Universidade Cruzeiro do Sul

Estrutura de dados

Conteúdo

- Estrutura de dados Fila.
- Fila estática sequencial.

Professores

Prof. Manuel F. Paradela **Ledón**
Profª Cristiane P. Camilo Hernandez
Prof. Amilton Souza Martha
Prof. Daniel Calife

Fila (Queue)

Universidade Cruzeiro do Sul

- Uma **fila** (**queue** em inglês) é uma estrutura de dados em que as inserções são realizadas em um extremo, enquanto as remoções são feitas no outro.

- O extremo onde os elementos são inseridos é denominado final (**tail**, cauda, último) e aquele de onde são removidos é denominado começo (**head**, cabeça, início, primeiro) da fila.

final
tail

início
head

Figura em: http://www.meusmiolos.com.br/wp-content/uploads/2015/07/fila4.jpg

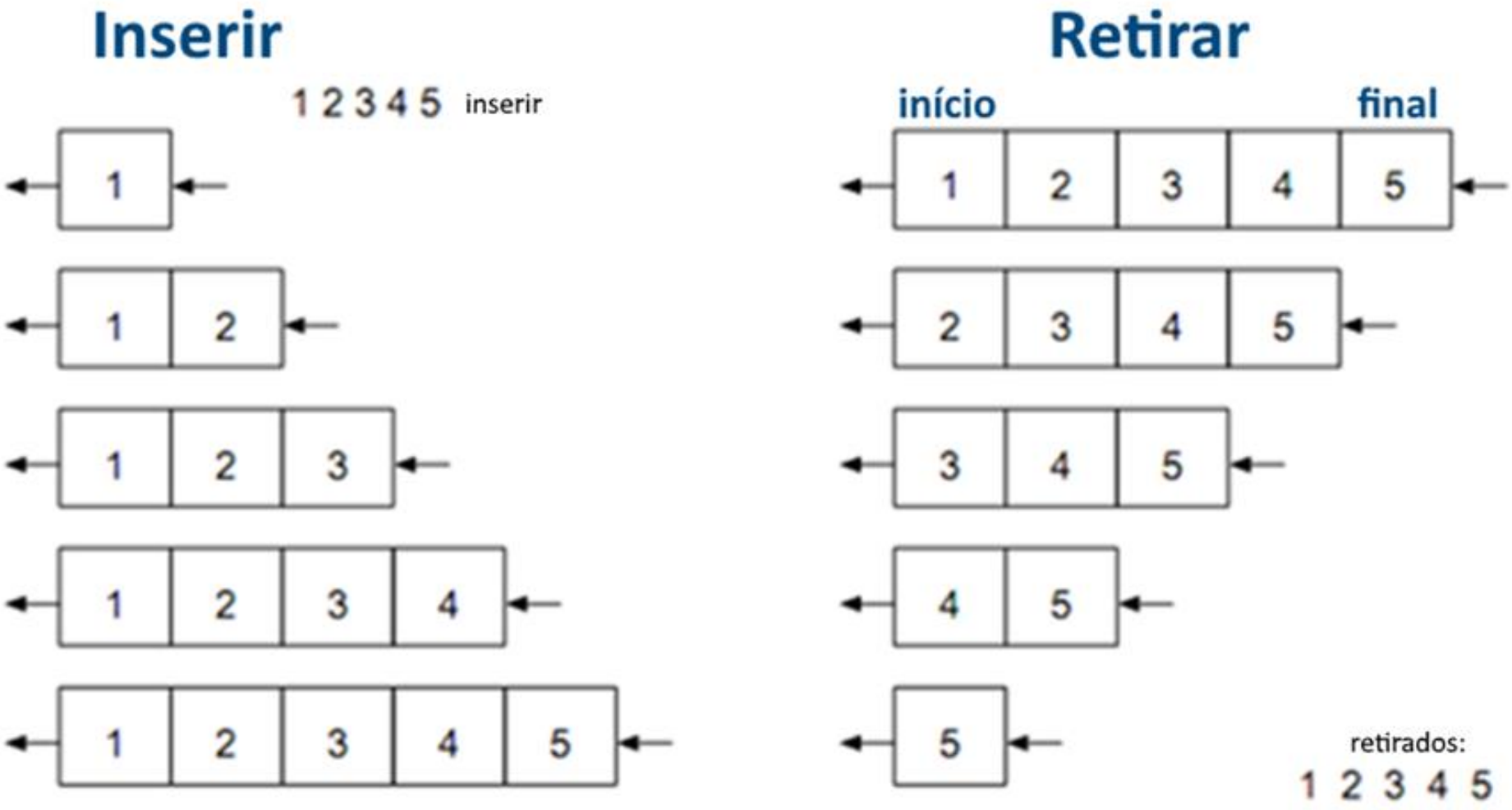
Operações básicas de uma fila

Universidade Cruzeiro do Sul

- A ordem de saída dos elementos de uma fila corresponde diretamente à ordem de entrada dos mesmos na fila, de modo que o primeiro elemento que entrou será o primeiro a sair (primeiro a ser atendido), caracterizando a estrutura como **FIFO** ➔ **F**irst-**I**n / **F**irst-**O**ut.

- As **operações básicas** que uma **fila** (queue) suporta são:
  - **enqueue**: insere um elemento/objeto no final da fila - O(1)
  - **dequeue**: remove um elemento/objeto do começo da fila - O(1)
  - **isEmpty**: verificar se a fila está vazia - O(1)
  - **isFull**: verificar se a fila está cheia (se for alocação estática) - O(1)
  - **peek**: (espiar, observar) retornar, sem eliminar, o objeto que se encontra no início da fila - O(1)
  - **toString**: retorna todos os itens/objetos da fila - O(n)

Exemplo: funcionamento de filas



Baseado na figura que estava em: https://lgsreal.wordpress.com/2011/01/24/ POSCOMP, 2010

Exemplo de operações com uma fila F

Operação	Estado da fila
----------	F:[ ]
F.enqueue(a)	F:[a]
F.enqueue(b)	F:[a, b]
F.enqueue(c)	F:[a, b, c]
F.enqueue(d)	F:[a, b, c, d]
F.dequeue() elimina e retorna a	F:[b, c, d]
F.enqueue(F.dequeue())	F:[c, d, b]
F.enqueue(e)	F:[c, d, b, e]
F.enqueue(F.dequeue())	F:[d, b, e, c]

Convenção gráfica utilizada:
F: [**início**, a, b, c, d, **fim**] ou F: [**head**, a, b, c, d, **tail**]

Outro exemplo de operações com uma fila

Seja a fila f = ["Primeiro", "Segundo"], inicialmente com esses valores anteriores, sendo "Primeiro" o primeiro item na fila e "Segundo" o último. Como ficará finalmente esta fila f, depois de executar a sequência de comandos a seguir?

```
f.enqueue("Lea")
f.enqueue("Betty")
var a = f.dequeue()
f.enqueue("Tony")
f.enqueue("José")
f.enqueue(f.dequeue())
f.enqueue("Luiz")
```

f = ["Primeiro","Segundo","Lea","Betty","Lea","Tony","José"]

f = ["Lea", "Betty", "Tony", "José", "Segundo", "Luiz"] resposta correta!

f = ["Lea","Betty","Lea","Tony","Luiz"]

f = ["Tony","José","Tony","Luiz"]

f = ["Tony","José","Betty","Luiz","Primeiro","Segundo"]

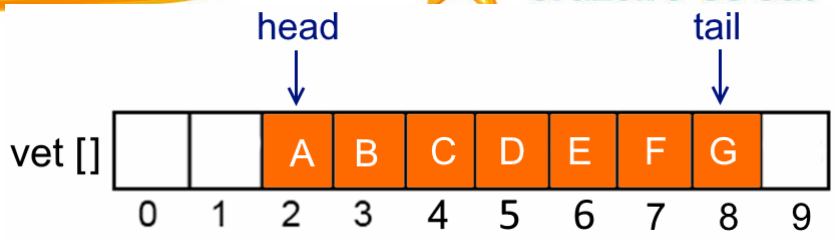
Mais um exemplo de operações com uma fila

Seja a fila f : ["Pedro", "Ana"], inicialmente com esses valores anteriores, sendo "Pedro" o primeiro item na fila e "Ana" o último. Como ficará finalmente esta fila f, depois de executar a sequência de comandos a seguir?

```
f.enqueue("Lea")
f.enqueue("Betty")
var a = f.dequeue()
f.enqueue("Tony")
f.enqueue("José")
f.enqueue(f.peek())
f.enqueue(a)
```

f : ["Ana", "Lea", "Betty", "Tony", "José", "Ana", "Pedro"] resposta correta

Fila estática sequencial

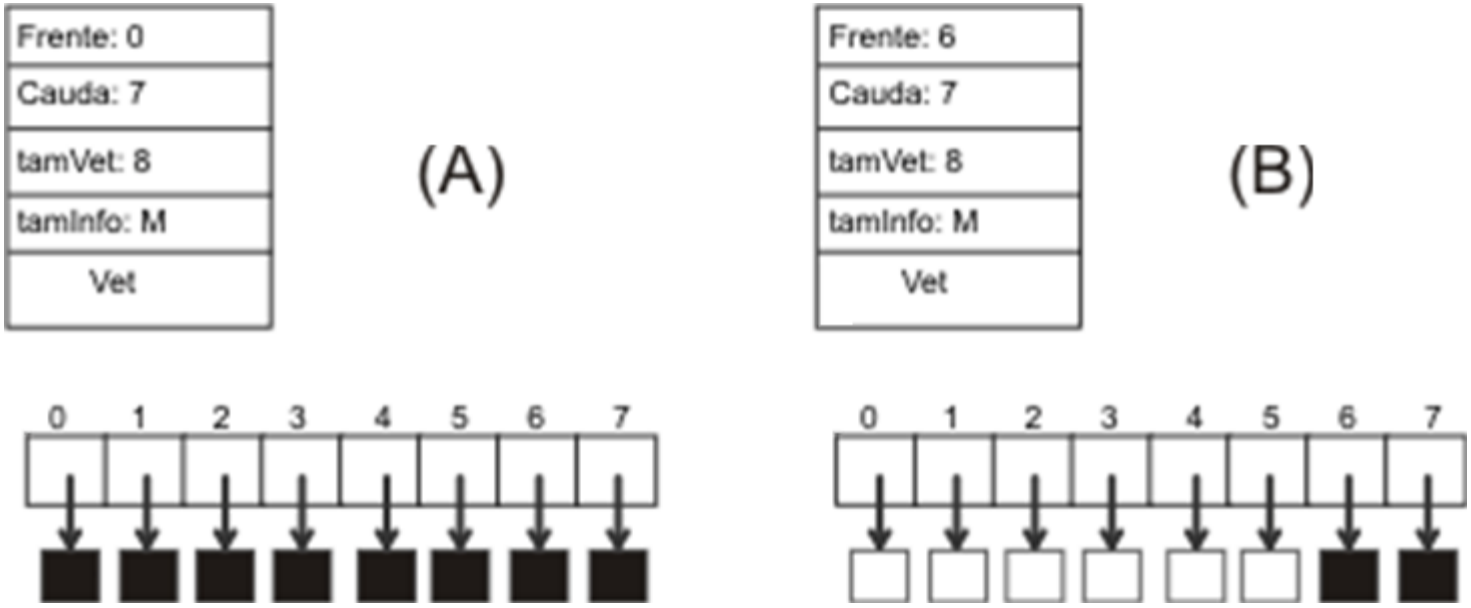


Ao tentarmos implementar uma Fila Estática Sequencial (guardada em um vetor), nos deparamos com **dois problemas**:

1. Sabemos que as inserções são feitas sempre no fim da fila e as remoções no começo. É inviável, quanto a performance, efetuar um deslocamento de todos os elementos para a esquerda em cada remoção, então existirão espaços em branco no início da fila (antes do head) que não poderiam ser reaproveitados, pois a inserção é sempre feita à direita;

2. Quando inserimos elementos e a fila ficar cheia (tail == MAX-1), não haveria mais possibilidades de inserção. Poderíamos achar que a fila está cheia, mas na verdade pode existir espaço vazio no lado esquerdo!

Exemplo

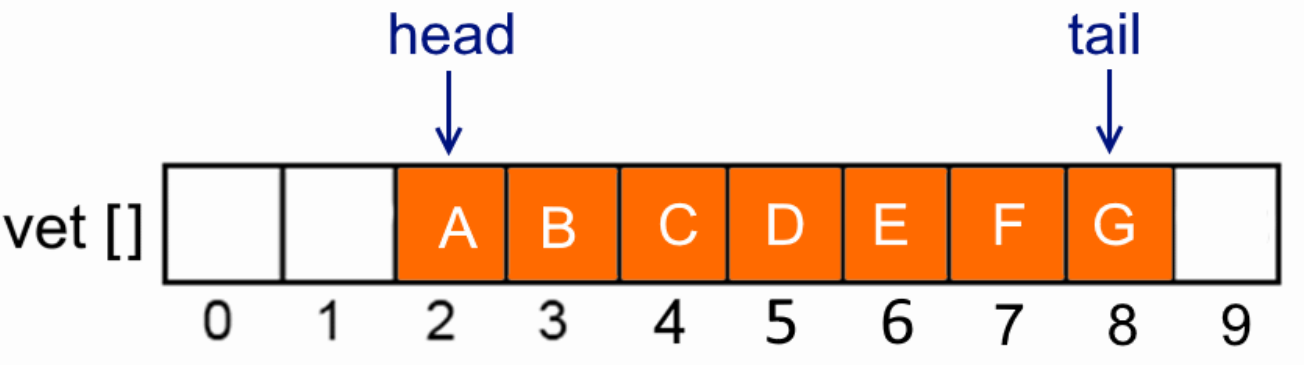


cheio **falso cheio**

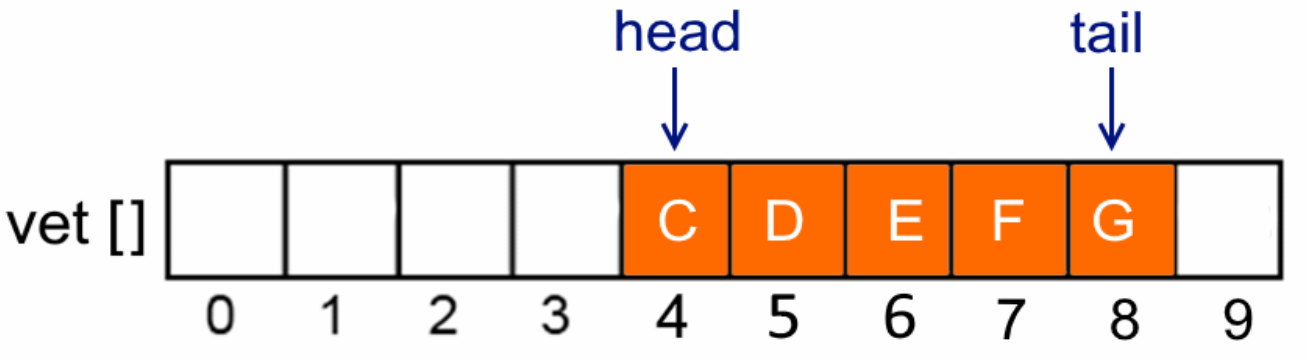
Fila estática sequencial - convenções úteis

- **Para resolver os problemas de aproveitamento da memória do vetor** que guarda a fila, podemos criar uma **fila circular** (na verdade: uma **fila com comportamento circular**) onde, após a última posição do vetor, voltaremos para a primeira posição (índice 0), seja em inserções ou em retiradas de elementos.

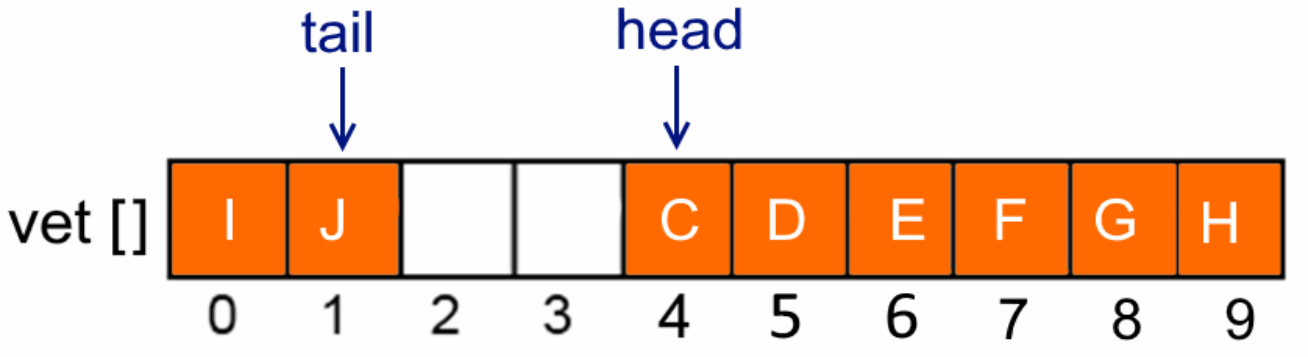
- Para resolver o problema de descobrir se a Fila está cheia ou vazia utilizaremos um **contador de elementos**, para contar os itens guardados na fila. Se este contador for zero significa que a fila está vazia e se tiver valor MAX (que é o tamanho do vetor) significa que a fila está cheia (ou seja, que o vetor está cheio).



Suponhamos uma fila guardada no vetor vet[] com este estado inicial.



Depois de **retirar** dois elementos, A e B, a fila ficaria assim (o início head avançou duas posições).



Depois de **inserir** três elementos H, I e J, a fila ficaria assim: observe que tail começou em zero para inserir I.

Implementar uma fila guardada em um vetor, com **funcionamento "circular"**, permite aproveitar toda a capacidade do mesmo!

TAD_Queue (tipo abstrato de dado)

```
public interface TAD_Queue {  // tipo abstrato de dado que descreve a Fila

    public boolean isEmpty();  //Verifica se a fila está vazia
    public boolean isFull();     //Está cheia? Só em alocação estática de memória.

    //Insere um objeto x no final da fila:
    public Object enqueue(Object x);

    //Remove e retorna um elemento/objeto do início da fila:
    public Object dequeue();

    //Retorna o objeto no início da fila (o primeiro da fila), sem eliminar:
    public Object peek();  //digamos que é equivalente ao top() da pilha

    //Retorna o conteúdo da fila (todos os elementos/objetos):
    public String toString();
}
```

Implementação da ED fila estática sequencial (1)

- Esta implementação se encontra dentro do projeto NetBeans chamado FilaEstaticaSequencial.
- Atributos da classe para a ED Queue:

```
//Implementação de uma Fila Estática Sequencial com funcionamento circular.

public class Queue implements TAD_Queue {
    private int total = 0;          //Contador de elementos (0 se a fila estiver vazia)
    private int head = -1;          //Início da queue  (head = -1 se a fila estiver vazia)
    private int tail = -1;          //Final da queue (convenção:  tail = -1 se fila vazia)
    private Object memo[];          //Vetor para armazenar os objetos da fila
    private int MAX;                //Capacidade do vetor, diferente do atributo 'total'
```

Implementação da ED fila estática sequencial (2)

```
//métodos construtores que inicializam a Queue em estado 'vazia'
public Queue() {
   MAX = 1000;  //capacidade para 1000 objetos
   memo = new Object[MAX];
   total = 0;
   head = -1;
   tail = -1;
}
```

métodos construtores

```
//inicializa a Queue em estado vazia,
//com um tamanho inicial para o vetor
public Queue(int qtde) {
   MAX = qtde;
   memo = new Object[MAX];
   total = 0;
   head = -1;
   tail = -1;
}
```

Implementação de uma fila estática sequencial (3)

```
// Retorna o objeto que está no início da fila (o primeiro da
// fila), mas sem eliminá-lo.

public Object peek () {
   if( !isEmpty() ) return ( memo[head] );  //retorna o item na cabeça
   else return null; //impossível retornar o objeto no início se a fila está vazia
}
```

```
// Outra estratégia seria lançar uma exception nos métodos peek,
// enqueue e dequeue, em lugar de retornar null:

public Object peek () throws  Exception  {
            if( !isEmpty() ) return ( memo[head] );
            else throw new Exception("a fila está vazia");
}
```

Implementação de uma fila estática sequencial (4)

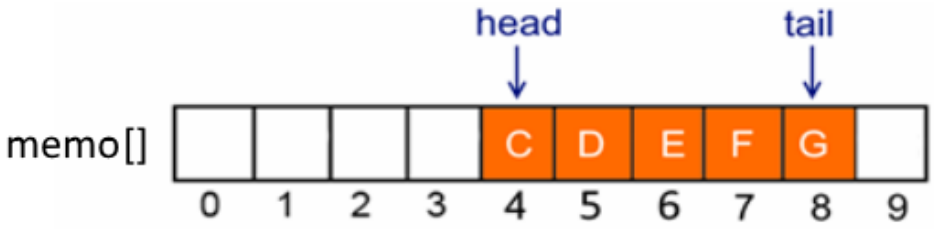
```
//Verifica se a fila está vazia
public boolean isEmpty () {
    return(total==0);
}

//Verifica se a fila está cheia
public boolean isFull () {
    return(total==MAX);
}
```

Os métodos enqueue e dequeue deverão atualizar a contagem na variável **total**

```
//outra forma de implementação:
public boolean isEmpty () {
    if (total==0) return true;
                else return false;
}
```

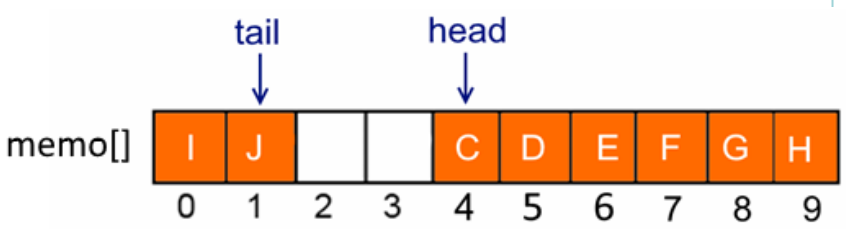
Implementação de uma fila estática sequencial (5)

```
//Insere um objeto x no final (tail) da fila
public Object enqueue (Object x) {
        if( !isFull()  &&  x!=null ) {   // pré-condições
                if( ++tail >= MAX ) tail = 0;  // avançamos a cauda (circular)
                if ( head == -1 ) head = tail;  // caso: a fila estava vazia
                memo[tail] = x;
                total++;
                return x;
        }
        else return null;  //não podemos inserir se a fila está cheia ou x==null
}
```



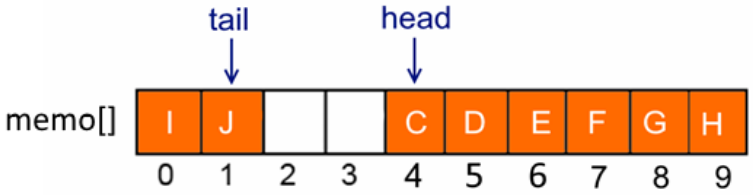
Implementação de uma fila estática sequencial (6)

```
//Retorna e remove o elemento que se encontra no início (head) da fila
public Object  dequeue () {
        if( !isEmpty() ) {  //verificamos a pré-condição
                Object objeto = memo[head];  //pegamos antes de avançar head
                if( ++head >= MAX) head = 0;  //avançamos a cabeça (circular)
                total--;
                if( total == 0 ) {  //ficou vazia
                        head = -1;
                        tail = -1;
                }
                return objeto;  //retornamos o objeto que estava na cabeça
        }
        else return null;  //não podemos retirar de uma fila vazia
}
```



Implementação de uma fila estática sequencial (7)

```
//Retorna o conteúdo da fila (queue), sem alterar a mesma
public String toString () {
        if( !isEmpty() ) {
                String saida = "";
                int pos = head;  //variável auxiliar, para não alterar head
                for(int i=1; i<=total; i++) {  // total é a qtde. de elementos
                        saida += memo[pos].toString();
                        if ( i != total )  saida += ", ";  //ou  saida += "\n";
                        pos++;
                        if ( pos >= MAX ) pos = 0;
                }
                return ( "F: " + " [ " + saida + " ]" );
        }
        else return ( "F: [ ]" );  // se a fila está vazia
}
```



Universidade Cruzeiro do Sul

Testando as operações da fila...

```
Queue f = new Queue();
System.out.println( f.toString() ); //fila vazia neste momento
if(f.peek() == null)System.out.println( "Não foi possível peek, porque a fila está vazia..." );
f.enqueue("mouse");
System.out.println( f.toString() );
f.enqueue("pendrive");
f.enqueue("smartphone");
System.out.println( f.toString() );
System.out.println("Retirado: " + f.dequeue());
System.out.println( f.toString() );
System.out.println("Retirado: " + f.dequeue());
System.out.println( f.toString() );
System.out.println("Retirado: " + f.dequeue());
System.out.println( f.toString() );
f.enqueue("notebook");
f.enqueue("tablet");
f.enqueue("smartwatch");
if ( f.enqueue("drone") == null ) System.out.println( "Não foi possível inserir...");
System.out.println( f.toString() );
System.out.println( "Como primeiro da fila se encontra: " + f.peek() );
System.out.println( "Retirando (eliminando) um item da fila de cada vez:" );
while ( !f.isEmpty() ) {
    System.out.println("--- retirado: " + f.dequeue());
}
System.out.println( f.toString() );
```

Testando a fila - saída na tela do exemplo anterior

```
run:
F: [ ]
Não foi possível peek, porque a fila está vazia...
F:  [ mouse ]
F:  [ mouse, pendrive, smartphone ]
Retirado: mouse
F:  [ pendrive, smartphone ]
Retirado: pendrive
F:  [ smartphone ]
Retirado: smartphone
F: [ ]
F:  [ notebook, tablet, smartwatch, drone ]
Como primeiro da fila se encontra: notebook
Retirando (eliminando) um item da fila de cada vez:
--- retirado: notebook
--- retirado: tablet
--- retirado: smartwatch
--- retirado: drone
F: [ ]
```

Exercício combinado de fila e pilha, para praticar

Sejam uma pilha p e uma fila f, com estados iniciais:

p: [a,b,c] e f: [d,e,h]

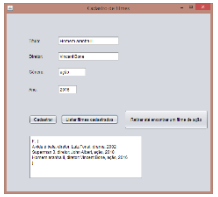
Quais serão os estados finais de p e f depois de executar os comandos a seguir?

```
p.push(w)
f.enqueue(t)
f.enqueue(p.pop())
p.push(f.dequeue())
p.push(f.peek())
```

Respostas: p: [...] f: [...]

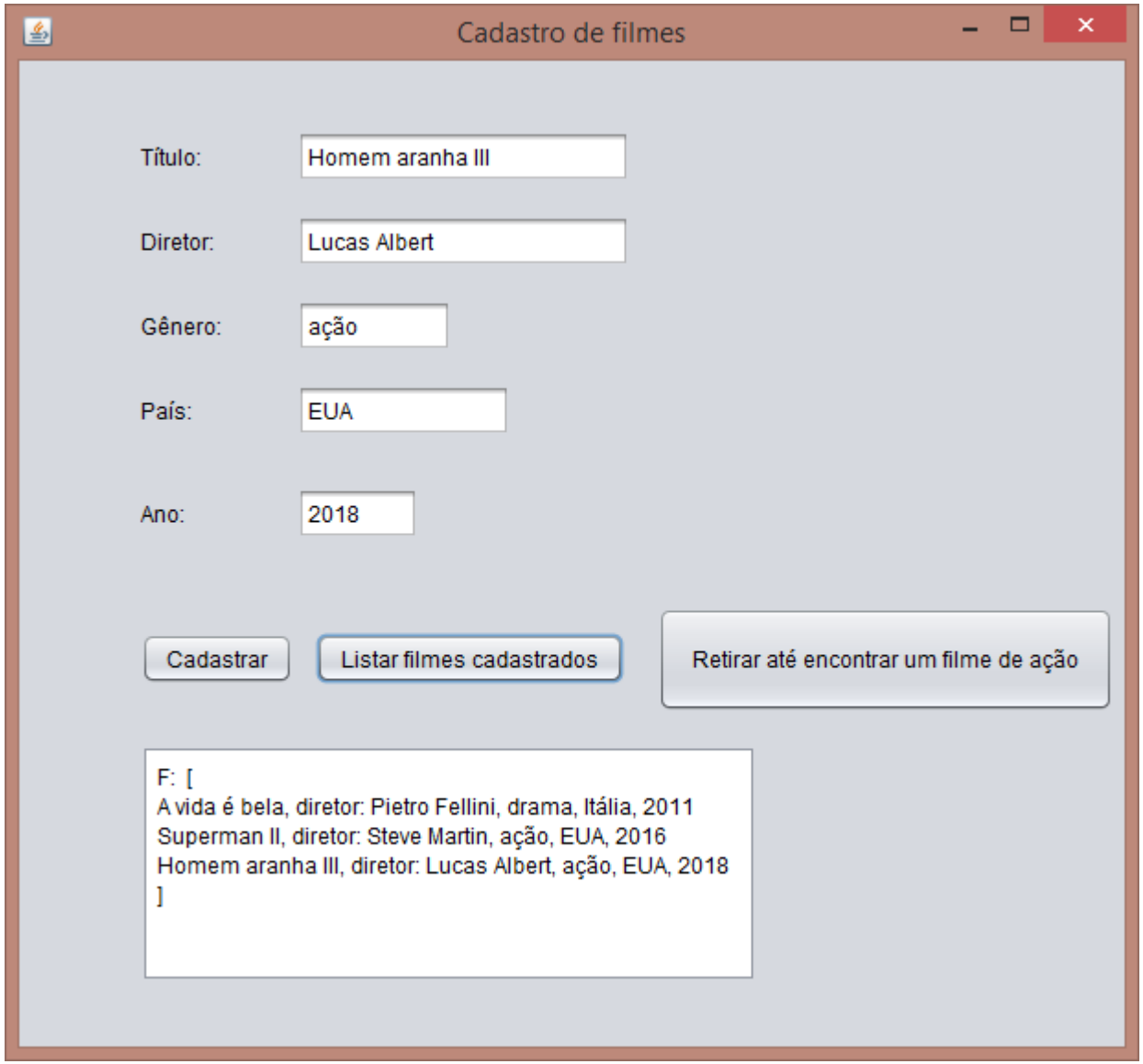
Exercício final para praticar e entregar

- Criar uma fila com dados de filmes. Insira na fila objetos f1, f2,... f5..., de pelo menos cinco filmes. Atributos para a classe Filme: título, diretor, gênero, país, ano.
- Opção: eliminar da fila (mostrar antes) todos os filmes que não sejam do gênero "ação", ou seja, extrair até encontrar o primeiro filme de "ação", filme que não será retirado da fila, mas mostraremos seus dados na tela.
- O ciclo anterior anterior também deverá parar se a fila fica vazia.
- Crie uma interface gráfica para efetuar o cadastro de filmes e efetuar as consultas.



veja visual

Exercício final para praticar e entregar



Bibliografia oficial da disciplina

BIBLIOGRAFIA BÁSICA	BIBLIOGRAFIA COMPLEMENTAR
CORMEN, T. H.; et al. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012. GOODRICH, M. T.; TAMASSIA, R. Estrutura de dados e algoritmos em java. 5. ed. Porto Alegre: Bookman, 2013. (livro físico e e-book) CURY, T. E., BARRETO, J. S., SARAIVA, M. O., et al. Estrutura de Dados (1. ed.) ISBN 9788595024328, Porto Alegre: SAGAH, 2018 (e-book)	ASCENCIO, A. F. G.; ARAÚJO, G. S. Estruturas de Dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++. São Paulo: Pearson Prentice Hall, 2010. (eBook) PUGA, S.; RISSETTI, G. Lógica de programação e estruturas de dados, com aplicações em Java. 3. ed. São Paulo: Pearson Education do Brasil, 2016. (eBook) DEITEL, P.;DEITEL, H. Java como programar. 10. ed. São Paulo: Pearson Education do Brasil, 2017. (eBook) BARNES, D. J.; KOLLING, M. Programação Orientada a Objetos com Java: uma introdução prática usando o Blue J. São Paulo: Pearson Prentice Hall, 2004. (eBook) BORIN, V. POZZOBON. Estrutura de Dados. ISBN: 9786557451595, Edição: 1ª . Curitiba: Contentus, 2020 (e-book)