IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

PA 2021/22

Programação Avançada

Enunciado de Laboratório (avaliado)

2

ADT Priority Queue e Unit Testing

Objetivos:

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

ADT Priority Queue (introdução)

O **ADT Priority Queue** representa uma fila onde os elementos armazenados têm uma *prioridade* associada. Isto significa que ao remover elementos de uma fila com prioridade, os elementos de maior "prioridade" serão removidos primeiro; elementos com a mesma prioridade respeitam o príncipio FIFO.

Na linguagem Java, a forma mais elegante de especificar o critério de prioridade é através do método **compareTo(...)** da *interface* Comparable<T>, pelo que o *ADT Priority Queue* só poderá ser instanciado para conter elementos que implementam esta interface, i.e., public class PriorityQueue<T extends Comparable<T>> implements Queue<T> {...}

Note que a interface Queue mantém-se inalterada e o aluno deverá ter como base a implementação de QueueLinkedList disponibilizada no template.

A diferença na implementação relativamente a uma fila tradicional (quer baseada em *array* ou *lista ligada*) virá da abordagem que adoptar, sendo que as possíveis são:

Abordagem A (privilegia eficiência na inserção):
– Um elemento é inserido na estrutura de dados de forma "normal", e.g., no final; mas ao retirar um elemento terá de devolver o elemento com maior prioridade que se encontra na fila há mais tempo.

Abordagem B (privilegia eficiência na remoção):
– Um elemento é adicionado na estrutura de dados na sua posição de "prioridade correta", i.e., ele terá de ficar à frente de todos os elementos existentes com menos prioridade; ao remover da fila, simplesmente remove-se o elemento que está na frente.

Abordagem C (menos eficiente):
– Um elemento é adicionado na estrutura de dados e os elementos existentes são imediatamente ordenados por prioridade, mas terá de ter em atenção o algoritmo utilizado por forma a respeitar o príncipio FIFO para os elementos com a mesma prioridade; ao remover da fila, simplesmente remove-se o elemento que está na frente.

Se pensar um pouco, em nenhuma destas abordagens irá conseguir obter complexidade simultânea de $O(1)$ para ambas as operações enqueue e dequeue; não existe nenhuma estrutura de dados capaz de permitir tal "proeza" para a fila com prioridade.

Nível 1

a) Execute o IntelliJ e clone o projecto que se encontra no seguinte endereço:
https://github.com/estsetubal-pa-geral/Lab2Template.git

b) Crie uma UT que teste a Classe PriorityQueueLinkedList utilizando apenas elementos do tipo Integer de modo a que:

- Os métodos `size` e `isEmpty` devolvem valores corretos à medida que são adicionados e removidos elementos.

`Nota: Ainda não poderá executar os testes, pois a classe não está completamente implementada.`

Nível 2

Dentro da classe implementada no nível 1, implemente ainda os seguintes testes

- Um teste para testar que os princípios da prioridade/FIFO são garantidos na invocação dos métodos: `enqueue, dequeue e front.`
- Um teste para garantir que a excepção é lançada no método dequeue
- Um teste para garantir que a excepção é lançada no método `front`

`Nota: Ainda não poderá executar os testes, pois a classe não está completamente implementada.`

Nível 3

a) Pretende-se implementar o ADT Priority Queue utilizando a **abordagem B,** para tal deve implementar o método o **enqueue** `da classe PriorityQueueLinked`, de forma que o elemento inserido seja colocado na fila tendo em conta a sua prioridade.

Sugere-se a criação de um método adicional que possa utilizar para saber onde deverá ser colocado o elemento. Utilize a seguinte assinatura:

```
private ListNode nextNodeForElement(E elem)
```

b) Corra os testes realizados no Nivel 1 e 2, para verificar que tudo está correto.

Nível 4

a) Implemente a classe PrintJob com a assinatura `PrintJob implements Comparable <PrintJob>`

contendo os atributos

```
name: String,
priority: Enum{LOW,NORMAL,HIGH}
e numberPages: int.
```

b) Implemente o método `compareTo` devolvendo um valor que espelhe a prioridade relativa mediante o atributo priority.

c) Por forma a testar objetivamente o nível anterior, deverá desenvolver um Main que ilustre a utilização do ADT PriorityQueue tendo como elementos o PriorityJob.

- Crie os seguintes PrintJobs:

```
new PrintJob("Job1", 3, PrintJob.PrintJobPriority.NORMAL),  //jobs[0]
new PrintJob("Job2", 5, PrintJob.PrintJobPriority.NORMAL),  //jobs[1]
new PrintJob("Job3", 2, PrintJob.PrintJobPriority.LOW),     //jobs[2]
new PrintJob("Job4", 7, PrintJob.PrintJobPriority.HIGH),    //jobs[3]
new PrintJob("Job5", 9, PrintJob.PrintJobPriority.NORMAL)   //jobs[4]
```

- Crie uma fila prioritária e adicione os trabalhos pela ordem criada;
- Retire os elementos da lista um a um mostrando a ordem com que saiem,
- Mostre que no fim a lista se encontra vazia.

Nível 5

a) Altere a classe base **PriorityQueueLinkedList** de modo a:

- Remover o atributo `size,` da implementação actual;
- Implementar o método `size(),` de forma a calcular o número de elementos presentes na fila (utilizando um algoritmo iterativo)
- Atualizar o restante código de forma a utilizar o método `size(),` em vez do atributo `size.`

b) Corra novamente os testes realizados no Nivel 1 e 2, para verificar que tudo está correto.