

# Filas de prioridade

## Conceitos e implementação

Prof. Marcelo de Souza

45RPE – Resolução de Problemas com Estruturas de Dados  
Universidade do Estado de Santa Catarina



## Leitura principal:

- ▶ Capítulo 9 de [Goodrich et al. \(2014\)](#)<sup>1</sup> – Filas de prioridade.

## Leitura complementar:

- ▶ Capítulo 6 de [Szwarcfiter e Markenzon \(2009\)](#)<sup>2</sup> – Listas de prioridades.
- ▶ Capítulo 4 de [Lafore e Machado \(2004\)](#)<sup>3</sup> – Pilhas e filas.

---

<sup>1</sup>Michael T Goodrich et al. (2014). *Data structures and algorithms in Java*. 6ª ed. John Wiley & Sons.

<sup>2</sup>Jayme Luiz Szwarcfiter e Lilian Markenzon (2009). *Estruturas de Dados e seus Algoritmos*. Vol. 2. Livros Técnicos e Científicos.

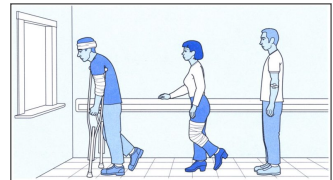
<sup>3</sup>Robert Lafore e Eveline Vieira Machado (2004). *Estruturas de dados & Algoritmos em Java*. Ciência Moderna.

# Filas de prioridade

## Ideia geral

**Fila de prioridade:** cada elemento tem uma prioridade, que determina a ordem de remoção.

- ▶ O elemento prioritário é o próximo a ser removido.
- ▶ **Atendimento médico:** a gravidade do paciente define sua prioridade de atendimento.



# Filas de prioridade

## Ideia geral

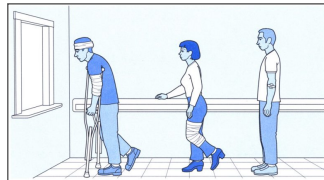


**Fila de prioridade:** cada elemento tem uma prioridade, que determina a ordem de remoção.

- ▶ O elemento prioritário é o próximo a ser removido.
- ▶ **Atendimento médico:** a gravidade do paciente define sua prioridade de atendimento.

### Operações:

- ▶ `insert`: insere um elemento na fila.
- ▶ `min`: retorna o elemento prioritário da fila.
- ▶ `removeMin`: remove (e retorna) o elemento prioritário da fila.



# Filas de prioridade

## Ideia geral

**Fila de prioridade:** cada elemento tem uma prioridade, que determina a ordem de remoção.

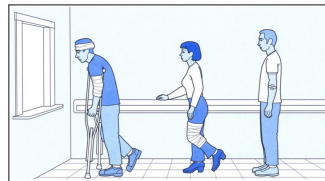
- ▶ O elemento prioritário é o próximo a ser removido.
- ▶ **Atendimento médico:** a gravidade do paciente define sua prioridade de atendimento.

## Operações:

- ▶ `insert`: insere um elemento na fila.
- ▶ `min`: retorna o elemento prioritário da fila.
- ▶ `removeMin`: remove (e retorna) o elemento prioritário da fila.

## Aplicações:

- ▶ Busca em grafos (e.g. algoritmo de Dijkstra).
- ▶ Otimização combinatória (e.g. bin packing).
- ▶ Inteligência artificial (e.g. algoritmo A\*).



# Filas de prioridade

## Ideia geral



### Funcionamento

- ▶ A prioridade é definida do **menor** valor para o maior (i.e. o menor tem prioridade).
- ▶ Para armazenar tipos naturalmente comparáveis, basta usar a estrutura.
  - ▶ e.g. Integer, Long, Float, Double, String, ...



### Funcionamento

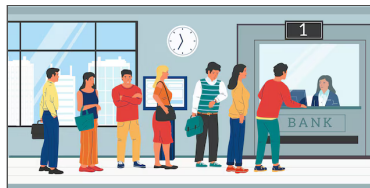
- ▶ A prioridade é definida do **menor** valor para o maior (i.e. o menor tem prioridade).
- ▶ Para armazenar tipos naturalmente comparáveis, basta usar a estrutura.
  - ▶ e.g. Integer, Long, Float, Double, String, ...
- ▶ Para armazenar objetos de outros tipos, eles precisam ser comparáveis.
  1. A classe deve implementar a interface Comparable.
  2. A classe deve implementar o método `compareTo`.

# Filas de prioridade

## Ideia geral

### Funcionamento

- ▶ A prioridade é definida do **menor** valor para o maior (i.e. o menor tem prioridade).
- ▶ Para armazenar tipos naturalmente comparáveis, basta usar a estrutura.
  - ▶ e.g. Integer, Long, Float, Double, String, ...
- ▶ Para armazenar objetos de outros tipos, eles precisam ser comparáveis.
  1. A classe deve implementar a interface Comparable.
  2. A classe deve implementar o método compareTo.
- ▶ Exemplo: **fila de um banco**.
  - ▶ Prioridade definida por vários atributos (idoso, gestante, cliente especial, tempo de chegada, ...).







# Filas de prioridade

## Exemplo de funcionamento

Seja uma fila de prioridade implementada usando uma lista sequencial, inicialmente vazia.

- Armazenaremos veículos, cuja prioridade é definida pelo ano de fabricação.

Método	Retorno	Conteúdo (não ordenado)
<code>insert((Fusca, 67))</code>	—	{ (Fusca, 67) }
<code>insert((Uno, 95))</code>	—	{ (Fusca, 67), (Uno, 95) }
<code>insert((Kombi, 60))</code>	—	{ (Fusca, 67), (Uno, 95), (Kombi, 60) }
<code>min()</code>	(Kombi, 60)	{ (Fusca, 67), (Uno, 95), (Kombi, 60) }
<code>removeMin()</code>	(Kombi, 60)	{ (Fusca, 67), (Uno, 95) }
<code>insert((Corcel, 74))</code>	—	{ (Fusca, 67), (Uno, 95), (Corcel, 74) }
<code>removeMin()</code>	(Fusca, 67)	{ (Uno, 95), (Corcel, 74) }
<code>removeMin()</code>	(Corcel, 74)	{ (Uno, 95) }
<code>removeMin()</code>	(Uno, 95)	{ }
<code>removeMin()</code>	null	{ }
<code>isEmpty()</code>	true	{ }



# Filas de prioridade

## Exemplo de funcionamento

Seja uma fila de prioridade implementada usando uma lista sequencial, inicialmente vazia.

- ▶ Armazenaremos veículos, cuja prioridade é definida pelo ano de fabricação.

Método	Retorno	Conteúdo ( <b>ordenado</b> )
insert((Fusca, 67))	–	{ (Fusca, 67) }
insert((Uno, 95))	–	{ (Uno, 95), (Fusca, 67) }
insert((Kombi, 60))	–	{ (Uno, 95), (Fusca, 67), (Kombi, 60) }
min()	(Kombi, 60)	{ (Uno, 95), (Fusca, 67), (Kombi, 60) }
removeMin()	(Kombi, 60)	{ (Uno, 95), (Fusca, 67) }
insert((Corcel, 74))	–	{ (Uno, 95), (Corcel, 74), (Fusca, 67) }
removeMin()	(Fusca, 67)	{ (Uno, 95), (Corcel, 74) }
removeMin()	(Corcel, 74)	{ (Uno, 95) }
removeMin()	(Uno, 95)	{ }
removeMin()	null	{ }
isEmpty()	true	{ }

# Filas de prioridade

## Análise de complexidade



Podemos implementar uma fila de prioridade usando **arranjos** ou **encadeamento**. E ainda podemos manter a estrutura **não ordenada** ou **ordenada**.



# Filas de prioridade

## Análise de complexidade

Podemos implementar uma fila de prioridade usando **arranjos** ou **encadeamento**. E ainda podemos manter a estrutura **não ordenada** ou **ordenada**.

Operação	Não ordenado		Ordenado	
	Arranjo	Lista encadeada	Arranjo	Lista encadeada
insert	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
min	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
removeMin	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

— não estamos considerando o tempo gasto com *resize* no arranjo.



# Filas de prioridade

## Análise de complexidade

Podemos implementar uma fila de prioridade usando **arranjos** ou **encadeamento**. E ainda podemos manter a estrutura **não ordenada** ou **ordenada**.

Operação	Não ordenado		Ordenado		Heap
	Arranjo	Lista encadeada	Arranjo	Lista encadeada	
insert	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
min	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
removeMin	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$

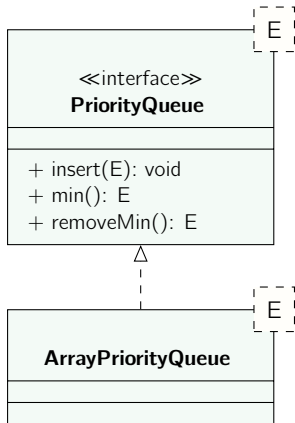
— não estamos considerando o tempo gasto com *resize* no arranjo.

Uma **heap** é um tipo de **árvore binária** usado para implementar filas de prioridade.

- ▶ É a forma como uma fila de prioridade é implementada pelo Java.
- ▶ Estudaremos esse tipo de estrutura mais adiante!

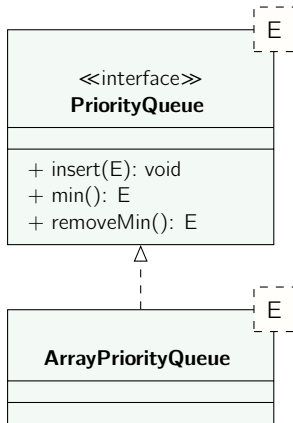
# Filas de prioridades

## Implementação



# Filas de prioridades

## Implementação



## Implementação e uso:

```
data-structures
├── priority-queue
│   ├── {PriorityQueue, ArrayPriorityQueue}.java
│   └── TestPriorityQueue[Patient].java
```

## Veja também:

- ▶ `java.util.PriorityQueue`<sup>1</sup>
  - ▶ Implementação baseada em uma *heap* binária.
- ▶ Apêndice I

<sup>1</sup>Veja em <https://docs.oracle.com/en/java/javase/24/docs/api/java.base/java/util/PriorityQueue.html>.

## Apêndices



# Apêndice I

## Implementação de filas de prioridade (interface)



Interface PriorityQueue:

```
1  public interface PriorityQueue<E> {  
2      int size();  
3      boolean isEmpty();  
4      void insert(E e);  
5      E min();  
6      E removeMin();  
7  }
```

# Apêndice I

## Implementação de filas de prioridade (interface)



### Interface PriorityQueue:

```
1 public interface PriorityQueue<E> {  
2     int size();  
3     boolean isEmpty();  
4     void insert(E e);  
5     E min();  
6     E removeMin();  
7 }
```

- ▶ A fila de prioridade é uma coleção de elementos do tipo genérico E.
- ▶ As operações `min` e `removeMin` retornam o elemento sendo consultado e removido, respectivamente.



# Apêndice I

## Implementação de filas de prioridade (arranjo ordenado)

```
1 public class ArrayPriorityQueue<E extends Comparable<? super E>> implements PriorityQueue<E> {  
2  
3     private List<E> list = new ArrayList<>();  
4     public int size() { return list.size(); }  
5     public boolean isEmpty() { return size() == 0; }  
6  
7     //...  
8 }
```



# Apêndice I

## Implementação de filas de prioridade (arranjo ordenado)

```
1 public class ArrayPriorityQueue<E extends Comparable<? super E>> implements PriorityQueue<E> {  
2  
3     private List<E> list = new ArrayList<>();  
4     public int size() { return list.size(); }  
5     public boolean isEmpty() { return size() == 0; }  
6  
7     //...  
8 }
```

- ▶ Na definição do tipo genérico E, a classe exige que o tipo seja comparável, para definir a prioridade de cada elemento armazenado (linha 1).
- ▶ A classe usa um ArrayList como estrutura fundamental (linha 3).

# Apêndice I

## Implementação de filas de prioridade (arranjo ordenado)



Método insert:

```
1 public void insert(E e) {  
2     if(isEmpty()) list.add(0, e);  
3     else {  
4         int j = 0;  
5         while(j < size() && e.compareTo(list.get(j)) < 0)  
6             j++;  
7         list.add(j, e);  
8     }  
9 }
```

# Apêndice I

## Implementação de filas de prioridade (arranjo ordenado)



Método insert:

```
1 public void insert(E e) {  
2     if(isEmpty()) list.add(0, e);  
3     else {  
4         int j = 0;  
5         while(j < size() && e.compareTo(list.get(j)) < 0)  
6             j++;  
7         list.add(j, e);  
8     }  
9 }
```

- ▶ Caso a fila esteja vazia, o elemento é inserido na primeira posição (linha 2).
- ▶ Caso contrário, as linhas 4 a 7 buscam a posição e o inserem mantendo a ordenação.

# Apêndice I

## Implementação de filas de prioridade (arranjo ordenado)



Método min:

```
1 public E min() {  
2     if (isEmpty()) return null;  
3     return list.get(size() - 1);  
4 }
```

Método removeMin:

```
1 public E removeMin() {  
2     if (isEmpty()) return null;  
3     return list.remove(size() - 1);  
4 }
```

45RPE – Resolução de Problemas com Estruturas de Dados  
Prof. Marcelo de Souza