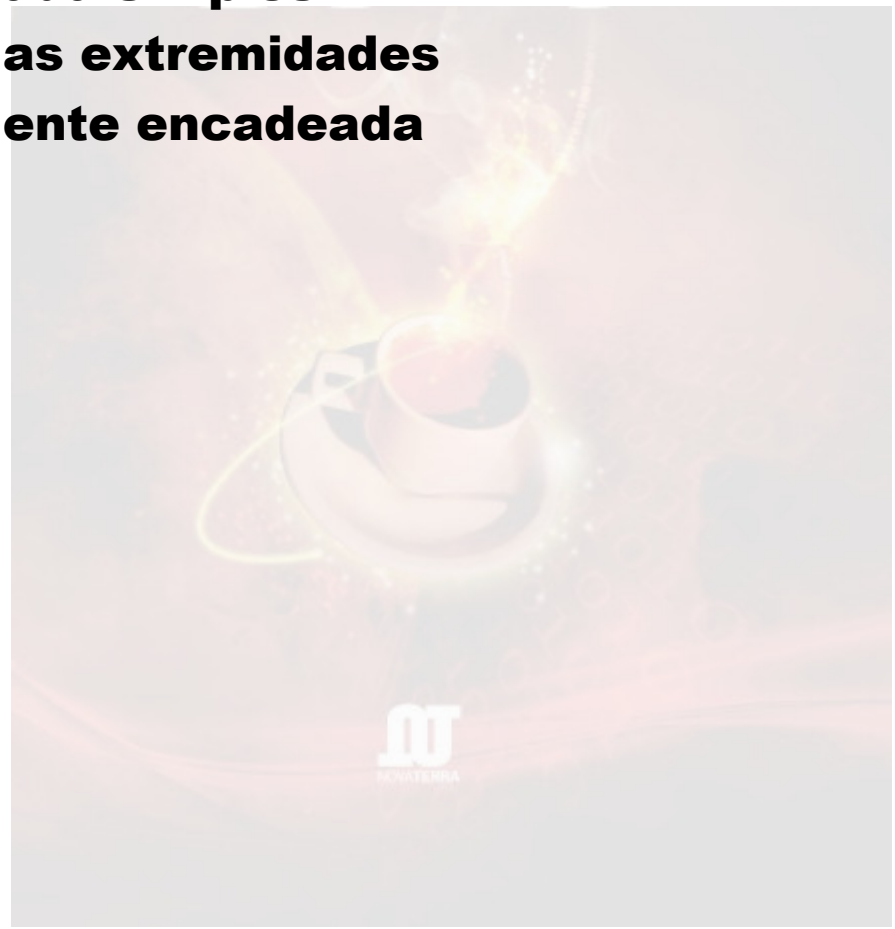


Listas Encadeadas

- **Visão geral**
  - Conjunto de objetos organizados de forma linear.
  - Objetos vinculados entre si através de **referências**.
  - Cada objeto da lista é chamada de **nó** ou **nodo**.
  - Cada nó pode manter:
    - ✓ Uma referência para o nó seguinte.
    - ✓ Uma referência para o nó anterior.

Listas Encadeadas

- **Tipos**
  - **Lista encadeada simples**
  - **Lista com duas extremidades**
  - **Lista duplamente encadeada**



Listas Encadeadas

- **Lista encadeada simples**
  - **Criar uma classe para representar cada nó.**
    - ✓ **Um atributo para armazenar o dado (tipo primitivo ou objeto).**
    - ✓ **Um atributo para manter a referência ao nó posterior.**
  - **Criar uma classe para representar a lista.**
    - ✓ **Um atributo para manter a referência para o primeiro nó.**
  - **Navegação:**
    - ✓ **Acesso inicial: primeiro nó.**
    - ✓ **Navegação unidirecional: do primeiro para o último nó.**

Listas Encadeadas

- **Lista com duas extremidades**
  - **Criar uma classe para representar cada nó.**
    - ✓ **Um atributo para armazenar o dado (tipo primitivo ou objeto).**
    - ✓ **Um atributo para manter a referência ao nó posterior.**
  - **Criar uma classe para representar a lista.**
    - ✓ **Um atributo para manter a referência para o primeiro nó.**
    - ✓ **Um atributo para manter a referência para o último nó.**
  - **Navegação:**
    - ✓ **Acesso inicial: primeiro nó ou último nó.**
    - ✓ **Navegação unidirecional: do primeiro para o último nó.**

Listas Encadeadas

❑ Lista duplamente encadeada

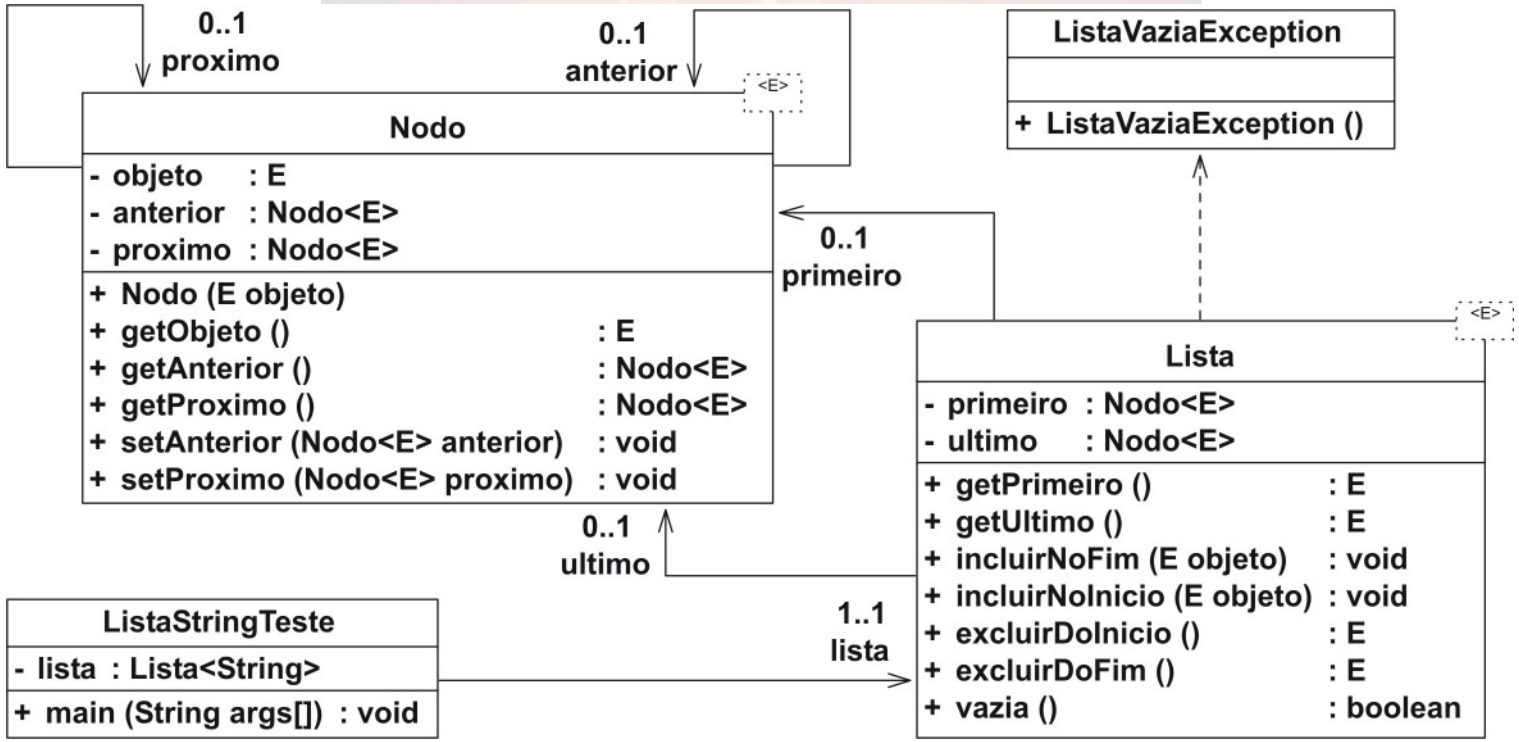
- ➢ **Criar uma classe para representar cada nó.**
  - ✓ **Um atributo para armazenar o dado (tipo primitivo ou objeto).**
  - ✓ **Um atributo para manter a referência ao nó posterior.**
  - ✓ **Um atributo para manter a referência ao nó anterior.**

- ➢ **Criar uma classe para representar a lista.**
  - ✓ **Um atributo para manter a referência para o primeiro nó.**
  - ✓ **Um atributo para manter a referência para o último nó.**

- ➢ **Navegação:**
  - ✓ **Acesso inicial: primeiro nó ou último nó.**
  - ✓ **Navegação bidirecional:**
    - o **Do primeiro para o último nó.**
    - o **Do último para o primeiro nó.**

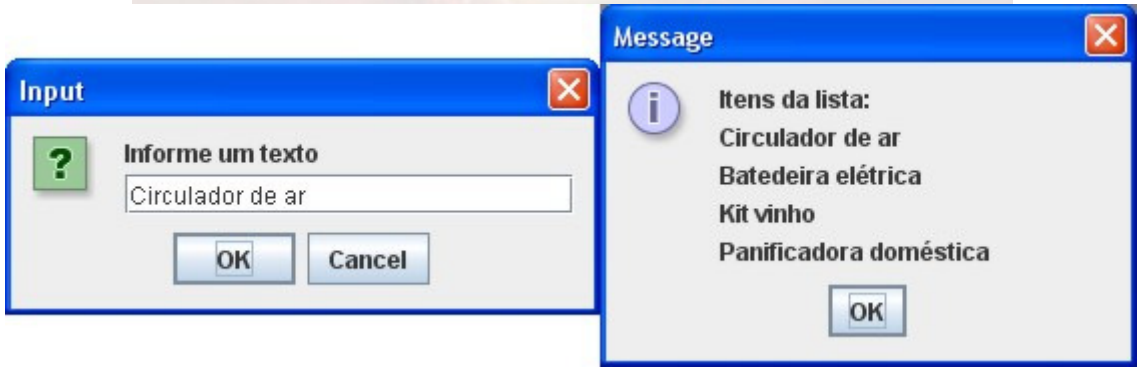
Listas Encadeadas

- **Estrutura de componentes:**
  - **Nodo: representa cada nó (classe genérica auto-referencial)**
  - **Lista: representa uma lista duplamente encadeada (genérica)**
  - **ListaVaziaException: exceção (excluir item de lista vazia)**
  - **ListaStringTeste: aplicativo que demonstra o uso da lista**



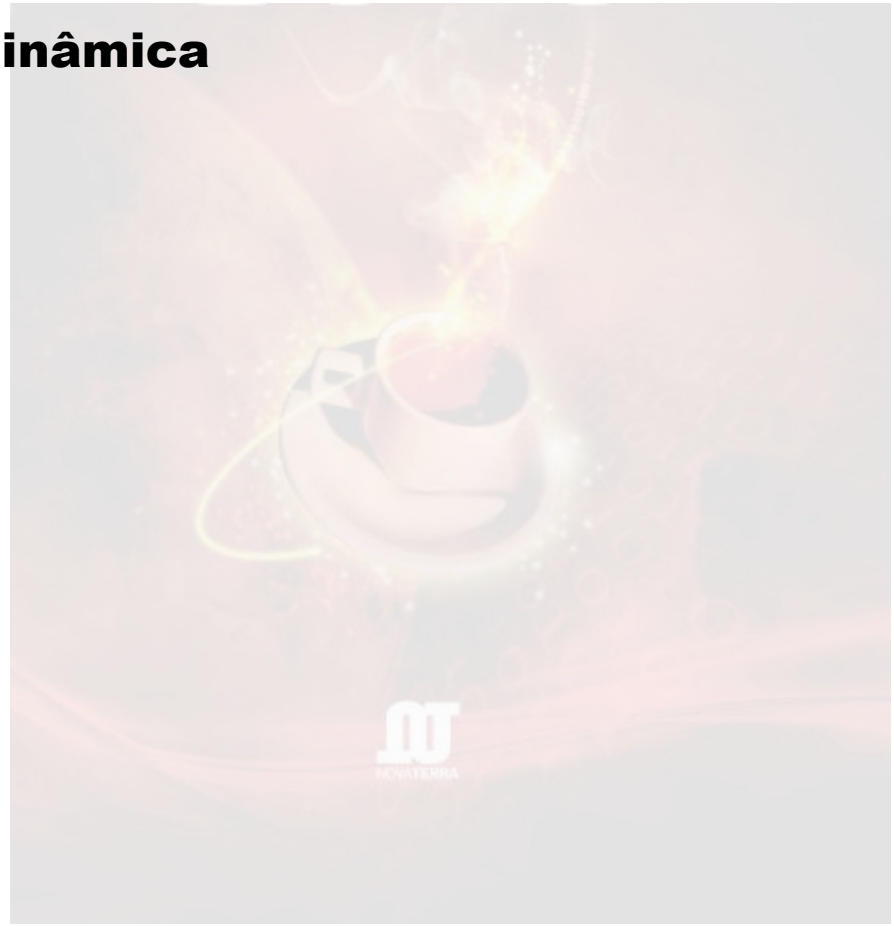
Listas Encadeadas

- Código 20.11 – Nodo.java
- Código 20.12 – ListaVaziaException.java
- Código 20.13 – Lista.java
- Código 20.14 – ListaStringTeste.java



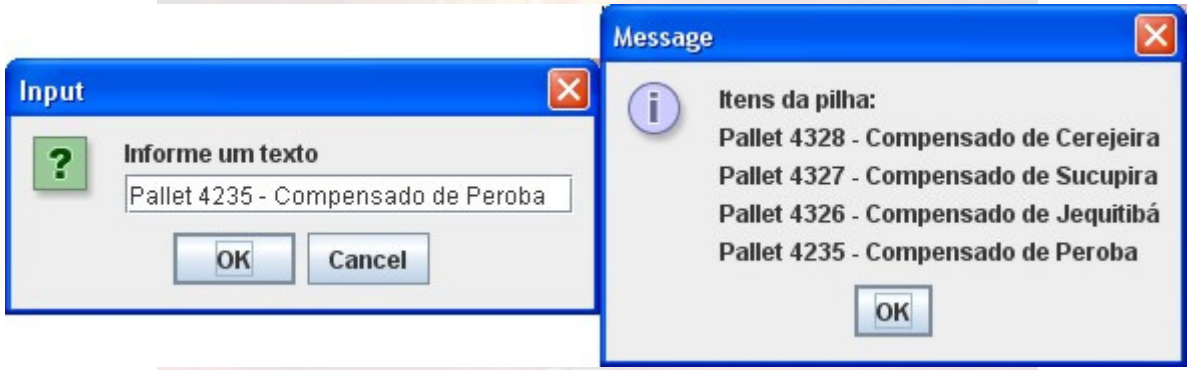
Pilhas Dinâmicas

- **Mecanismo de armazenamento para pilhas:**
  - **Vetor: pilha estática**
  - **Lista: pilha dinâmica**



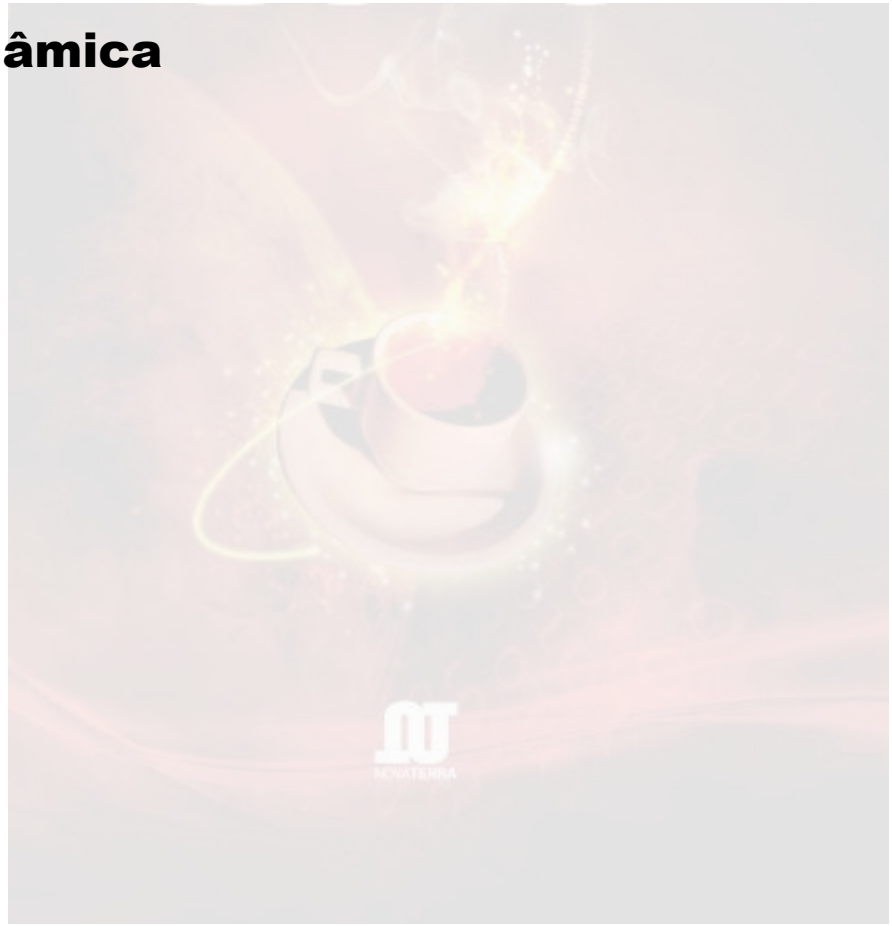
Pilhas Dinâmicas

- **Código 20.15 – PilhaDinâmica.java**
- **Código 20.16 – PilhaDinâmicaTeste.java**



Filas Dinâmicas

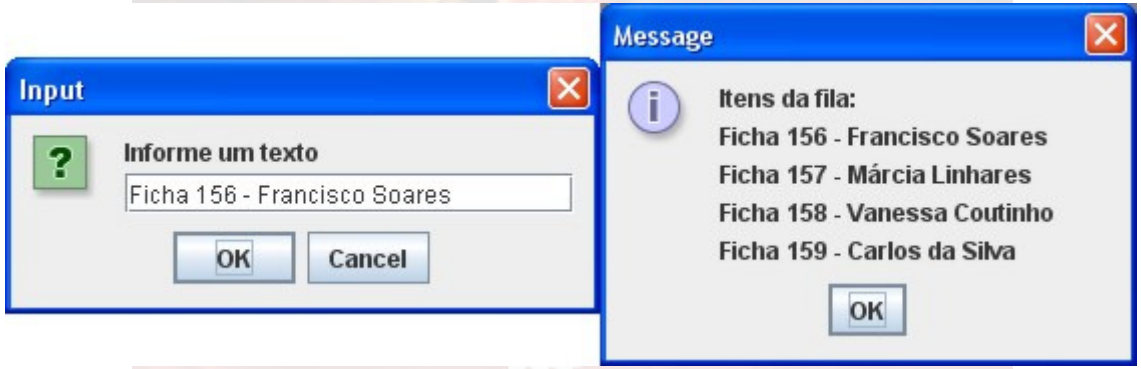
- **Mecanismo de armazenamento para filas:**
  - **Vetor: fila estática**
  - **Lista: fila dinâmica**



Filas Dinâmicas

- **Código 20.17 – FilaDinâmica.java**

- **Código 20.18 – FilaDinâmicaTeste.java**

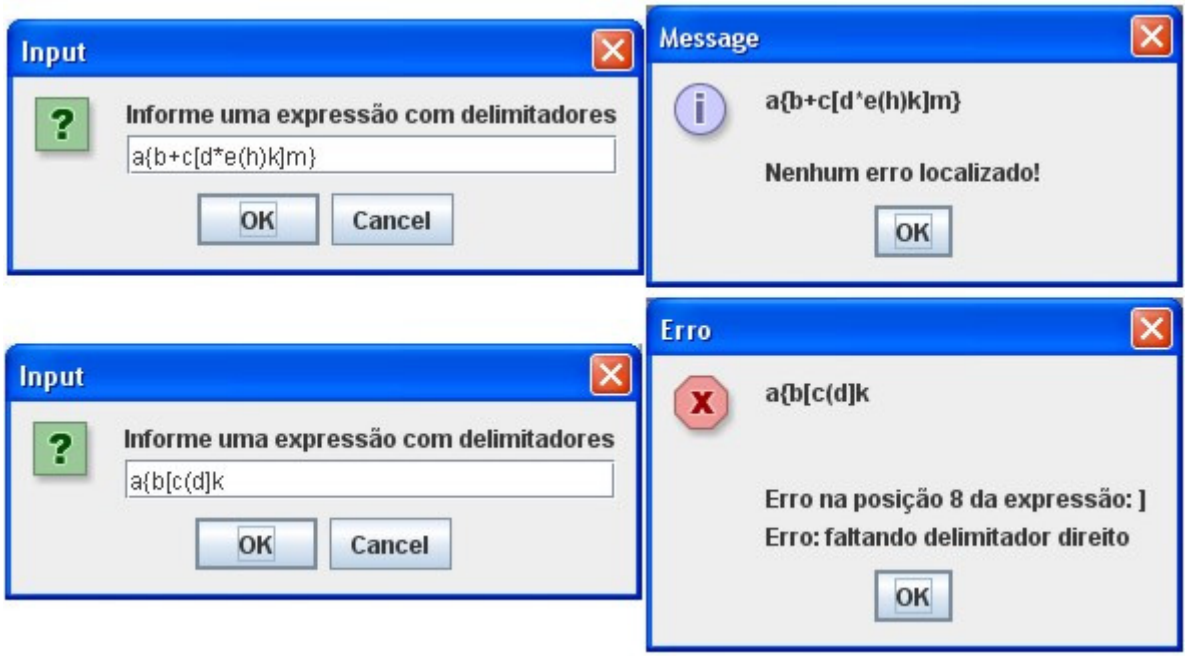


Exercícios

- **Os exercícios propostos a seguir têm o intuito de levá-lo a exercitar seu domínio sobre as estruturas de dados apresentadas neste capítulo, principalmente as listas duplamente encadeadas, as pilhas dinâmicas e as filas dinâmicas. O primeiro exercício pode ser resolvido com uma pilha dinâmica, o segundo exercício pode ser resolvido com uma fila dinâmica e o terceiro exercício pode ser resolvido com uma lista duplamente encadeada.**

Exercício 1

- **Crie um novo aplicativo chamado ValidadorDelimitadores que seja capaz de analisar qualquer expressão e que possa indicar se os delimitadores foram utilizados corretamente.**
  - **A figura abaixo ilustra os diálogos que deverão ser produzidos por este aplicativo.**



Exercício 1

- O primeiro diálogo é aquele que capta a expressão a ser analisada. Como resultado final, este aplicativo pode exibir uma mensagem que indica que nenhum erro foi localizado ou pode exibir uma mensagem de erro que indique quais foram os problemas detectados.
- Este aplicativo deve avaliar apenas três delimitadores: as chaves, os colchetes e os parênteses.
  - Todos os demais caracteres que fizerem parte da expressão devem ser ignorados.
  - A expressão deve ser considerada válida somente se houver um delimitador de fechamento para cada delimitador de abertura e se os fechamentos ocorrerem na ordem correta.
  - O último delimitador de abertura inserido na expressão deve ser o primeiro a ser fechado.

Exercício 1

- A mensagem de erro deve indicar toda ocorrência onde for encontrado um delimitador de fechamento que não coincide com o último delimitador de abertura.
  - Nesse caso, a mensagem deve indicar a posição em que o delimitador de fechamento se encontra.
  - Se um delimitador de fechamento for encontrado quando não houver mais nenhum delimitador de abertura para ser fechado, a mensagem de erro também deve incluir esta ocorrência e indicar a posição em que ele se encontra.
  - Esta mensagem ainda deve indicar quando estiver faltando um delimitador direito, ou seja, quando a expressão terminar e ainda houver algum delimitador de abertura que não foi fechado.

Exercício 1

- Para resolver este problema, você pode criar uma pilha de caracteres.
  - Sugere-se que utilize a classe PilhaDinamica para instanciar esta pilha.
  - O procedimento básico consiste em inserir todo delimitador de abertura na pilha e removê-lo quando o delimitador de fechamento correspondente for encontrado.
  - Se o delimitador de fechamento encontrado não coincide com o último delimitador de abertura da pilha, basta registrar o erro e prosseguir.
  - O mesmo deve ser feito se for encontrado um delimitador de fechamento quando a pilha estiver vazia ou se o final da expressão for atingido e a pilha ainda tiver algum caractere.

Exercício 2

- **Crie um novo aplicativo que permita o registro do atendimento de pacientes em um posto de saúde.**
  - **A figura abaixo apresenta o primeiro diálogo que deve ser exibido e as opções que ele deve oferecer.**

Exercício 2

- O primeiro atendimento ao paciente deve ser prestado por um funcionário.
  - É este funcionário que deve registrar a chegada do paciente utilizando a primeira opção do aplicativo.
  - Para fazer este registro, o funcionário deverá informar três dados: o nome completo do paciente, a sua idade e o seu sexo.
  - Depois disso, o funcionário deverá solicitar que o paciente aguarde em uma sala de espera.
- A segunda opção do aplicativo deve ser acionada pelo funcionário para chamar o próximo paciente que será atendido.
  - Neste momento, o aplicativo deve produzir uma mensagem contendo todos os dados do paciente que se encontra na frente da fila e, em seguida, deve remover este registro.
- A terceira opção do aplicativo poderá ser acionada a qualquer momento pelo funcionário para consultar qual é o próximo atendimento.
  - Neste caso, o aplicativo também deve produzir uma mensagem com os dados do paciente que está na frente da fila, mas não deve remover este registro.

Exercício 3

- **Crie um aplicativo, chamado CadastroCliente, que permita a realização de cadastros de clientes e utilize uma lista duplamente encadeada como mecanismo de armazenamento para os dados.**
  - **A figura abaixo apresenta o primeiro diálogo que deve ser exibido e as opções que ele deve oferecer.**

Exercício 3

- Este aplicativo deve permitir a realização das seguintes operações:
  - Gravar novos cadastros na lista.
  - Excluir cadastros da lista.
  - Alterar os dados de um cadastro existente.
  - Recuperar e exibir os dados de um cadastro existente.
  - Exibir um relatório com os dados de todos os cadastros.
- Os dados que devem ser gravados em cada cadastro de cliente são os seguintes: o seu código, o seu nome completo, a sua data de nascimento e o seu telefone.
  - Se um dado inválido for informado, o aplicativo deve exibir uma mensagem de erro e solicitá-lo novamente.
  - As regras para validação de cada dado ficam a seu critério.

Exercício 3

- A exclusão de um cadastro de cliente deve ser realizada com base em seu código.
  - O usuário deve informar o código do cliente cujo cadastro deve ser excluído e o aplicativo deve localizá-lo e removê-lo da lista.
  - Ao final desta operação o aplicativo deve exibir uma mensagem que confirme a exclusão do cadastro e exiba os dados do mesmo.
  - Se não houver nenhum cliente cadastrado com o código informado, o aplicativo deve exibir uma mensagem de erro.
- A alteração de um cadastro de cliente deve iniciar com a solicitação do seu código.
  - Se não houver nenhum cadastro com o código informado, o aplicativo deve exibir uma mensagem de erro e abortar a operação.
  - Caso o cadastro seja localizado, o aplicativo deve solicitar a atualização de cada um dos demais dados do cliente: seu nome, sua data de nascimento e seu telefone.
  - Ao final, uma mensagem deve confirmar a realização da alteração.

Exercício 3

- A localização de um cadastro deve ser realizada com base no código do cliente.
  - O usuário deve informar este dado e o aplicativo deve tentar localizá-lo.
  - Se o cadastro for localizado, uma mensagem com todos os dados deve ser exibida.
  - Caso contrário, uma mensagem de erro deve ser produzida.
- O relatório deve ser apresentado através de uma mensagem gráfica.
  - Ele deve apresentar os dados cadastrais de cada cliente em uma linha distinta.
  - Além disso, estes dados devem ser separados com algum tipo de sinal e devem estar formatados adequadamente.

Contato

Com o autor:

Rui Rossi dos Santos
E-mail: livros@ruirossi.pro.br
Web Site: http://www.ruirossi.pro.br

Com a editora:

Editora NovaTerra
Telefone: (21) 2218-5314
Web Site: http://www.editoranovaterra.com.br