



Simulador de coleta de lixo



Aline Dias e Kaielly Sousa



Simulador

O trabalho proposto foi a criação de um simulador de coleta de lixo da cidade de Teresina, baseado nas cinco zonas urbanas (Sul, Norte, Centro, Leste e Sudeste) e no uso de caminhões com cargas definidas pelo projeto:

- **Caminhões pequenos: 2, 4, 8 e 10 toneladas;**
- **Caminhões grandes: 20 toneladas (operando nas estações de transferência até o aterro sanitário).**



●●● Pastas

```
▼ src
  ▼ Caminhos
    © CaminhaoGrande
    © CaminhaoGrandePadrao
    © CaminhaoPequeno
    © CaminhaoPequenoPadrao
  ▼ Estações
    © EstacaoPadrao
    © EstacaoTransferencia
  ▼ Simulador
    © Evento
    © Fila
    © FilaCaminhoesPequenos
    © No
    © Simulador
```

```
▼ Util
  © TempoUtil
  ▼ Zonas
    © ZonaUrbana
    © Main
  .gitignore
  Teste-simulador.iml
```



TADs

```
package Simulador;

/**
 * Classe que implementa uma fila genérica usando nós encadeados.
 * Funciona no modelo FIFO (First In, First Out).
 */
public class Fila<T> { 2 usages
    private No<T> inicio; 8 usages
    private No<T> fim; 5 usages
    private int tamanho; 5 usages

    /**
     * Cria uma fila vazia.
     */
    public Fila() { 1 usage
        this.inicio = null;
        this.fim = null;
        this.tamanho = 0;
    }

    /**
     * Adiciona um elemento no final da fila.
     * @param elemento Elemento a ser adicionado.
     */
    public void enfileirar(T elemento) { 2 usages
        No<T> novo = new No<>(elemento);
        if (fim != null) {
            fim.setProximo(novo);
        } else {
            inicio = novo;
        }
        fim = novo;
        tamanho++;
    }
}
```

```
/**
 * Retorna o tamanho atual da fila.
 * @return Número de elementos na fila.
 */
public int tamanho() { no usages
    return tamanho;
}

/**
 * Verifica se a fila está vazia.
 * @return true se vazia, false se tiver elementos.
 */
public boolean estaVazia() { no usages
    return tamanho == 0;
}

/**
 * Permite percorrer a fila usando um for-each.
 * @return Iterable dos elementos na ordem da fila.
 */
public Iterable<T> iterar() { 2 usages
    return () -> new java.util.Iterator<T>() {
        private No<T> atual = inicio; 4 usages

        public boolean hasNext() {
            return atual != null;
        }

        public T next() {
            T valor = atual.getElemento();
            atual = atual.getProximo();
            return valor;
        }
    };
}
}
```

```
package Simulador;

/**
 * Classe que representa um nó genérico para estruturas encadeadas,
 * como listas, pilhas e filas.
 *
 * @param <T> Tipo de dado armazenado no nó.
 */
public class No<T> { 8 usages
    private T elemento; 2 usages
    private No<T> proximo; 3 usages

    /**
     * Cria um nó com o elemento fornecido.
     * @param elemento Elemento a ser armazenado.
     */
    public No(T elemento) { 1 usage
        this.elemento = elemento;
        this.proximo = null;
    }

    /**
     * Retorna o elemento armazenado no nó.
     * @return Elemento do nó.
     */
    public T getElemento() { 2 usages
        return elemento;
    }

    /**
     * Retorna a referência para o próximo nó.
     * @return Próximo nó ou null se não houver.
     */
    public No<T> getProximo() { 2 usages
        return proximo;
    }

    /**
     * Define o próximo nó na sequência.
     * @param proximo Nó a ser vinculado como próximo.
     */
    public void setProximo(No<T> proximo) { 1 usage
        this.proximo = proximo;
    }
}
```

TADs

```
package Simulador;

/**
 * Classe que representa um nó genérico para estruturas encadeadas,
 * como listas, pilhas e filas.
 *
 * @param <T> Tipo de dado armazenado no nó.
 */
public class No<T> { 8 usages
    private T elemento; 2 usages
    private No<T> proximo; 3 usages

    /**
     * Cria um nó com o elemento fornecido.
     * @param elemento Elemento a ser armazenado.
     */
    public No(T elemento) { 1 usage
        this.elemento = elemento;
        this.proximo = null;
    }

    /**
     * Retorna o elemento armazenado no nó.
     * @return Elemento do nó.
     */
    public T getElemento() { 2 usages
        return elemento;
    }

    /**
     * Retorna a referência para o próximo nó.
     * @return Próximo nó ou null se não houver.
     */
    public No<T> getProximo() { 2 usages
        return proximo;
    }

    /**
     * Define o próximo nó na sequência.
     * @param proximo Nó a ser vinculada como próximo.
     */
    public void setProximo(No<T> proximo) { 1 usage
        this.proximo = proximo;
    }
}
```

```
public class FilaCaminhoesPequenos { no usages
    /**
     * public CaminhaoPequeno desenfileirar() { no usages
         if (vazia()) return null;
         CaminhaoPequeno valor = frente.valor;
         frente = frente.proximo;
         if (frente == null) tras = null;
         return valor;
     }

    /**
     * Retorna o caminhão na frente da fila sem remover.
     * @return Caminhão na frente ou null se vazia.
     */
    public CaminhaoPequeno espiar() { no usages
        return vazia() ? null : frente.valor;
    }

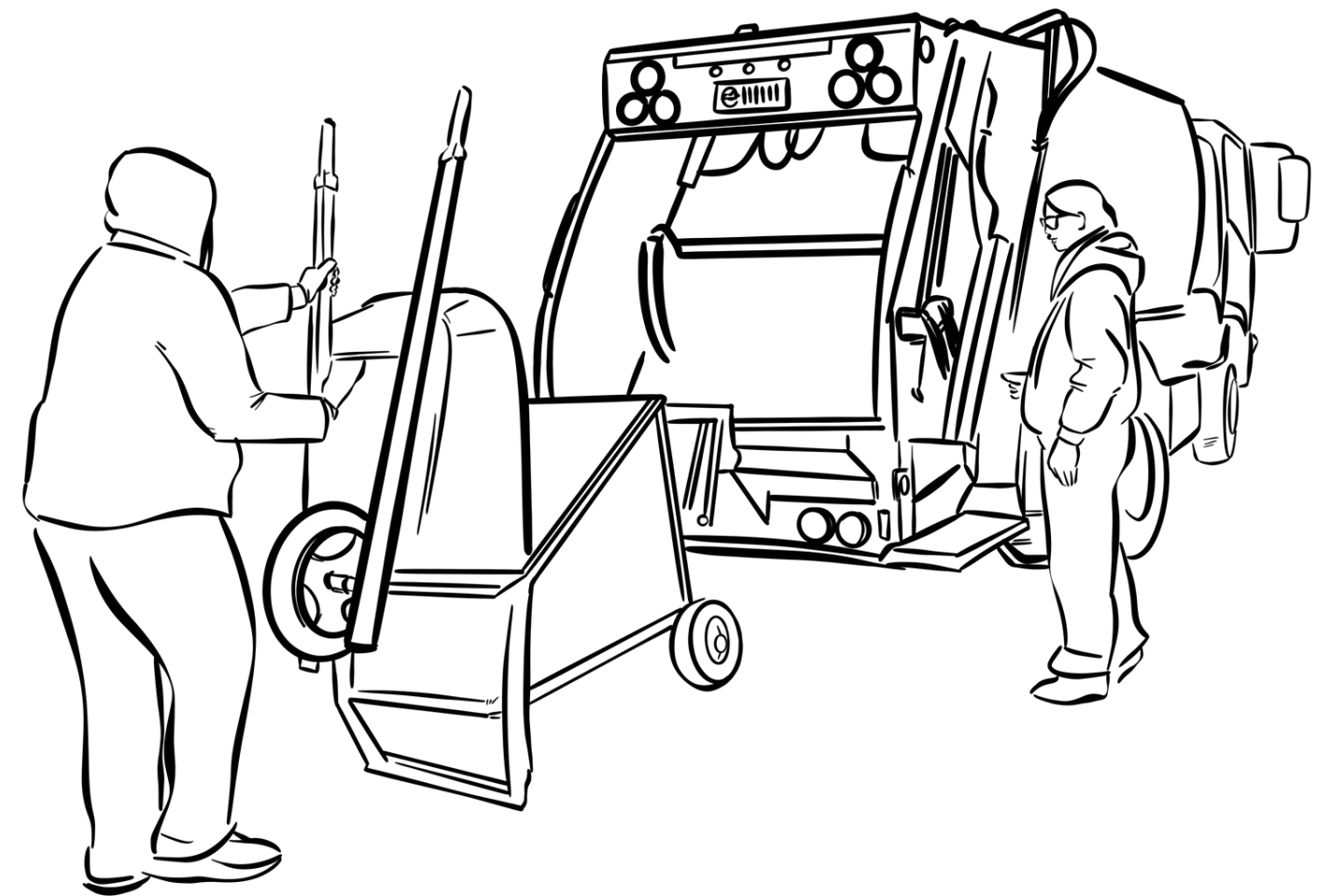
    /**
     * Verifica se a fila está vazia.
     * @return true se vazia, false se tiver elementos.
     */
    public boolean vazia() { 3 usages
        return frente == null;
    }

    /**
     * Retorna o nó da frente da fila.
     * @return Nó da frente.
     */
    public No getFrente() { no usages
        return frente;
    }



    /**
     * Permite percorrer a fila usando for-each.
     * @return Iterable dos caminhões na ordem da fila.
     */
    public Iterable<CaminhaoPequeno> iterar() { no usages
        return () -> new java.util.Iterator<CaminhaoPequeno>() {
            private No atual = frente; 4 usages

            public boolean hasNext() {
                return atual != null;
            }

            public CaminhaoPequeno next() {
                CaminhaoPequeno valor = atual.valor;
                atual = atual.proximo;
                return valor;
            }
        };
    }
}
```



Métodos das classes

 Classe	 Métodos Principais
CaminhaoPequeno	carregarLixo(), descarregar(), estaCheio()
CaminhaoPequenoPadrao	(Herda de CaminhaoPequeno)
CaminhaoGrande	carregarLixo(), descarregar(), estaCheio()
CaminhaoGrandePadrao	(Herda de CaminhaoGrande)
EstacaoPadrao	receberCaminhao(), liberarCaminhao(), filaVazia()
EstacaoTransferencia	descarregarNoCaminhaoGrande(), getCaminhaoGrande()
ZonaUrbana	gerarLixo(), getNome()
Fila	enqueue(), dequeue(), estaVazia(), tamanho(), iterar()
FilaCaminhoesPequenos	enqueue(), dequeue(), espiar(), vazia(), iterar()
No	(getter/setter de elemento e proximo)
Evento	getDescricao(), getTempo()
Simulador	inicializar(), executar(), processarEventos(), gerarRelatorio()
TempoUtil	converterHoraParaMinuto(), converterMinutoParaHora(), verificarHorarioDePico()



**Quantos Caminhões
de 20 toneladas no
mínimo o município
devera possuir para
atender a demanda da
geração de lixo em
Teresina?**

Total de lixo mensal: 40.000 toneladas.

Capacidade Caminhão: 20t

$40.000/20 = 2.000 \rightarrow$ Que faria de viagens

$2000/30 \cong 67$

por dia \rightarrow 2 viagens

$67/2 \cong 34$ caminhões




```
System.out.println("=====");
System.out.println("      TRABALHO ENTREGUE!");
System.out.println("=====");

System.out.println("-----isso foi um desafio-----.");
System.out.println("+++++");
System.out.println("-----Obrigada pela atenção!-----");
System.out.println("=====");
```