Capítulo 7

Filas

"O êxito não se consegue só com qualidades especiais. É sobretudo um trabalho de constância, de método e de organização"
— J. P. Sergent

7.1 - Introdução

No dia a dia, estamos acostumados com as filas em diversos lugares: nos bancos, nos mercados, nos hospitais, nos cinemas entre outros. As filas são importantes pois elas determinam a ordem de atendimento das pessoas.

As pessoas são atendidas conforme a posição delas na fila. O próximo a ser atendido é o primeiro da fila. Quando o primeiro da fila é chamado para ser atendido a fila "anda", ou seja, o segundo passa a ser o primeiro, o terceiro passa a ser o segundo e assim por diante até a última pessoa.

Normalmente, para entrar em uma fila, uma pessoa deve se colocar na última posição, ou seja, no fim da fila. Desta forma, quem chega primeiro tem prioridade.

Neste capítulo, estamos interessados em desenvolver estrutura de dados com o comportamentos das filas. Assim como Listas e Pilhas, as Filas são estruturas de dados que armazenam os elementos de maneira sequencial.

Assim como as Pilhas, as Filas têm operações mais **restritas** do que as operações das Listas. Nas Filas, os elementos são adicionados na última posição e removidos da primeira posição. Nas Listas, os elementos são adicionados e removidos de qualquer posição.

Então, podemos implementar uma Fila simplesmente colocando as restrições adequadas nas operações de adicionar e remover elementos de uma Lista. Isso é bem parecido ao que fizemos com as Pilhas.

Vamos implementar na seqüência uma Fila de Aluno. Iremos aproveitar a classe Aluno feita no capítulo de armazenamento sequencial.

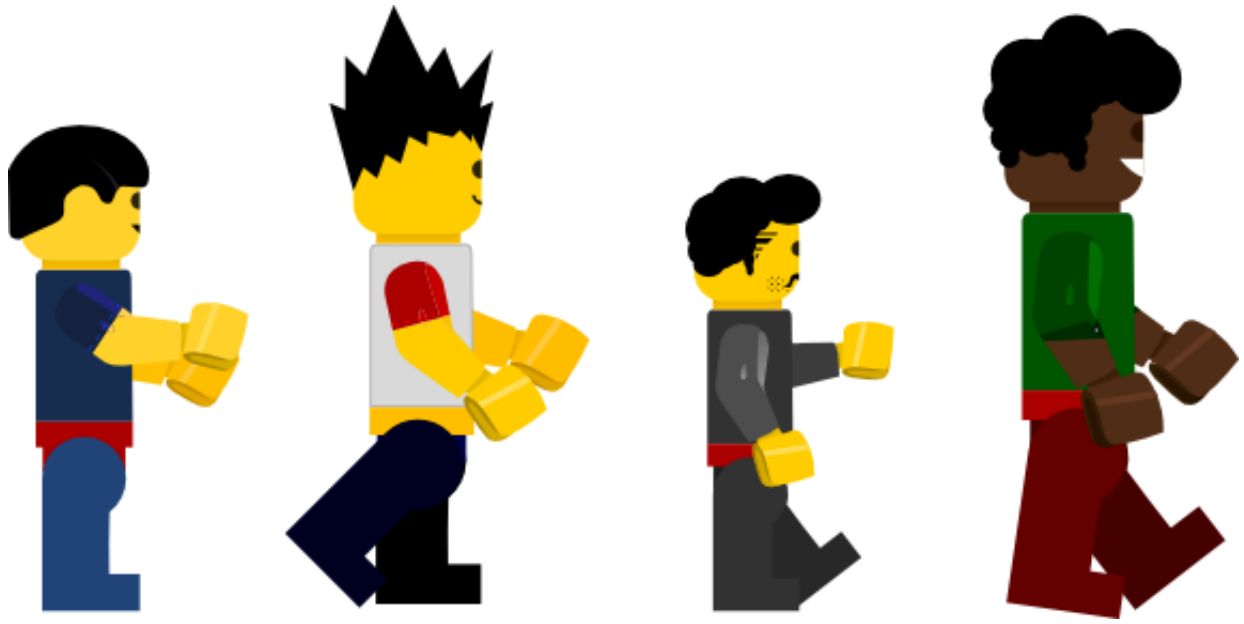

Figura 7.1: Fila de pessoas

7.2 – Interface de uso

As operações que formam a interface de uso da Fila de alunos são:

1. Insere um Aluno (coloca um aluno no fim da Fila).
2. Remove um Aluno (retira o aluno que está no começo da Fila).
3. Informa se a Fila está vazia.

O esboço da classe `Fila` seria mais ou menos assim:

```
public class Fila {

  public void insere(Aluno aluno) {
    // implementação
  }

  public Aluno remove() {
    // implementação
  }

  public boolean vazia() {
    // implementação
  }
}
```

Agora que já temos a interface de uso da Fila definida vamos escrever algum teste sobre como ela deveria se comportar.

```
public class Teste {

  public static void main(String[] args) {
    Fila fila = new Fila();

    Aluno aluno = new Aluno();
    fila.insere(aluno);

    Aluno alunoRemovido = fila.remove();

    if (fila.vazia()) {
      System.out.println("A fila está vazia");
    }
  }
}
```

Como já foi dito aqui vamos implementar Fila utilizando algum tipo de Lista. Neste capítulo, vamos utilizar a classe `LinkedList` para armazenar os alunos da Fila.

```
public class Fila {

  private List<Aluno> alunos = new LinkedList<Aluno>();

}
```


7.3 – Operações em Fila

Em seguida, implementaremos as operações da Fila de aluno.

7.4 – Inserir uma aluno

Os alunos que entram na Fila devem sempre se colocar no fim da mesma. Vamos definir que o fim da Fila é o fim da Lista que estamos utilizando para implementar.

Então, entrar na Fila e adicionar no fim da Lista.

```
public class Fila {

  private List<Aluno> alunos = new LinkedList<Aluno>();

  public void insere(Aluno aluno) {
    this.alunos.add(aluno);
  }
}
```

7.5 – Remover um aluno

O próximo aluno a ser atendido é sempre o que está no início da Fila. No nosso caso, quem está no início da Fila é o aluno que está no início da Lista.

Então, basta remover o primeiro aluno.

```
public class Fila {

  private List<Aluno> alunos = new LinkedList<Aluno>();

  ...
  public Aluno remove() {
    return this.alunos.remove(0);
  }
}
```

É bom observar que se o método `remove()` for usado com a Fila vazia então uma exceção será lançada pois o método `removeFirst()` lança `IndexOutOfBoundsException` quando não existir elemento para remover.

Casa do Código, livros para o programador.

7.6 – Informar se a Fila está vazia

Para implementar esta operação basta verificar se o tamanho da Lista é zero.

```
public class Fila {

  private List<Fila> alunos = new LinkedList<Fila>();

  ...
  public boolean vazia() {
    return this.alunos.size() == 0;
  }
}
```

7.7 – Generalização

Nossa Fila só funciona para guardar objetos da classe `Aluno`. Vamos generalizá-la para poder armazenar qualquer tipo de objeto. Isso será feito utilizando a classe `Object` da qual todas as classes derivam direta ou indiretamente. Criaremos uma `LinkedList` de `Object` em vez de uma `LinkedList` de `Aluno`.

```
public class Fila {

  private List<Object> objetos = new LinkedList<Object>();

  public void insere(Object objeto) {
    this.objetos.add(objeto);
  }

  public Object remove() {
    return this.objetos.remove(0);
  }

  public boolean vazia() {
    return this.objetos.size() == 0;
  }
}
```

Agora, podemos guardar qualquer tipo de objeto na Fila. Isso é uma grande vantagem pois a classe `Fila` poderá ser reaproveitada em diversas ocasiões. Mas, há uma desvantagem, quando removemos um elemento da Fila não podemos garantir qual é o tipo de objeto que virá.

A solução para este problema é utilizar o recurso do Generics. A nossa classe

`Fila` vai ser uma classe parametrizada. Assim, quando criarmos uma Fila poderemos definir com qual tipo de objetos ela deve trabalhar.

Algo deste tipo:

```
Fila de alunos fila = new Fila de alunos();
```

Traduzindo este código para Java, ficaria assim:

```
Fila<Aluno> fila = new Fila<Aluno>();
```

Agora, precisamos parametrizar a classe Fila.

```
public class Fila<T> {

  private List<T> objetos = new LinkedList<T>();

  public void insere(T t) {
    this.objetos.add(t);
  }

  public T remove() {
    return this.objetos.remove(0);
  }

  public boolean vazia() {
    return this.objetos.size() == 0;
  }
}
```

Vamos criar duas Filas, uma para `Aluno` e outra para `String`.

```
public class Teste {

  public static void main(String[] args) {
    Fila<Aluno> fila = new Fila<Aluno>();

    Aluno aluno = new Aluno();
    fila.insere(aluno);

    Aluno alunoRemovido = fila.remove();

    if (fila.vazia()) {
      System.out.println("A fila está vazia");
    }

    Fila<String> filaDeString = new Fila<String>();
    filaDeString.insere("Adelaide");
    filaDeString.insere("Carolina");

    String carolina = filaDeString.remove();
    String adelaide = filaDeString.remove();

    System.out.println(carolina);
```

```
    System.out.println(adelaide);
  }
}
```

Em tempo de compilação, é verificado o tipo de objetos que estão sendo adicionados na Fila. Portanto, se você tentar inserir um objeto do tipo `Aluno` em uma Fila de `String` um erro de compilação será gerado.

```
public class Teste {

  public static void main(String[] args) {
    Aluno aluno = new Aluno();
    Fila<String> filaDeString = new Fila<String>();

    // este código não compila
    filaDeString.insere(aluno);
  }
}
```

7.8 - API do Java

Na biblioteca do Java, existe uma interface que define a estrutura de dados Fila. Essa interface chama-se `Queue`, umas das classes que implementam `Queue` é a `LinkedList`. O funcionamento fica extremamente parecido com a implementação que fizemos neste capítulo.

```
public class Teste {
  public static void main(String[] args) {

    Queue fila = new LinkedList();

    Aluno aluno = new Aluno();
    fila.offer(aluno);

    Aluno alunoRemovido = (Aluno)fila.poll();

    if(fila.isEmpty()){
      System.out.println("A fila está vazia");
    }
  }
}
```

Para evitar fazer casting de objetos, podemos utilizar o recurso de Generics aqui também.

```
public class Teste {
  public static void main(String[] args) {

    Queue<Aluno> fila = new LinkedList<Aluno>();
```

```
    Aluno aluno = new Aluno();
    fila.offer(aluno);

    Aluno alunoRemovido = fila.poll();

    if(fila.isEmpty()){
      System.out.println("A fila está vazia");
    }
  }
}
```


7.9 – Exercícios: Fila

1. Implemente a classe `Fila` para alunos vista neste capítulo. Coloque a classe no pacote **br.com.caelum.ed.filas**

```
package br.com.caelum.ed.filas;

import java.util.LinkedList;
import java.util.List;

import br.com.caelum.ed.Aluno;

public class Fila {

  private List<Aluno> alunos = new LinkedList<Aluno>();

  public void insere(Aluno aluno) {
    this.alunos.add(aluno);
  }

  public Aluno remove() {
    return this.alunos.remove(0);
  }

  public boolean vazia() {
    return this.alunos.size() == 0;
  }
}
```

Faça alguns testes.

```
package br.com.caelum.ed.filas;

import br.com.caelum.ed.Aluno;

public class Teste {

  public static void main(String[] args) {
    Fila fila = new Fila();

    Aluno aluno = new Aluno();
    fila.insere(aluno);

    Aluno alunoRemovido = fila.remove();

    if (aluno != alunoRemovido) {
      System.out.println("Erro: o aluno removido não é " +
        " igual ao que foi inserido");
    }

    if (!fila.vazia()) {
      System.out.println("Erro: A fila não está vazia");
    }
  }
}
```

Se não for impresso nenhuma mensagem de erro significa que a Fila está funcionando.

2. Implemente a classe `FilaGenerica` para objetos (genérica) vista neste capítulo. Coloque a classe no pacote **br.com.caelum.ed.filas**

```
package br.com.caelum.ed.filas;

import java.util.LinkedList;
import java.util.List;

public class FilaGenerica {

  private List<Object> objetos = new LinkedList<Object>();

  public void insere(Object objeto) {
    this.objetos.add(objeto);
  }

  public Object remove() {
    return this.objetos.remove(0);
  }

  public boolean vazia() {
    return this.objetos.size() == 0;
  }
}
```

Faça alguns testes.

```
package br.com.caelum.ed.filas;

import br.com.caelum.ed.Aluno;

public class TesteFilaGenerica {

  public static void main(String[] args) {
    FilaGenerica filaDeAlunos = new FilaGenerica();

    Aluno aluno = new Aluno();
    filaDeAlunos.insere(aluno);

    Aluno alunoRemovido = filaDeAlunos.remove();

    if (aluno != alunoRemovido) {
      System.out.println("Erro: o aluno removido não é igual " +
        " ao que foi inserido");
    }

    if (!filaDeAlunos.vazia()) {
      System.out.println("Erro: A fila não está vazia");
    }
  }
}
```

Perceba que a classe `TesteFilaGenerica` contém um erro de compilação. Quando você remove um elemento da `FilaGenerica` você recebe uma referência do tipo `Object` e não do tipo `Aluno`.

Altere a linha:

```
Aluno alunoRemovido = filaDeAlunos.remove();
```

Por:

```
Object alunoRemovido = filaDeAlunos.remove();
```

Isso faz o código compilar mas agora você não tem mais a garantia de tipo. Não sabe se a referência que você recebeu realmente está apontado para um objeto do tipo `Aluno`.

3. Implemente a classe `FilaParametrizada` utilizando o recurso do **Generics**. Coloque a classe no pacote **br.com.caelum.ed.filas**

```
package br.com.caelum.ed.filas;

import java.util.LinkedList;
import java.util.List;

public class FilaParametrizada<T> {
```

```
  private List<T> objetos = new LinkedList<T>();

  public void insere(T t) {
    this.objetos.add(t);
  }

  public T remove() {
    return this.objetos.remove(0);
  }

  public boolean vazia() {
    return this.objetos.size() == 0;
  }
}
```

Faça alguns testes:

```
package br.com.caelum.ed.filas;

import br.com.caelum.ed.Aluno;

public class TesteFilaGenerica {

  public static void main(String[] args) {
    FilaParametrizada<Aluno> filaDeAlunos =
        new FilaParametrizada<Aluno>();

    Aluno aluno = new Aluno();
    filaDeAlunos.insere(aluno);

    Aluno alunoRemovido = filaDeAlunos.remove();

    if (aluno != alunoRemovido) {
      System.out.println("Erro: o aluno removido não é igual " +
        " ao que foi inserido");
    }

    if (!filaDeAlunos.vazia()) {
      System.out.println("Erro: A fila não está vazia");
    }

    FilaParametrizada<String> filaDeString =
        new FilaParametrizada<String>();

    filaDeString.insere("Diana");
    filaDeString.insere("Joaquim");

    System.out.println(filaDeString.remove());
    System.out.println(filaDeString.remove());
  }
}
```

4. (**opcional**) É possível implementar a nossa Fila utilizando internamente uma `ArrayList` em vez de `LinkedList`? Teremos algum ganho ou perda no consumo de

tempo de alguma das operações? Mostre a diferença através de um código que adiciona e remova muita gente da fila.

CAPÍTULO ANTERIOR:

Pilhas

PRÓXIMO CAPÍTULO:

Armazenamento sem repetição com busca rápida

Casa do Código

Twitter