IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

PA 2021/22

Programação Avançada

Enunciado de Laboratório

1

ADT Queue e Unit Testing

Objetivos:

- Especificação de implementação de ADTs na linguagem Java;
- Desenvolvimento de testes unitários.

Introdução

Execute o IntelliJ e clone o projecto que se encontra no seguinte endereço:
https://github.com/estsetubal-pa-geral/lab1.git

As operações que deverão ser suportadas sobre uma *queue* Q são apresentadas de seguida:

Operações principais:

- **enqueue(e)** - insere o elemento e no final de Q; a operação deve resultar em **erro** se não houver capacidade/memória para mais elementos;
- **dequeue()** - remove e devolve o elemento que se encontra atualmente no início de Q; a operação deve resultar em *erro* se Q estiver vazia.
- **front()** - devolve, sem remover, o elemento que se encontra atualmente no início de Q; a operação deve resultar em *erro* se Q estiver vazia.

Operações genéricas de coleções:

- **size()** - devolve a contagem do número de elementos atualmente em Q.
- **isEmpty()** - devolve um valor lógico que indica se Q está vazia, ou não.
- **clear()** - descarta todos os elementos presentes em Q voltando ao estado de *vazia*.

Nível 1 - Definição de ADT

- Defina a interface Queue<T> que descreve o comportamento de uma fila na linguagem Java, de acordo com a especificação fornecida, que armazena elementos do tipo T.

- Forneça a documentação *Javadoc* para esta interface, adicionando os comentários de classe/interface e dos métodos; identifique os autores com as *tags* apropriadas.

Nível 2 - Implementação de ADT

- Forneça uma implementação de Queue<T>, baseada em *lista ligada*, na classe QueueLinkedList, usando a abordagen da Figura 1.

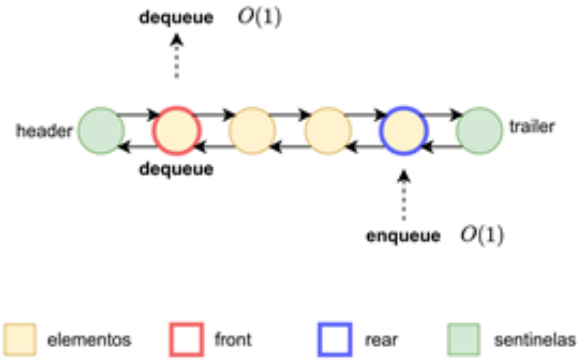

Figure 1 - Implementação do ADT Queue baseada em linked list e possíveis abordagens com respetivas complexidades para as operações principais, com inserção (enqueue) no final da lista e remoção (dequeue) do início da lista

- Adicione os comentários *Javadoc* à classe, detalhando a sua implementação e complexidades algorítmicas das operações, ao construtor da classe, aos seus atributos e classe interna.

Nível 3 - Utilização

- No método **main()** implemente um pequeno programa que ilustre o correto comportamento *FIFO* da implementação efetuada e dos restantes métodos.

Nível 4 - Unit Testing

- Por forma a testar objetivamente implementações de Queue, deverá desenvolver um conjunto de testes unitários para verificar a correta implementação da classe QueueLinkedList, nomeadamente se:

  (Utilize instância(s) de QueueLinkedList<Integer> no desenvolvimento dos testes)

  - O príncipio FIFO é garantido na invocação dos métodos enqueue, dequeue e front;
  - As exceções são corretamente lançadas nos métodos dequeue e front, nas condições previstas.
  - O método size() devolve valores corretos à medida que são adicionados e removidos elementos;
  - O método size() devolve valor correto após invocação do método clear() (existindo elementos).
  - O método isEmpty() devolve valores corretos à medida que são adicionados e removidos elementos;
  - O método isEmpty() devolve valor correto após invocação do método clear() (existindo elementos).

Nível 5 – Nova Implementação e Unit Testing

- Implemente a classe QueueLinkedListNoNulls que é uma classe derivada da classe implementada no Nível 2. Esta implementação garante que não é possível adicionar elementos null à fila; se for o caso deverá ser lançada uma exceção denominada NullNotAllowedException (a criar por si).
- Crie uma nova classe com o método main para ilustrar a correção da sua implementação.
- Crie um conjunto de testes unitários para esta classe, incluindo todos os existentes em QueueLinkedListTest e um adicional que permita verificar o lançamento da exceção aquando da inserção de um elemento null.
  - Questão: como procederia para evitar a "duplicação" de testes?