

Tipo abstrato de dados - Filas



Atividades previstas para esta aula:

- 1) Introdução.
- 2) Aplicações
- 3) Operações.
- 4) Exemplo da utilização das operações da fila.
- 5) *Warm up.*



1) **Introdução**





Fila:

- Uma “fila” (do inglês *stack*) é uma coleção de objetos que são *inseridos* e *removidos* de acordo com o princípio de que o primeiro que entra é o primeiro que sai (FIFO, do inglês *first-in, first-out*).



Fila:

- A estrutura deriva da metáfora de uma fila de pessoas esperando para andar em um brinquedo de parque de diversões.



2) Aplicações





Aplicações da estrutura de Fila:

- Estrutura utilizada no tratamento de chamadas para uma central de reservas da bilheteria de um cinema.
- Fila de arquivos para impressões.
- *Buffer* para gravação de dados em uma mídia.
- Processo de comunicação em redes de computadores.



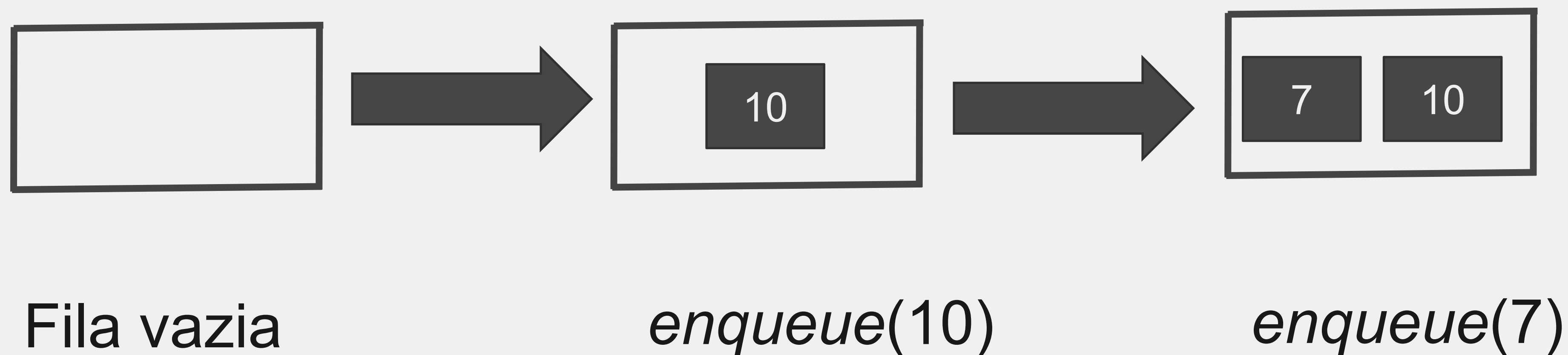
3) Operações





Operações:

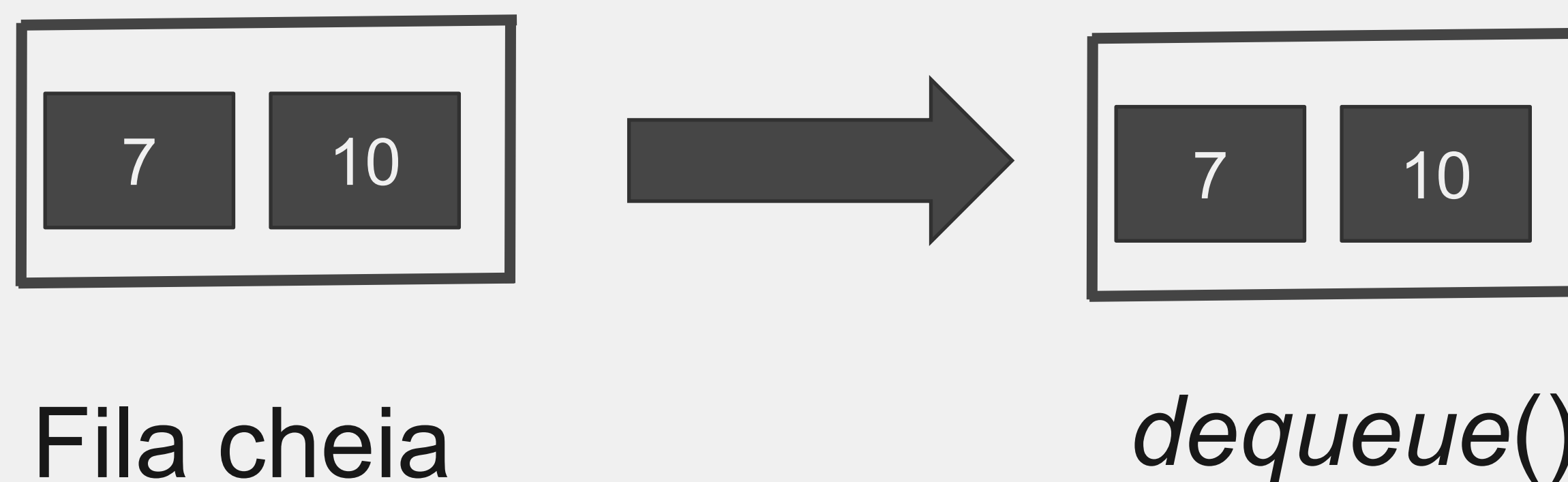
- *enqueue(e)* – Insere o elemento **e** no fim da fila.





Operações:

- *dequeue()* – Retira e retorna o elemento da frente da fila.
Ocorre um erro se a fila estiver vazia.





Operações adicionais:

- *size()* - Retorna o número de elementos da fila.
- *isEmpty()* - Retorna um booleano indicando se a fila está vazia
- *front()* - Retorna o elemento que está na frente da pilha, sem retirá-lo; ocorre um erro se a fila estiver vazia.



4) Exemplo de utilização das operações da fila



Condições iniciais:

- A fila de inteiros Q está inicialmente vazia.

Operação	Saída	Frente Q Fim
enqueue(5)		
enqueue(7)		
dequeue()		
enqueue(5)		
dequeue()		
front()		
isEmpty()		
size()		



5) ***Warm up***





Condições iniciais:

- Se reunir em equipes de 5 pessoas.
- Escrever as respostas em um documento *word* para entregar.





5) *Warm up:*

a) Como utilizar a estrutura de fila para simular o jogo infantil “*batata-quente*”? Desenvolva um pseudocódigo.



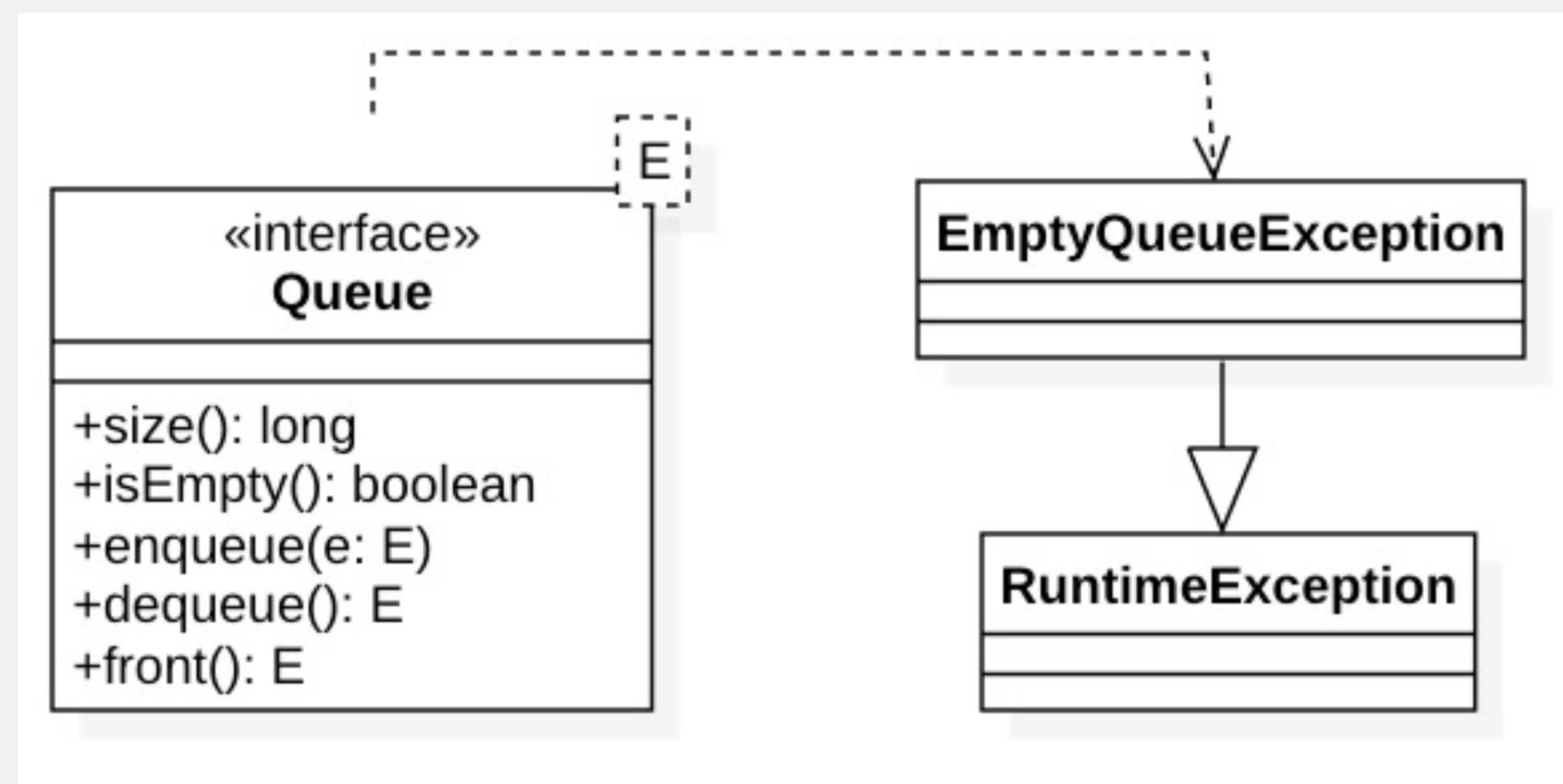
6) **Interface da fila**





Interface para filas em Java:

- Exceção *EmptyQueueException* é lançada pelos métodos `dequeue()` ou `front()` quando a fila está vazia.





Classe EmptyQueueException:

```
public class EmptyQueueException extends RuntimeException{  
  
    public EmptyQueueException(String error) {  
        super(error);  
    }  
  
}
```





Interface Stack:

```
public interface Queue<E> {  
    public int size();  
    public boolean isEmpty();  
    public E front() throws EmptyQueueException;  
    public void enqueue(E element);  
    public E dequeue() throws EmptyQueueException;  
}
```



7) Implementação da fila baseada em arranjos





Implementação de uma fila baseada em arranjos:

- A fila pode armazenar os elementos em um arranjo. Uma possibilidade é considerar que o $Q[0]$ seja a frente da fila e deixar a fila crescer. Porém não é uma solução eficiente.
- **Por quê?**





Implementação de uma fila baseada em arranjos:

- Para evitar mover objetos, definem-se duas variáveis f e r que possuem os seguintes significados:
- f é um índice de uma célula de Q que guarda o primeiro elemento da fila, a não ser que a fila esteja vazia.
- r é um índice para a próxima posição livre em Q .
- Inicialmente, atribui-se $f = r = 0$, ou seja fila vazia.






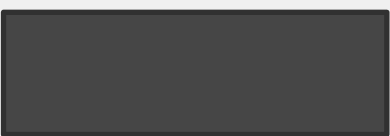





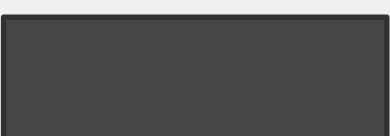
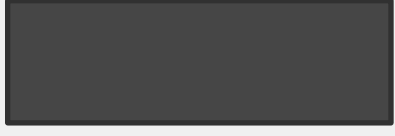
Implementação de uma fila baseada em arranjos:

- Quando se remove um elemento da frente da fila, incrementa-se f para indicar a próxima célula.
- Quando se acrescenta um elemento, ele é armazenado em $Q[r]$, e incrementa-se r para indicar a próxima célula livre em Q .
- Métodos *front*, *enqueue* e *dequeue* em tempo constante $O(1)$.
- Porém, surge outro problema.



Implementação de uma fila baseada em arranjos:

- Considere o seguinte cenário, com uma pilha de inteiros Q com tamanho 2.

Operação		
Início		
enqueue(7)		
enqueue(4)		
dequeue()		
dequeue()		
enqueue(8)		

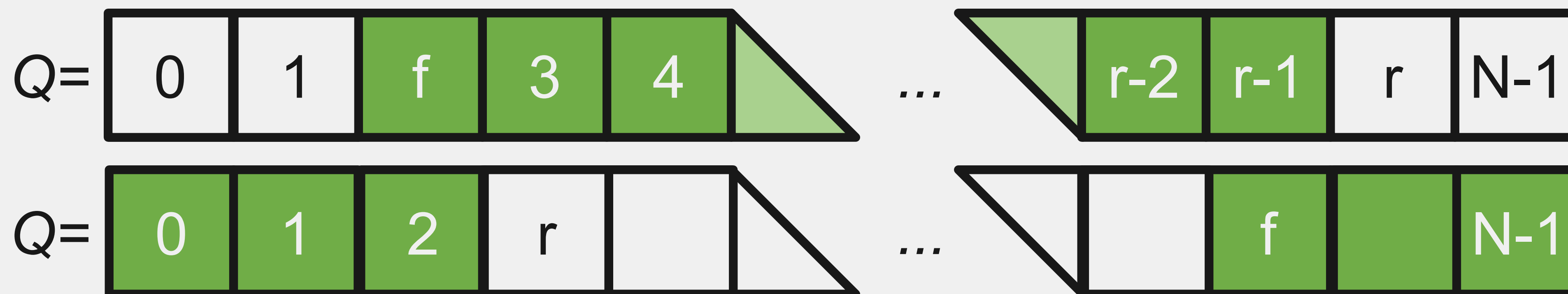
Erro índice fora de faixa
Não existe índice 2 num vetor tamanho 2





Implementação de uma fila baseada em arranjos:

- Para evitar este problema e voltar utilizar o arranjo Q , faz-se com que os índices f e r “façam a volta” ao final de Q .
- Passa considerar um arranjo circular que vai de $Q[0]$ a $Q[N-1]$ e recomeça em $Q[0]$ outra vez.





Implementação de uma fila baseada em arranjos:

- Visão circular de Q é simples. Toda vez que se incrementa f ou r , simplesmente calcula-se este incremento como $(f+1) \bmod N$ ou $(r+1) \bmod N$, respectivamente.
- A exceção **FullQueueException** é utilizada para indicar que o arranjo está cheio. É lançada quando é tentado inserir um novo elemento na fila.





Implementação de uma fila baseada em arranjos:

```
Algoritmo size():  
    retorna (N-f+r) mod N
```

```
Algoritmo isEmpty():  
    retorna (f == r)
```

```
Algoritmo front():  
    se isEmpty() então  
        lançar uma EmptyQueueException  
    retorna Q[f];
```

Complexidade:	O(1)
---------------	------

Complexidade:	O(1)
---------------	------

Complexidade:	O(1)
---------------	------



Implementação de uma fila baseada em arranjos:

Algoritmo enqueue(*e*):
 se size() == $N - 1$ **então**
 lançar uma FullQueueException
 $S[r] \leftarrow e$
 $r \leftarrow (r + 1) \bmod N$

Algoritmo dequeue():
 se isEmpty() **então**
 lançar uma EmptyQueueException
 $temp \leftarrow Q[f]$
 $Q[f] \leftarrow \text{null}$
 $f \leftarrow (f + 1) \bmod N$
 retorna *temp*;

Complexidade: $O(1)$

Complexidade: $O(1)$



Implementação concreta da fila:

```
public class ArrayQueue<E> implements Queue<E>{
    protected int f = 0;
    protected int r = 0;
    protected int capacity;
    public static final int CAPACITY = 1000;
    protected E Q[];
    public ArrayQueue() {
        this(CAPACITY);
    }
    public ArrayQueue(int capacity) {
        this.capacity = capacity;
        Q = (E[])new Object[this.capacity];
    }
}
```



8) **Codificação da fila baseada em arranjos**





Condições iniciais:

- Faça o *download* do projeto ***queues-implementations-project*** em <https://github.com/leandersonandre/estruturas-de-dados>.
- Projeto criado com Eclipse IDE 2019-03





8) *Codificação:*

a) Desenvolva os métodos da fila, classe

ArrayQueue.java, com base nos pseudocódigos.





8) *Codificação:*

b) Verifique a corretude dos métodos através dos testes unitários (classe **ArrayQueueTests.java**).



**9) Limitação da fila
baseada em arranjos**





Limitação de uma fila baseada em arranjos:

- A fila baseada em arranjos é simples e eficiente.
- Sua limitação consiste em **assumir um limite superior fixo de memória.**
- Leva a subutilização da memória.
- Estouro da pilha, lançando a exceção `FullQueueException`.
- Tipo Abstrato de Dados Pilha é veloz e útil, principalmente quando se tem uma boa estimativa do número de elementos a ser armazenados.



10) Implementação da fila baseada em lista encadeada





Implementação de uma fila baseada em lista encadeada:

- O *head* da lista encadeada é considerado o front da fila.
- O *tail* da lista encadeada é considerado o final da fila.
- Dessa forma, remove-se da frente da fila e insere-se no final.
- Necessário utilizar uma variável *size* para contar o número de elementos.





Implementação de uma fila baseada em lista encadeada:

Algoritmo size():
 retorna size

Algoritmo isEmpty():
 retorna (head == null)

Algoritmo front():
 se isEmpty() **então**
 lançar uma EmptyQueueException
 retorna head.getElement();

Complexidade:	O(1)
---------------	------

Complexidade:	O(1)
---------------	------

Complexidade:	O(1)
---------------	------



Implementação de uma fila baseada em lista encadeada:

```
Algoritmo enqueue(e):  
  v ← criar novo Node<E>(elem, null)  
  se size() == 0 então  
    head ← v  
  senão  
    tail.setNext(v)  
  tail = v  
  size++
```

Complexidade:	$O(1)$
---------------	--------





Implementação de uma fila baseada em lista encadeada:

Algoritmo dequeue():
 se isEmpty() **então**
 lançar uma EmptyQueueException
 temp ← head.getElement()
 head ← head.getNext()
 size ← size-1
 se isEmpty() **então**
 tail ← null
 retorna temp;

Complexidade:	$O(1)$
---------------	--------



Implementação concreta da fila:

```
public class NodeQueue<E> implements Queue<E>{  
    protected int size;  
    protected Node<E> head;  
    protected Node<E> tail;  
    public NodeQueue() {  
        head = null;  
        tail = null;  
        size = 0;  
    }  
}
```



11) Codificação da fila baseada em lista encadeada





Condições iniciais:

- Faça o *download* do projeto ***queues-implementations-project*** em <https://github.com/leandersonandre/estruturas-de-dados>.
- Projeto criado com Eclipse IDE 2019-03





11) *Codificação:*

a) Desenvolva os métodos da fila, classe

NodeQueue.java, com base nos pseudocódigos.





11) *Codificação:*

b) Verifique a corretude dos métodos através dos testes unitários (classe **NodeQueueTests.java**).



Obrigado!

Bacharel em Engenharia de Software
Bacharel em Sistemas de Informação

Prof. MSc. Leanderson André
leandersonandre@univille.br

