Lista de Exercícios 2 - Estruturas de Dados Lineares

Oficina de AEDs2 - Prof. Matheus Pereira

1. Exercícios em Java

1. Pilha - Histórico de Navegação Web

Contexto: Um navegador web precisa armazenar o histórico de páginas visitadas para permitir que o usuário volte às páginas anteriores. Implemente uma pilha para simular esse comportamento.

Enunciado: Implemente uma classe `WebHistory` que utiliza um array de tamanho fixo para armazenar URLs (strings). A classe deve incluir os seguintes métodos:

- `WebHistory(int maxSize)`: Construtor que define o tamanho máximo do histórico.
- `void push(String url)`: Adiciona uma nova página ao histórico.
- `String pop()`: Remove e retorna a última página visitada.
- `boolean isEmpty()` e `boolean isFull()`: Verificam o estado da pilha.
- `String peek()`: Retorna a última página sem removê-la.

Utilize um array de `String` e uma variável para controlar o topo da pilha.

2. Fila - Sistema de Impressão

Contexto: Um sistema de impressão gerencia documentos enviados por múltiplos usuários em ordem de chegada. Implemente uma fila para processar esses documentos.

Enunciado: Crie uma classe `PrintQueue` usando um array estático para armazenar nomes de arquivos. Implemente:

- `PrintQueue(int capacity)`: Construtor que define a capacidade da fila.
- `void enqueue(String file)`: Adiciona um documento à fila.
- `String dequeue()`: Remove e retorna o próximo documento.
- `boolean isEmpty()` e `boolean isFull()`: Verificam o estado da fila.
- `String front()`: Retorna o próximo documento sem removê-lo.

Use variáveis para controlar os índices de início e fim da fila.

3. Fila Circular - Buffer de Dados

Contexto: Um sistema embarcado precisa processar dados de sensores em tempo real, onde novos dados sobrescrevem os mais antigos se o buffer estiver cheio.

Enunciado: Implemente uma classe `SensorBuffer` com array para armazenar leituras de sensores (inteiros). A classe deve incluir:

- `SensorBuffer(int size)`: Construtor que define o tamanho do buffer.
- `void enqueue(int data)`: Adiciona uma nova leitura (sobrescreve o início se necessário).
- `int dequeue()`: Remove e retorna o dado mais antigo.
- `boolean isEmpty()` e `boolean isFull()`: Verificam o estado do buffer.
- `int getCount()`: Retorna o número de elementos atualmente no buffer.

Implemente controle de índices circulares usando operador módulo (%).

4. Lista Ordenada - Catálogo de Produtos

Contexto: Um e-commerce precisa manter um catálogo de produtos ordenados por preço para buscas rápidas.

Enunciado: Crie uma classe `ProductCatalog` e uma classe `Product` (com campos `String name` e `double price`). Implemente:

- `ProductCatalog(int maxProducts)`: Construtor que define a capacidade máxima.
- `void insertSorted(Product product)`: Insere um produto mantendo a ordenação por preço.
- `boolean removeElement(String name)`: Remove um produto pelo nome (retorna `true` se encontrado e removido).
- `Product[] searchByPriceRange(double min, double max)`: Retorna array com produtos na faixa de preço.
- `int size()`: Retorna o número atual de produtos.

Mantenha o array sempre ordenado por preço crescente.

5. Lista Redimensionável - Agenda de Contatos

Contexto: Um aplicativo de agenda precisa expandir automaticamente sua capacidade quando o número de contatos excede o limite inicial.

Enunciado: Crie uma classe `ContactList` e a classe `Contact` (campos `String name` e `String phone`). Implemente:

- `ContactList(int initialCapacity)`: Construtor com capacidade inicial.
- `void insert(Contact contact)`: Adiciona contato (redimensiona automaticamente se necessário).
- `boolean remove(String name)`: Remove contato pelo nome.
- `Contact getContact(int index)`: Retorna contato pela posição.
- `private void resize()`: Método interno que dobra o tamanho do array quando necessário.
- `int size()`: Retorna número atual de contatos.

Implemente manualmente o método para redimensionar o array.

6. Fila de Prioridade - Pronto-Socorro

Contexto: Um pronto-socorro precisa atender pacientes com base na gravidade de seus casos (prioridade alta ou baixa).

Enunciado: Implemente uma classe `PriorityQueue` e a classe `Patient` (campos `String name` e `int priority`). Implemente:

- `PriorityQueue(int capacity)`: Construtor que define a capacidade.
- `void enqueue(Patient patient)`: Adiciona paciente na posição correta por prioridade (1 = mais urgente, 5 = menos urgente).
- `Patient dequeue()`: Remove e retorna paciente com maior prioridade (menor valor numérico).
- `Patient peek()`: Retorna paciente com maior prioridade sem remover.
- `boolean isEmpty()` e `boolean isFull()`: Verificam estado da fila.
- `int size()`: Retorna número atual de pacientes.

Mantenha o array ordenado por prioridade para eficiência no desenfileiramento.