Filas Dinâmicas

Prof. Leandro Colevati

Introdução

- Definição
  - É uma estrutura de dados de tamanho variável, sendo que elementos são incluídos (enfileirados) pelo fim da fila e removidos (desenfileirados) pelo início da fila.

	15	12	19	51	7	213	54	99	88	

INÍCIO FIM

Introdução

- Note que uma fila é uma estrutura de dados do tipo FIFO (First In First Out).

- Isto porquê o primeiro elemento enfileirado é sempre o primeiro a ser desenfileirado.

Introdução

Simular operações

Insert(1)
Insert(2)
Insert(10)
Remove()
List()
Insert(5)
Insert(8)
Remove()
Remove()
List()
Size()

Introdução

- Operações Básicas:
  - Teste de fila vazia;
  - Criação da fila;
  - Enfileiramento;
  - Desenfileiramento;
  - Acesso aos elementos da fila.
    - Lista os elementos da fila
    - Tamanho

Introdução

- Do ponto de vista da alocação de memória para esse tipo de estrutura de dados, podem ser implementadas usando:
  - Alocação Estática: Em geral através de arranjo ou vetor;
  - Alocação Dinâmica: Utilizando ponteiro (Implícito ou Explícito)

Alocação Estática

- Características:
  - Teste de Fila Vazia: Índice fim menor que inicio;
  - Cria-se um vetor de tamanho suficientemente grande para a finalidade de uso;
  - Quando um elemento é enfileirado o valor do fim é igual ao tamanho do vetor;
  - Quando um elemento é desenfileirado (se possível) o valor do inicio é acrescido em 1.

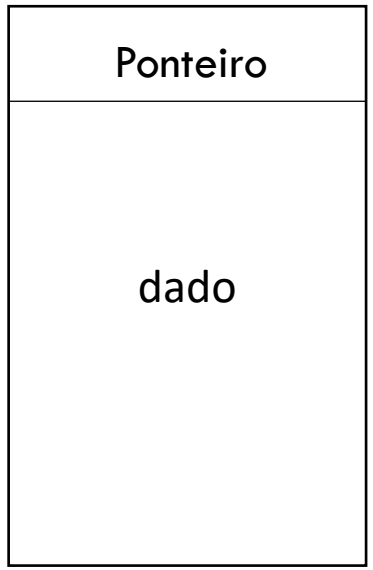
Alocação Estática

- Características:
  - Note que, desta forma, posições vão se perdendo na fila estática.
  - Um solução, é deslocar o conteúdo do vetor em uma posição no sentido do índice inicial do vetor.
  - Assim, como o início será sempre um mesmo valor, há apenas a necessidade da variável fim.

Alocação Dinâmica

- Considere a definição do tipo Fila abaixo:

```
class No {
   tipo    dado;
   No      próximo; //Ponteiro
}
```



Alocação Dinâmica

- Considere a definição do tipo Fila abaixo:
  - Ponteiro Inicial → NULL
  - Ponteiro Final → NULL

Alocação Dinâmica

- Teste de fila vazia:

```
No inicio;
No fim;
booleano filaVazia() {
    se (inicio == nulo && fim == nulo) {
        retorne verdadeiro;
    } senão {
        retorne falso;
    }
}
```

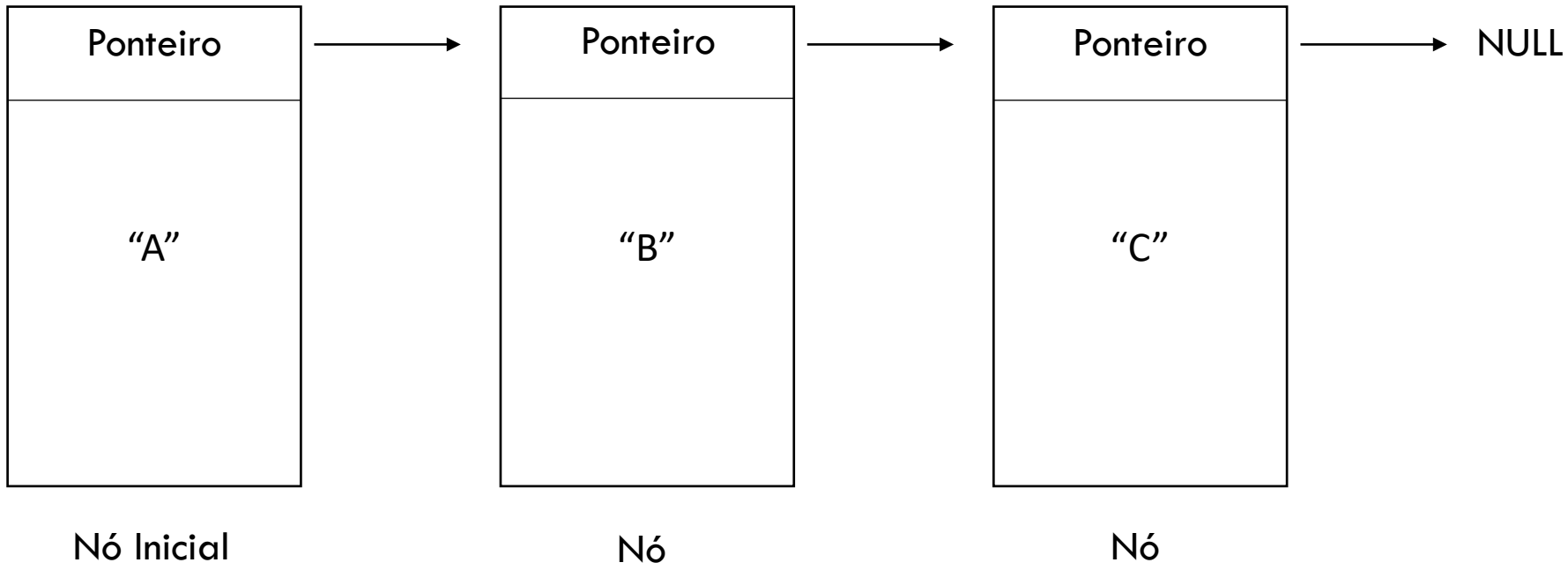
Alocação Dinâmica

- Enfileirando um elemento (Inserindo):

```
No inicio;
No fim;
void insert(tipo valor) {
    No elemento = new No();
    elemento.dado = valor;
    se (inicio == nulo) {//Verificar se é o primeiro dado
            inicio = elemento;
            fim = elemento;
            elemento.proximo = nulo;
    } senao {
            se (inicio == fim && inicio != nulo) {//Verificar se é o 2o dado
                    fim = elemento;
                    inicio.proximo = fim;
                    fim.proximo = nulo;
            } senao {
                    fim.proximo = elemento;
                    elemento.proximo = nulo;
                    fim = elemento;
            }
   }
}
```

Alocação Dinâmica

- Enfileirando um elemento (Inserindo):



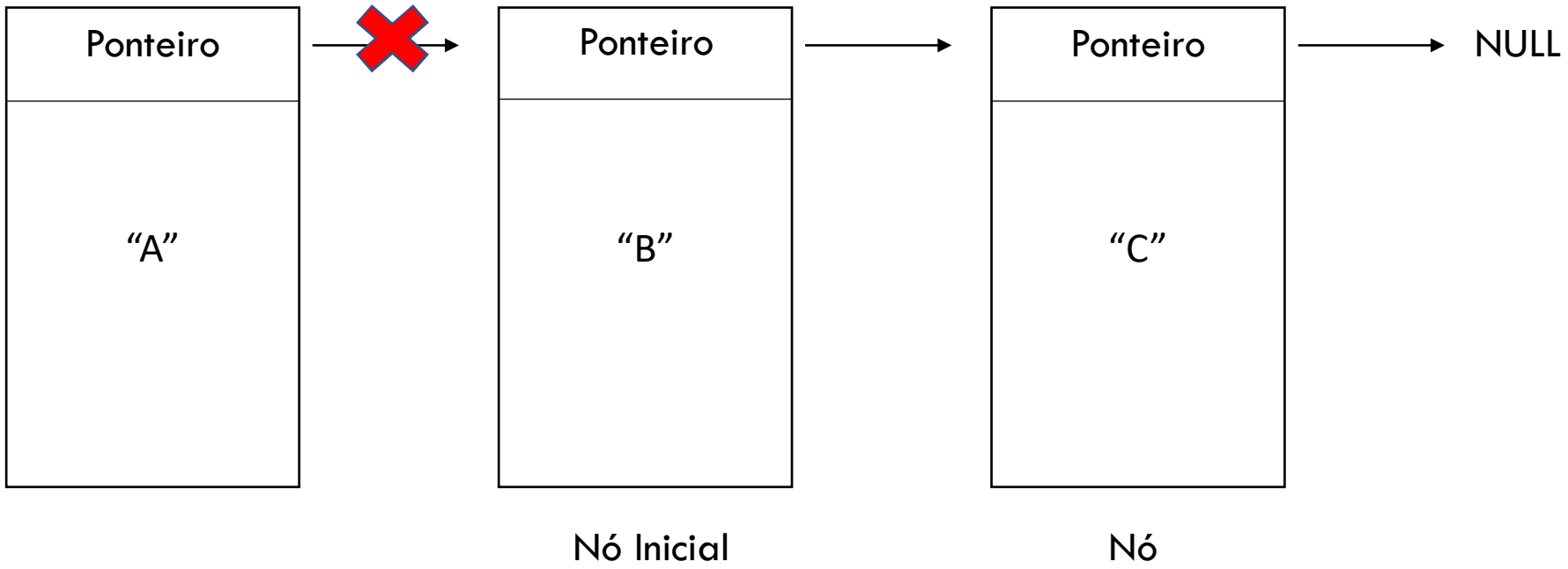
Alocação Dinâmica

- Desenfileirando um elemento (Removendo):

```
No inicio;
No fim;
tipo remove() {
    se (filaVazia() == verdadeiro) {
       exceção("Não há elementos na fila");
    }
    No auxiliar = inicio;
    se (inicio == fim && inicio != nulo) {//Fila tem só 1 elemento
       inicio = nulo;
       fim = nulo;
    } senão {
       inicio = inicio.próximo;
    }
    retorne auxiliar.dado
}
```

Alocação Dinâmica

- Desenfileirando um elemento (Removendo):



Alocação Dinâmica

- Acessando elementos da fila
  - Como estamos usando uma lista simplesmente encadeada podemos acessar todos os elementos da fila, a partir do inicio, fazendo pequenos ajustes na função de acesso aos elementos de uma lista simplesmente encadeada para usar com o tipo fila.

Alocação Dinâmica

- Listando os elementos da fila:

```
No inicio;
No fim;
void list() {
        se (filaVazia() == verdadeiro) {
                exceção("Não há elementos na fila");
        }
        No auxiliar = inicio;
        enquanto (auxiliar != nulo) {
                escreva(auxiliar.dado);
                auxiliar = auxiliar.próximo;
        }
}
```

Alocação Dinâmica

- Tamanho da fila:

```
No inicio;
No fim;
int size() {
        int cont = 0;
        se (filaVazia() == falso) {
            No auxiliar = inicio;
            enquanto (auxiliar != nulo) {
                cont = cont + 1;
                auxiliar = auxiliar.próximo;
                 }
        }
        retorne cont;
}
```

Alocação Dinâmica

- Exemplo(Fila de inteiros):

```
class exemplo {
        void main(String[] args) {
            Fila f = new Fila();
            booleano vazia = f.filaVazia();
            escreva(vazia);
            int tamanho = f.size();
            escreva("Tamanho:"+tamanho);
            int dado = f.remove();
            f.insert(5);
            f.insert(4);
            f.insert(3);
            f.insert(2);
            f.insert(1);
            tamanho = f.size();
            escreva("Tamanho:"+tamanho);
            f.list();
        ...
```

...continuação

```
class exemplo {
        void main(String[] args) {
            ...
            int removido = f.remove();
            escreva(removido);
            tamanho = f.size();
            escreva(tamanho);
            f.list();
            vazia = f.filaVazia();
            escreva(vazia);
        }
}
```

Teste de Mesa

Considere o vetor:

36	28	146	14	-65	117	-40	24	138	116

Faça o teste de mesa conforme o algoritmo:

```
Fila f = new Fila();
Para (int valor : vetor) {
        Se (filaVazia()) {
            f.insert(valor * 10);
        } Senao Se (valor >= 0) {
            f.insert(valor + 10);
        } Senao Se (valor < 0) {
            int auxiliar = f.remove();
            escreva(auxiliar);
           f.insert(valor);
        }
}
escreva(f.list());
escreva(f.size());
```

Exercício 1

Simular o comportamento de pilhas dinâmicas para os algoritmos abaixo (A simulação deve deixar evidente a pilha que sobrou na memória):

a)

```
Para (i = 0 ; i < 10 ; i++) {
        Se (i % 2 == 0) {
                Insert(i * i);
        } Senão {
                Se (i <= 5) {
                   Insert(i);
                } Senão {
                   Escreva(Remove());
                }
        }
        List();
}
Escreva(Size());
```

b)

```
Para (i = 100 ; i < 115 ; i++) {
        Se (isEmpty()) {
                Insert(i + 100);
        } Senão {
                Se (Size() <= 4) {
                     Insert(i + 50);
                } Senão {
                     Escreva(Remove());
                }
        }
        List();
}
Escreva(Size());
```

Exercício 2

a) Adaptar o modelo de Fila Dinâmica desenvolvido em aula, com os métodos esperados, para uma Fila de Strings.
Transformar o projeto em uma biblioteca, gerando o JAR FilaStrings.

b) Adaptar o modelo de Fila Dinâmica desenvolvido em aula, com os métodos esperados, para uma Fila de int.
Transformar o projeto em uma biblioteca, gerando o JAR FilaInt.

c) Adaptar o modelo de Fila Dinâmica desenvolvido em aula, com os métodos esperados, para uma Fila de objetos.
Transformar o projeto em uma biblioteca, gerando o JAR FilaObject.

Exercício 3

Criar um projeto Java que implementa com a biblioteca FilaStrings para simular um identificador de chamadas telefônicas enquanto o aparelho está fora da rede ou desligado. A aplicação deve ter uma classe no package controller, TelefoneController que tem os seguintes métodos:

- insereLigacao(Fila f, String numeroTelefone): void, que insere números de telefone em uma fila iniciada
- consultaLigacoes(Fila f):void, que desenfilera cada ligação realizada e exibe no console. Exibir uma exceção caso não haja ligação

Deve ter também uma classe Principal no package view com operações usando JOptionPane que permita ao usuário inserir números e consultar as chamadas perdidas. A aplicação deve rodar até uma opção de saída ser selecionada pelo usuário

Exercício 4

Criar um projeto Java que implementa com a biblioteca FilaStrings para simular um ambiente corporativo, comum, como agências bancárias que tem diversos computadores e 1 impressora central. Nesses casos, a impressora tem uma fila de impressões para que cada documento enviado comece e termine a impressão sem que documentos se misturem.

A aplicação deve ter uma classe no package controller, ImpressoraController que tem os seguintes métodos:

- insereDocumento(Fila f, String documento): void, o documento no formato ID_PC;Nome_Arquivo (Já validado antes do envio) deverá enfileirar os documentos enviados
- imprime(Fila f):void, que desenfilera um documento da fila, por vez, exibe no console [#PC: ID_PC – Arquivo: Nome_Arquivo]. Cada impressão dura de 1 a 2 segundos usar Math.random() ou a classe Random e um Thread.sleep*(tempo) para simular o tempo de impressão. Exibir uma exceção caso não haja documento na fila de impressão.

Deve ter também uma classe Principal no package view com operações usando JOptionPane que permita ao usuário inserir e validar os documentos de entrada e iniciar um procedimento de impressão. A aplicação deve rodar até uma opção de saída ser selecionada pelo usuário

*** Como a classe de ImpressoraController não é uma Thread, para forçar um sleep, deve-se usar uma chamada estática da classe Thread (Thread.sleep(tempo))**