# Projeto 4

Filas


Unidade Curricular de
Laboratório de Programação

2020/2021

# Objetivos

- Uso de Filas

# Antes de Começar

De modo a poder realizar este projeto deverá recordar a estrutura de dados Fila com a filosofia *First in, First out* (FIFO) estudada em AED. Deve ainda rever a interface `List` e a classe `LinkedList` do pacote `java.util`.

Deve descarregar o ficheiro `alunosProjeto4.zip` disponível na página de LabP e, em seguida, no Eclipse escolher `File` → `Import` → `General` → `Existing Projects into Workspace`, para importar esse ficheiro.

Deverá passar a ter um projeto chamado Projeto4 contendo:

- Na pasta `src`, os ficheiros `RunHospital.java` e `UrgencyStatus.java`
- Na pasta `tests`, o ficheiro `TestsProject4.java`, com vários testes `JUnit`
- Dois ficheiros de dados de entrada, a utilizar pela classe `RunHospital`, e o ficheiro de saída que será obtido a partir desses dados (para poderem comparar com o vosso resultado).

# Enunciado

No hospital `Saúde Para Todos` há uma equipa de médicos que lida com os vários doentes que vão chegando ao serviço de urgências. A equipa de médicos é formada por um dado número de elementos que resolve todos os casos.

Para cada paciente, o serviço de urgências faz o seu registo no sistema (sendo-lhe atribuído então um nº de ordem de chegada) e, de seguida, atribui o paciente a um dos médicos que prestam assistência nesse dia (ver critério de atribuição mais à frente). Todos os médicos atendem os pacientes com prioridade urgente antes dos com prioridade não urgente; pacientes com a mesma prioridade são atendidos pelo número de ordem de chegada.

Cada paciente tem um número (o seu número de ordem de chegada ao hospital), um valor que indica a urgência que lhe foi atribuída na triagem e o tempo que será necessário para o tratar.

É dado um ficheiro com a informação dos pacientes que chegam ao hospital. Este ficheiro descreve um paciente por linha, onde cada linha contém o valor de urgência do paciente e o respetivo tempo de tratamento. Vamos considerar apenas dois graus de urgência, `URGENT` e `NONURGENT`, valores do tipo enumerado `UrgencyStatus`.

Pretende-se que seja criada uma "distribuição de serviço" automatizada, o mais equitativa possível para os médicos. Uma forma de se conseguir isso é atribuir cada paciente ao médico com menor tempo de espera (ou seja, aquele que levará o menor tempo para atender todos os pacientes que já lhe foram atribuídos e que ainda não atendeu). Caso haja vários com o mesmo tempo de espera, o paciente será atribuído ao que tiver o menor número de pacientes e, ainda em caso de empate, àquele que tiver menos doentes urgentes por atender.

Ao longo do tempo vão-se registando as alterações feitas à distribuição de serviço. Por exemplo, o registo de novos pacientes que vão chegando e dos pacientes que vão tendo alta. Neste projeto pretende-se elaborar um programa que seja capaz de operar e registar as várias alterações que vão sendo feitas à medida que o serviço decorre.

# O que fazer

Será necessário implementar pelo menos as seguintes classes:

- A classe `Patient`, cujas instâncias representam pacientes. Cada paciente tem um identificador (o número de ordem de entrada no serviço de urgência), a urgência atribuída na triagem (um valor do tipo enumerado `UrgencyStatus`) e o respetivo tempo esperado de tratamento. Crie métodos que auxiliem a manipulação dos objetos do tipo `Patient`, incluindo o método `toString()`.

- A classe `Doctor`, cujas instâncias representam médicos. Cada médico deverá ter, entre outra informação que ache necessária: a identificação do médico (o seu número no serviço de urgência) e a fila (ou filas) dos seus pacientes, que guarda os vários pacientes que foram atribuídos a esse médico. Assume-se que a identificação dos médicos é única. As filas de pacientes (tantas quantas os níveis possíveis de urgência) devem ser implementadas usando a classe `LinkedList` do pacote `java.util`. Veja mais abaixo os métodos fundamentais que a classe deve oferecer.

- A classe `EmergencyService` que implementa a distribuição de serviço automatizada do serviço de urgências do hospital pelos médicos de serviço.

A classe `EmergencyService` deve ter pelo menos:

- O construtor `EmergencyService(int m)` que cria um serviço de urgência com `m` médicos, sendo `m` um inteiro positivo. Os médicos ficam identificados pelos números de `1` a `m`.
- O método `boolean hasDoctor(int doctorID)` que verifica se existe um médico com esse identificador no serviço de urgência.
- O método `int checkIn (UrgencyStatus urgency, int consultationTime)` que regista a entrada de um novo paciente no serviço de urgência, dados o seu grau de urgência e tempo de tratamento, e devolve o identificador (número de ordem) desse paciente.
- O método `boolean hasCheckedIn(int patientID)` que verifica se já foi registada a entrada do paciente com esse identificador no serviço de urgência.
- O método `int chooseDoctor()` que determina o médico a quem será atribuído o próximo paciente, de acordo com o critério descrito acima, devolvendo a identificação desse médico.
- O método `void assignPatient(int doctorID, int patientID)` que adiciona o paciente à lista de pacientes por atender desse médico, consoante a urgência do paciente. Assume-se que `hasDoctor(doctorID)` e `hasCheckedIn(patientID)`.
- O método `int getNumberOfDoctors()` que devolve o número de médicos no serviço de urgência.
- O método `int totalPatients()` que devolve o número de pacientes que permanecem no serviço de urgência, isto é, que já entraram e ainda não tiveram alta.
- O método `List<int> attendanceTimeAbove(int bound)` que devolve uma lista com a identificação dos médicos cujo tempo total para atender todos os doentes a eles atribuídos é superior a `bound`, tendo em conta a(s) sua(s) fila(s) de espera de pacientes.

- O método `int minWaitingTime()` que devolve o tempo mínimo de espera que um novo paciente que chegue ao serviço levará a começar a ser atendido (o tempo de atendimento do médico com o menor tempo de atendimento).
- O método `int dischargePatient()` que determina o paciente com menor tempo de tratamento que está em primeiro lugar para ser atendido por um dos médicos, retira esse paciente da fila desse médico e devolve o número de ordem desse paciente. Caso haja vários pacientes possíveis, deve retirar e retornar aquele com menor número de ordem. Assume-se que `totalPatients() > 0`.
- O método `String toString()` que retorna uma `string` representando, para cada um dos médicos de serviço, a lista de todos os pacientes que ainda tem para atender, por ordem de chegada ao hospital, indicando para cada um: o nº de ordem de chegada, o código de urgência e o tempo de atendimento.

A classe `Doctor` deve ter no mínimo os seguintes métodos:

- O método `int getID()` que devolve o número de identificação do médico.
- O método `int attendanceTime()` que devolve o tempo total necessário para atender todos os doentes atribuídos a esse médico.
- O método `boolean isFree()` que indica se o médico não tem nenhum doente em espera.
- O método `void addPatient(int patientID, UrgencyStatus urgency, int consultationTime)` que adiciona o paciente com as características indicadas à lista de espera do médico, consoante o seu grau de urgência.
- O método `int nextToAttend()` que devolve o identificador do primeiro doente que o médico irá atender de seguida (mas não o retira da lista de espera desse médico), assumindo `!isFree()`.
- O método `int dischargePatient()` que retira da lista de espera do médico o primeiro doente que ele irá atender em seguida e devolve o seu identificador, assumindo `!isFree()`.

- O método `int numberPatients(UrgencyStatus urgency)` que retorna o número de pacientes, com a dada urgência, que o médico tem em espera.
- O método `String forAttendance()` que retorna uma String com a lista dos pacientes em fila de espera para o médico, indicados por ordem de chegada ao hospital, indicando para cada um também a sua urgência e tempo de atendimento. Para cada paciente, a String devolvida contém `"Patient " + numeroDoPaciente + ": " + urgenciaDoPaciente + " " + tempoConsultaDoPaciente + "; "` (esta String deverá corresponder ao resultado do método `toString()` do paciente).

# A classe `RunHospital`

É dada uma classe `RunHospital`, já implementada, que controla o desencadear da distribuição de serviço de urgência: cria o serviço de urgência e vai enviando a esse serviço os pacientes que chegam ao hospital e dando alta aos pacientes cujo atendimento já terminou.

Existem 2 ficheiros de *input*:

- `patients.txt` que contém a lista dos pacientes (um por linha, com uma *string* descritiva da urgência e um inteiro representando o tempo de tratamento) que poderão vir a dar entrada no serviço de urgência.
- `events.txt` que descreve a sequência de eventos que terá lugar no hospital ao longo do tempo: um evento pode corresponder, por exemplo, à chegada do próximo paciente, à saída de um paciente ou a um pedido de informação sobre o estado do serviço de urgências. Os eventos possíveis, e seu significado, são:
  - `nextArrival:` chegada ao serviço do próximo paciente, caso exista algum, seu registo no serviço de urgência e atribuição a um dos médicos de serviço
  - `nextDeparture:` saída do paciente seguinte a ter alta do serviço de urgência, caso exista algum paciente no serviço

e os pedidos de informação sobre o estado do serviço de urgência que são os seguintes:

i. `howManyDoctors`: indicação do número de médicos em serviço,
ii. `forAttendance`: indicação, para cada um dos médicos, dos pacientes em fila de espera para esse médico, indicados por ordem de chegada ao hospital, indicando para cada um também a sua urgência e tempo de atendimento
iii. `totalPatients`: indicação do número de doentes que estão no serviço de urgência
iv. `minWaitingTime`: indicação do tempo mínimo de espera de um próximo doente que chegue para ser atendido
v. `attendanceTimeAbove m`: indicação da lista de médicos cujo tempo de atendimento é superior a `m` minutos

Os ficheiros `patients.txt` e `events.txt` são lidos na classe `RunHospital`.

Para cada um dos eventos indicados no ficheiro `events.txt`, é escrita, pelo `RunHospital`, no ficheiro de saída `logEmergencyService.txt` uma mensagem correspondente.

# O que entregar

Deve criar o ficheiro `P4fcxxxxx.zip`, onde `xxxxx` é o seu número de aluno, contendo os ficheiros:

- `Patient.java`, `Doctor.java` e `EmergencyService.java`
- e ainda um diretório `Docs` contendo todos os ficheiros gerados pelo `Javadoc`, com a documentação de todas as classes do projeto.

Note que **não tem de entregar quaisquer ficheiros de dados** que possa ter usado.