Exercício 4

Tipos genéricos de dados, iteradores e equals

Unidade Curricular de
Laboratório de Programação

2020/2021

Objetivos

- Reforço de conhecimentos sobre classes genéricas;
- Uso de Iteradores em classes Java.

Antes de Começar

Criar um projeto java e importar os ficheiros

- Fazer *download* do ficheiro `alunosExercicio4.zip` na página de LabP.
- No Eclipse:
  - Escolher **File > Import > General > Existing Projects into Workspace**
  - De seguida **Select Archive File** e selecionar `alunosExercicio4.zip`
    Deverá passar a ter um projeto contendo:
    - Na pasta `src`, os ficheiros `Queue.java`, `ArrayQueue.java`, `ArrayQueueSystem.java`, `FileToPrint.java`, `Printer.java` e `RunPrinter.java`;
    - Na pasta `test` o ficheiro `TestsPrinter.java`, com testes Junit para as operações da classe `Printer`;
    - Na raiz do projeto, um ficheiro de dados, `files1.`

Algumas informações úteis

Para realizar este exercício deverá consultar os guiões sobre tipos genéricos de dados e sobre iteradores e a matéria dada em AED sobre filas de espera (*queues*), bem como, sobre o método `equals` da classe Object.
Poderá ainda consultar a documentação da API do java sobre as interfaces `Iterator<E>` (https://docs.oracle.com/javase/9/docs/api/java/util/Iterator.html) e `Iterable<E>` (https://docs.oracle.com/javase/9/docs/api/java/lang/Iterable.html).

Enunciado

Neste exercício vamos considerar a **gestão dos ficheiros que chegam a uma impressora partilhada** por vários utilizadores.

Cada ficheiro tem os seguintes atributos:

- Nome do ficheiro:
- Tamanho (em KB);
- Nome do utilizador que enviou o ficheiro (*owner*).

O sistema de *spooling*, que controla a impressão de ficheiros de uma impressora, pode estar concretizado usando diferentes políticas de impressão e considerando

diferentes prioridades. Neste exercício, usaremos a política de impressão e de atribuição de prioridades que a seguir se descreve. Serão concretizadas apenas algumas funcionalidades básicas do sistema de impressão.

O sistema de *spooling* desta impressora considera 4 prioridades de impressão [0..3], onde:

- A prioridade mais elevada, 0, corresponde aos ficheiros enviados pelo utilizador "admin";
- As prioridades seguintes são para os ficheiros dos restantes utilizadores tendo em atenção o tamanho do ficheiro (em KB):
  - Prioridade 1, se 0 < Tamanho <= 64
  - Prioridade 2, se 64 < Tamanho <= 1024
  - Prioridade 3, se 1024 < Tamanho

Em cada uma das prioridades os ficheiros são impressos por ordem de chegada.

A política de impressão estabelece a seguinte ordem de impressão:

1. todos os ficheiros com prioridade 0;
2. todos os ficheiros com prioridade 1;
3. todos os ficheiros com prioridade 2;
4. todos os ficheiros com prioridade 3.

A impressora tem uma determinada capacidade de memória para armazenar ficheiros em espera. Ao inserir um novo ficheiro em espera para impressão deve ser verificado se há memória suficiente. Se o tamanho do ficheiro exceder a capacidade livre da impressora, o ficheiro não é colocado em espera, gera-se uma mensagem de erro correspondente, e a impressora continua a receber outros pedidos de impressão.

O que fazer então?

Atendendo a que em cada uma das prioridades o tratamento é feito por ordem de chegada, vamos organizar os ficheiros a imprimir num array de filas de espera (*queues*) cada uma correspondente a uma prioridade.
Para este efeito vamos considerar as classes genéricas `ArrayQueue`, que implementa a interface `Queue`, utilizando um array circular, com as operações sobre filas de espera, e `ArrayQueueSystem`, que implementa um array de filas de espera.
A classe `FileToPrint` é utilizada para representar os ficheiros e a classe `Printer` implementa o sistema de gestão da impressora, recorrendo à classe genérica `ArrayQueueSystem`. A classe `RunPrinter` faz a leitura do ficheiro de dados e cria uma instância da classe `Printer` para executar as operações desta classe sobre os dados lidos.
A interface `Queue` e as classes `ArrayQueue` e `RunPrinter` já são fornecidas na totalidade e podem ser consultadas nos respetivos ficheiros. As restantes classes,

`ArrayQueueSystem`, `FileToPrint` e `Printer`, parcialmente já implementadas com os métodos abaixo descritos, devem ser completadas de acordo com as especificações indicadas a seguir. Para além dos métodos indicados, podem ser criados métodos privados considerados adequados.

1) A classe `ArrayQueueSystem<E>` implementa uma fila de espera com prioridades, de elementos do tipo `E`, através de um *array* de filas de espera, cada uma delas correspondente a uma prioridade.

   A classe tem os seguintes atributos:
   - `currentPriorityQueue`, que é o índice da fila onde será inserido o próximo elemento, e
   - `priorityQueue`, que é um array de `ArrayQueue<E>`.

   Nota: como poderá verificar, o tipo dos elementos do *array* `priorityQueue` é declarado como sendo a interface (`Queue<E>`). Considera-se uma boa prática de programação pois facilita a reutilização/alteração do código.

   Esta classe `ArrayQueueSystem<E>`, além de ela própria implementar a interface `Queue<E>`, implementa também a interface `Iterable<E>`.

   O construtor e os métodos já dados têm as seguintes assinaturas:

   - `ArrayQueueSystem (int numberOfQueues)` cria uma fila de espera com prioridades com `numberOfQueues` filas.
   - `void setCurrentPriority (int index)` atualiza o atributo `currentPriority` com o valor `index`.
   - `void enqueue(E elem)` adiciona o elemento `elem` no final da fila cujo índice é `currentPriority`.
   - `void dequeue()` retira de `priorityQueue` o primeiro elemento com maior prioridade.
   - `E front()` devolve o primeiro elemento com maior prioridade em `priorityQueue`.
   - `isEmpty()` verifica se `priorityQueue` está vazia.
   - `int size()` devolve o número de elementos em `priorityQueue`.
   - `String toString()` devolve a representação textual de `priorityQueue`.

   Deve completar esta classe implementando um iterador. Escreva o método `Iterator<E> iterator()`, para implementar a interface `Iterable<E>`, e a classe privada `IteratorQueueSystem<E>`, que concretiza a interface `Iterator<E>`.

2) A classe `FileToPrint` implementa o conceito de ficheiro com os atributos `name` e `owner`, ambos do tipo `String`, respetivamente o nome do ficheiro e o identificador do utilizador que pediu a impressão, e o atributo `size`, do tipo `int`, que guarda o tamanho do ficheiro.

   O construtor e os métodos já dados têm as seguintes assinaturas:

   - `FileToPrint (String name, int size, String owner )` em que `name`, `size` e `owner` vão inicializar os valores dos atributos referidos acima.
   - `String getName()` devolve o nome do ficheiro.
   - `int getSize()` devolve o tamanho do ficheiro.

- `String getOwner()` devolve o identificador do utilizador que mandou imprimir o ficheiro.
- `String toString ()` devolve a representação textual no formato [name, size, owner].

Deve completar esta classe redefinindo o método `equals`, com assinatura `boolean equals (Object other)`, tendo em atenção que duas instâncias desta classe são iguais se tiverem o mesmo nome e tamanho.

3) A classe `Printer` implementa uma impressora com a política de impressão descrita acima.
   Esta classe tem como atributos:
   - `printerQueue`, uma instância de `ArrayQueueSystem<FileToPrint>` que armazena os ficheiros em espera organizados de acordo com a política de impressão;
   - `maxCapacity`, do tipo `int`, que guarda a capacidade de memória da impressora, e
   - `leftCapacity`, do tipo `int`, que guarda a memória da impressora ainda disponível.

   O construtor e os métodos já implementados têm as seguintes assinaturas:
   - `Printer (int maxCapacity)` em que `maxCapacity` corresponde à capacidade de memória desta instância de `Printer`.
   - `int leftCapacity()` devolve a capacidade de memória da impressora ainda disponível.
   - `int size()` devolve o número de ficheiros em espera na impressora.
   - `String toString()` devolve uma representação textual da impressora.

   Deve completar esta classe implementando os seguintes métodos:
   - `int priorityPolicy (FileToPrint f)` que devolve o índice da fila de espera onde deve ser inserido o ficheiro `f` tendo em atenção a política de impressão descrita acima.
   - `boolean add(FileToPrint f)` que, no caso de a impressora ainda ter espaço de memória para aceitar o ficheiro `f`, o insere na fila correta da impressora e devolve `true`. Caso contrário, devolve `false`.
   - `int ocurrenciesOfFile (FileToPrint f)` que devolve o número de ficheiros em espera iguais a `f`, isto é, o número de ficheiros `fp` tais que `f.equals(fp)`.

Ficheiro de dados

O ficheiro de dados contém na primeira linha a capacidade máxima da impressora e em cada uma das linhas seguintes a descrição de um ficheiro.

Exemplo de ficheiro de dados:

```
5500
Prog 50 admin
Prog 50 ana
Alg 50 ana
list 150 admin
```

```
Aq 1 mjc
time 250 afc
chapt.txt 1350 admin
chapt.txt 1500 hrh
```

Para este ficheiro de dados, o resultado da execução de `RunPrinter` será o seguinte:

```
Printer
maxCapacity: 5500
leftCapacity: 2099
Queue 0: <[Prog,50,admin], [list,150,admin], [chapt.txt,1350,admin]<
Queue 1: <[Prog,50,ana], [Alg,50,ana], [Aq,1,mjc]<
Queue 2: <[time,250,afc]<
Queue 3: <[chapt.txt,1500,hrh]<
Size 8
Number of files equal to [Prog,50,admin]: 2
```

O que entregar

Deve entregar os ficheiros `ArrayQueueSystem.java`, `FileToPrint.java` e `Printer.java`.

ATENÇÃO: Antes de submeter o trabalho, verifique que documentou o seu projeto (incluindo `@author` com o seu número de aluno).

Importante

O facto das vossas classes passarem nos testes fornecidos não significa que a classe esteja 100% correta. Há testes e verificações que não são feitas de propósito, de modo a incentivar os alunos a irem à procura de pontos de eventuais falhas no código. Além disso, os pesos para a avaliação dados a cada um dos testes pode ser diferente.