Programação Orientada por Objetos

2020/2021

2º Projeto

“Base de dados hospitalar de doenças infectocontagiosas”

Docentes:

- Sergi Bérmudez
- Luís Ferreira
- Bernardo Gouveia

Membros do grupo:

- Hugo Rocha nº 2046019
- Sérgio Oliveira nº 2046719
- Marina Spínola nº 2049519

Índice

Introdução

Neste projeto foi-nos pedido que realizássemos uma aplicação que permitisse gerir a base de dados de uma unidade hospitalar de doenças infectocontagiosas, permitindo também a inserção e consulta de informação relevante, aplicando os conteúdos que nos foram ensinados nas aulas.

Assim sendo é possível observar no nosso projeto funcionalidades como:

1. Menu Médico:

- Listar pacientes em espera no hospital.
- Listar pacientes a aguardar alta.
- Diagnóstico ao paciente.
- Dar alta hospitalar.
- Requerimento de auxiliares.

2. Menu Enfermeiro:

- Listar enfermeiros de médico.
- Listar pacientes a aguardar curativo.
- Atribuir enfermeiro-especialista a médico.
- Aplicar curativo a paciente.
  (Aplicadas adicionalmente por nós)
- Listar requisitos de enfermeiros auxiliares
- Atender aos requisitos de enfermeiros auxiliares
- Diagnóstico ao paciente
- Listar pacientes à espera de diagnóstico

3. Menu Administrador:

- Criar médico.
- Criar enfermeiro-especialista.
- Criar enfermeiro-auxiliar.
- Criar novo paciente.
- Promover enfermeiro a chefe.
- Aumentar anos de carreira de todos os enfermeiros.
- Listar enfermeiros.
- Listar médicos.
- Listar pedidos para enfermeiros-auxiliares.
- Listar pacientes em espera no hospital.
- Atirar pedidos para enfermeiros-auxiliares para a trituradora.
- Atende ao pedido para enfermeiros-auxiliares.
- Vírus outbreak.
- Relatório hospitalar.
- Sair da aplicação.

Desenvolvimento

Neste tópico iremos falar sobre a nossa abordagem face ao desenvolvimento desta aplicação.

Com o desenvolvimento do projeto foram feitos 4 packages, cada uma contendo classes diferentes, sendo Package Hospital-Database a principal.

Consideramos ***overriding*, quando o redefinimos o comportamento de um método herdado por uma superclasse ou interface**, que contem mesmo nome variando apenas no comportamento.

Package Hospital-Database

Classe Files

Nesta classe é feito o "populate" do hospital. Isto é possível através da leitura de ficheiros. Para tal ser possível seguimos esta nomenclatura na leitura do ficheiro:

Classe, ID, Nome, Ano de Nascimento, anos de carreira (no caso do dos enfermeiros)

Com base nisso, associamos cada linha do ficheiro a um array que contém a informação da mesma. Este leva split de modo a podermos aceder à informação em cada índice, para que possamos a criar cada enfermeiro, médico ou paciente.

De seguida, verificamos a respetiva classe de modo a aceder aos índices que interessam para cada pessoa.

Classe Menu

Nesta classe contém todos os métodos relacionados à interface de interação com os menus de interação com o utilizador.

Classe Main

Aqui ocorre a inicialização do programa, sendo dado nome do ficheiro responsável por dar "populate".

Classe Remedy

Decidimos implementar uma classe respetivamente aos curativos, de modo a deixar simples e de fácil de acesso. Esta contém o nome do curativo a ser aplicado, relativamente à doença em questão, como também a data que foi aplicado. Assim como contém os métodos seletores e modificadores ("*getters e setters*") de cada atributo.

Classe Hospital

Visto que o enunciado apenas fala de um hospital decidimos implementar um **singleton (classe que só pode ter um único objecto)** permitindo criação de outros hospitais caso necessário.

Métodos incluídos:

Populate()

Método responsável por adicionar as pessoas ao hospital, provenientes do ficheiro.

addMedic()

Método responsável por criar e adicionar um novo médico ao hospital.

addSpecialistNurse()

Método responsável por criar e adicionar um novo enfermeiro especialista ao hospital.

addAuxiliaryNurse()

Método responsável por criar e adicionar um enfermeiro auxiliar ao hospital.

addNewPacient()

Método responsável por adicionar um novo paciente ao hospital.

promoteSpecialistNurseToChief()

Método reponsável por promover um enfermeiro especialista a enfermeiro chefe, verificando os seus anos de anos.

progressNursesCareerYears()

Método responsável por adicionar a todos os enfermeiros do hospital mais 1 ano de carreira.

listNurses()

Método responsável por demonstrar todos os enfermeiros presentes no hospital.

listMedics()

Método responsável por demonstrar os medicos presentes no hospital.

listRequestsForAuxiliaryNurses()

Método responsável por demonstrar todos o pedidos de enfermeiros auxiliares enviados para o hospital, caso estes não possam ser aceites por enfermeiro chefe.

trashRequestsForAuxiliaryNurses()

Método responsável por descartar um número aleatório de pedidos de enfermeiros auxiliares que estejam no hospital.

virusOutbreak()

Método responsável por infetar todos os funcionários do hospital com uma percentagem de 10%.

hospitalReports()

Método responsável por gerar os relatórios hospitalares. Em grupo decidimos demonstrar neste método o total de pacientes recebidos no hospital, o número de altas, o número de mortes após a aplicação do curativo e o total de curativos aplicados.

listPacientsInHospitalQueue()

Este método é responsável por demonstrar os pacientes que estão na fila de espera do hospital.

fulfilMedicAuxiliaryRequest()

Método responsável por ceder ao pedido de enfermeiros auxiliares por parte do médico desde que hajam enfermeiros auxiliares suficientes. Caso estes não estejam ainda associados ao médico, o método associa.

Package Hospital-Database.Person

Interface Chief Nurse

Nesta interface são definidos os métodos que levaram ***@override*** quando forem implementados noutras classes.

Decidimos implementar enfermeiro chefe como interface pois um enfermeiro chefe não é nada mais do que um enfermeiro especialista com comportamentos adicionais.

Interface Infectable

Esta interface contém um método que com o ***@override*** permitirá gerar os sintomas da pessoa. É somente implementada na classe Person.

Classe SpecialistNurse

Nesta classe encontram-se implementadas funções que levaram ***@override*** da interface ChiefNurse.

Métodos incluídos:

ListMedicAuxiliaryRequests()

Dá print de todos os pedidos de auxiliaries por parte dos medicos. Este método só pode ser chamado por um enfermeiro chefe.

fulfilMedicAuxiliaryRequest()

Este método cede os enfermeiros auxiliares que o médico pediu. Tal como o outro método este também só pode ser chamado por um enfermeiro chefe.

attributeSpecialistNurseToMedic

Método responsável por atribuir um enfermeiro especialista a um médico. Tal como os outros métodos, este também só pode ser chamado por um enfermeiro chefe.

Classe Nurse

Nesta classe encontramos os métodos modificadores e seletores ("*getters e setters*") relativamente aos atributos do enfermeiro, como também o ***@override*** dos métodos ***equals*** e ***toString***.

Dentro desta classe, estão incluídos métodos que dizem respeito às funcionalidades de um enfermeiro.

Métodos incluídos:

helpsDiagnostic()-

Usado para conciliar o diagnóstico a um paciente. Caso seja verificado algum dos sintomas, é adicionado à agenda de modo a que seja aplicados os curativos. Caso o paciente não conteenha nenhum sintoma é removido do hospital sendo assim dado alta, entrando assim para os registos do hospital

applyCureToPacient()-

Usado apra aplicar o curativo no primeiro paciente à espera no agenda do enfermeiro. Como é pedido no enunciado, existe dentro deste método uma percentagem de que ao aplicação do curativo corra mal e o paciente morra, entrando assim para o registo do hospital.

listPacientsWaitingForCure()-

Usado para dar print do ***Set atual*** de pacientes a aguardar pelo curativo.

listPacientsWaitingForDiagnostic()-

Usado para dar print da ***Queue atual*** de pacientes a aguardar pelo diagnóstico.

listMedicNurses()-

Usado listar todos os enfermeiros associados aos médicos.

Classe Person

Esta classe é essencial neste projeto, pois tratamos todos os elementos do hospital (médicos, enfermeiros, pacientes), como pessoas.

Aqui encontram-se os métodos seletores e modificadores ("*getters e setters*") dos respetivos atributos de uma pessoa, juntamente com ***@override*** dos métodos ***equals*** e ***toString***(dá a informação da respetiva pessoa). Contém também o método **infect()**, que permite gerar os sintomas da própria pessoa.

Classe Medic

Nesta classe temos os métodos seletores e modificadores relativamente aos médicos, pois estes têm como **atributo** as **listas** de **enfermeiros** (auxiliares, especialistas) e de **pacientes**.

Estão incluídos o ***@override*** dos métodos ***toString*** e ***equals***.

Métodos incluídos

listPacientsInHospitalQueue()

Esta método serve para listar todos os pacientes na fila do de espera hospital. O método em questão também é utilizado no menu do administrador.

listPacientsAwaitingDischarge()

Este método serve para demonstrar todos os pacientes que estão à espera de que o médico dê-lhes.

pacientDiagnostic()

Este é o método principal por fazer o diagnóstico aos pacientes. Só funciona desde que o médico tenha no máximo 3 pacientes. Também são gerados os sintomas que decidem quais os curativos a ser aplicados no paciente.

dischargePacient()

Neste método o paciente é enviado para um enfermeiro espcialista, que verifica se o paciente continua a ter os sintomas, para depoois receber alta.

requestAuxiliaryNurses

Neste método é enviado um pedido a pedir enfermeiros auxiliares, feito pelo médico, para o enfermeiro chefe. Caso não hajam enfermeiros auxiliares suficientes para completer o pedido, este é enviado para o hospital. Se houver enfermeiros auxiliares, o pedido é enviado para o enfermeiro chefe.

Package Hospital-Database.Exceptions

Nesta package todos os métodos relativamente às exceções, dão ***output*** de uma **mensagem correspondente à exceção**.

Classe NoPacientsToDiagnoseException

Aqui é feita uma exceção quando não existem pacientes para o médico diagnosticar.

Classe NotEnoughCareerYearsException

Aqui é feita uma exceção para quando o enfermeiro-especialista a promover não contém anos de experiência.

Classe MaximumCapacityFilled

Aqui é feita uma exceção para o caso de o médico já ter esgota do número de pacientes a que está associado.

Classe NoMedicRequestsExistException

Aqui é feita uma exceção caso não hajam pedidos feitos pelos médicos relativamente ao requisito de enfermeiros auxiliares.

Classe NoMedicsExistException

Aqui é feita uma exceção caso não haja nenhum médico no hospital.

Classe NoNursesExistException

Aqui é feita uma exceção caso não haja nenhum enfermeiro no hospital.

Classe NoPacientsAwaitingCureException

Aqui é feita uma exceção caso não haja nenhum paciente à espera de levar com o curativo.

Classe NoPacientsAwaitingDischargeException

Aqui é feita uma exceção caso não haja nenhum paciente à espera de levar alta hospitalar.

Classe NoPacientsInWaitingQueueException

Aqui é feita uma exceção caso não haja nenhum paciente na fila de espera do hospital.

Classe NotEnoughAuxiliaryNursesException

Aqui é feita uma exceção caso o pedido feito pelo médico exceda o número de enfermeiros auxiliares disponível.

Classe NotEnoughPermissionsException

Aqui é feita uma exceção caso um enfermeiro tente fazer alguma função que seja de um enfermeiro chefe.

Classe IDNotFoundException

Aqui é feita uma exceção quando o ID inserido não pertence a nenhuma das listas que contêm os ID´s.

Classe NoPacientsToDiagnoseException

Aqui é feita uma exceção caso não haja nenhuma paciente para realizar o diagnóstico.

Classe NoPacientsWaitingForDiagnosticException

Aqui é feita uma exceção quando não existem pacientes à espera pelo diagnóstico.

Classe NoSpecialistNursesAttributedToMedicException

Aqui é feita uma exceção caso não haja nenhum enfermeiro especialista para atribuir

Classe NurseAlreadyAttributedToMedicException

Aqui é feita uma exceção caso o enfermeiro (especialista ou auxiliar) já esteja associado a um médico.

Package Hospital-Database.UserInterface

Classe AwaitsUserInput

Esta classe é uma classe de utilidade, usada para esperar que o utilizador insira alguma informação.

Classe ClearConsole

Esta classe é uma classe de utilidade, feita sobretudo para nos ajudar a fazer *debug* e verificar os outputs que estaríamos a receber dos métodos, como também para limpar a consola durante a execução do programa.

Conclusão

Com este projeto conseguimos aplicar bem os conceitos e fundamentos do paradigma de programação orientada por objetos, em contexto da linguagem JAVA, permitindo assim colocar em prática os conceitos abordados nas aulas.

Anexos

```
package Hospital_Database.Exceptions;


public class IDNotFoundException extends Exception {


   /**
    *
    */
   private static final long serialVersionUID = 1L;


   public IDNotFoundException(String message) {
      super(message);
   }
}
package Hospital_Database.Exceptions;


public class MaximumCapacityFilled extends Exception {
```

```
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public MaximumCapacityFilled(String message) {
        super(message);
    }

}
package Hospital_Database.Exceptions;

public class NoMedicRequestsExistException extends Exception {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public NoMedicRequestsExistException(String message) {
        super(message);
    }

}
package Hospital_Database.Exceptions;

public class NoMedicsExistException extends Exception {

    /**
     *
```

```
    */
    private static final long serialVersionUID = 1L;

    public NoMedicsExistException(String message) {
        super(message);

    }

}
package Hospital_Database.Exceptions;

public class NoNursesExistException extends Exception {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public NoNursesExistException(String message) {
        super(message);

    }

}
package Hospital_Database.Exceptions;

public class NoPacientsAwaitingCureException extends Exception {

    /**
     *
     */
```

```
    private static final long serialVersionUID = 1L;

    public NoPacientsAwaitingCureException(String message) {
        super(message);
    }
}
package Hospital_Database.Exceptions;

public class NoPacientsAwaitingDischargeException extends Exception {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public NoPacientsAwaitingDischargeException(String message) {
        super(message);
    }

}
package Hospital_Database.Exceptions;

public class NoPacientsInWaitingQueueException extends Exception {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public NoPacientsInWaitingQueueException(String message) {
        super(message);
```

```
    }

}
package Hospital_Database.Exceptions;

public class NoPacientsToDiagnoseException extends Exception {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public NoPacientsToDiagnoseException(String message) {
        super(message);
    }

}
package Hospital_Database.Exceptions;

public class NoPacientsWaitingForDiagnosticException extends Exception {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public NoPacientsWaitingForDiagnosticException(String message) {
        super(message);
    }

}
```

```
package Hospital_Database.Exceptions;

public class NoSpecialistNursesAttributedToMedicException extends Exception {

   /**
    *
    */
   private static final long serialVersionUID = 1L;

   public NoSpecialistNursesAttributedToMedicException(String message) {
      super(message);
   }
}
package Hospital_Database.Exceptions;

public class NotEnoughAuxiliaryNursesException extends Exception {

   /**
    *
    */
   private static final long serialVersionUID = 1L;

   public NotEnoughAuxiliaryNursesException(String message) {
      super(message);
   }

}
package Hospital_Database.Exceptions;

public class NotEnoughCareerYearsException extends Exception {
```

```
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public NotEnoughCareerYearsException(String message) {
        super(message);
    }

}
package Hospital_Database.Exceptions;

public class NotEnoughPermissionsException extends Exception {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public NotEnoughPermissionsException(String message) {
        super(message);
    }

}
package Hospital_Database.Exceptions;

public class NurseAlreadyAttributedToMedicException extends Exception {

    /**
     *
     */
```

```
    private static final long serialVersionUID = 1L;

    public NurseAlreadyAttributedToMedicException(String message) {
        super(message);
    }
}
package Hospital_Database.Person;

public class AuxiliaryNurse extends Nurse {

    public AuxiliaryNurse(int ID, String name, int birthdayYear, int careerYears) {

        super(ID, name, birthdayYear, careerYears);
    }

    @Override
    public String toString() {
        return super.toString();
    }

    @Override
    public boolean equals(Object object) {
        return super.equals(object);
    }

}
package Hospital_Database.Person;

import java.util.HashMap;

import Hospital_Database.Hospital;
```

```
import Hospital_Database.Exceptions.IDNotFoundException;
import Hospital_Database.Exceptions.NoMedicRequestsExistException;

public interface ChiefNurse {

   public abstract void attributeSpecialistNurseToMedic(Hospital hospital) throws
IDNotFoundException;

   public abstract HashMap<Medic, Integer> getMedicAuxiliaryRequests();

   public abstract void listMedicAuxiliaryRequests() throws NoMedicRequestsExistException;

   public abstract void fulfilMedicAuxiliaryRequest(Hospital hospital) throws
NoMedicRequestsExistException;

}
package Hospital_Database.Person;

public interface Infectable {

   public abstract void infect();

} package Hospital_Database.Person;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

import Hospital_Database.Hospital;
import Hospital_Database.Menu;
import Hospital_Database.Exceptions.IDNotFoundException;
import Hospital_Database.Exceptions.MaximumCapacityFilled;
```

```
import Hospital_Database.Exceptions.NoPacientsAwaitingDischargeException;
import Hospital_Database.Exceptions.NoPacientsInWaitingQueueException;
import Hospital_Database.Exceptions.NoPacientsToDiagnoseException;
import Hospital_Database.Exceptions.NoPacientsWaitingForDiagnosticException;
import Hospital_Database.Exceptions.NoSpecialistNursesAttributedToMedicException;
import Hospital_Database.Exceptions.NotEnoughAuxiliaryNursesException;
import Hospital_Database.UserInterface.AwaitsUserInput;
import Hospital_Database.UserInterface.ClearConsole;

public class Medic extends Person {

   // ! Instance variables
   private ArrayList<SpecialistNurse> specialistNurses;
   private ArrayList<AuxiliaryNurse> auxiliaryNurses;
   private Queue<Person> pacientsAwaitingDiagnotic;
   private Queue<Person> pacientsAwaitingDischarge;

   // ! Constructor
   public Medic(int ID, String name, int birthdayYear) {
      super(ID, name, birthdayYear);
      specialistNurses = new ArrayList<>();
      auxiliaryNurses = new ArrayList<>();
      pacientsAwaitingDischarge = new LinkedList<>();
      pacientsAwaitingDiagnotic = new LinkedList<>();
   }

   // ! Getters
   public Queue<Person> getPacientsAwaitingDiagnotic() {
      return pacientsAwaitingDiagnotic;
   }
```

```
    public ArrayList<AuxiliaryNurse> getAuxiliaryNurses() {
        return auxiliaryNurses;
    }

    public ArrayList<SpecialistNurse> getSpecialistNurses() {
        return specialistNurses;
    }

    public Queue<Person> getPacientsAwaitingDischarge() {
        return pacientsAwaitingDischarge;
    }

    // ! Overriden methods
    @Override
    public String toString() {

        // Adds all associated auxilary nurses IDs to a array list
        ArrayList<Integer> auxiliaryNursesIDs = new ArrayList<>();
        for (AuxiliaryNurse tempAuxiliaryNurse : auxiliaryNurses) {
            auxiliaryNursesIDs.add(tempAuxiliaryNurse.getID());
        }

        // Adds all associated specialist nurses IDs to a array list
        ArrayList<Integer> specialistNursesIDs = new ArrayList<>();
        for (SpecialistNurse tempSpecialistNurse : specialistNurses) {
            specialistNursesIDs.add(tempSpecialistNurse.getID());
        }

        return "ID: " + getID() + "\n" + "Name: " + getName() + "\n" + "Auxiliary Nurses: "
                + auxiliaryNursesIDs.toString() + "\n" + "Specialist Nurses: " +
specialistNursesIDs.toString();
```

```
    }

    @Override
    public boolean equals(Object object) {
        if (this == object) {
            return true;
        }

        if (object == null) {
            return false;
        }

        if (this.getClass() != object.getClass()) {
            return false;
        }

        Medic medic = (Medic) object;
        return super.getID() == medic.getID();
    }

    // ! User interface related methods
    public void listPacientsWaitingForDiagnostic() throws
NoPacientsWaitingForDiagnosticException {
        // Lists all pacients waiting for discharge

        ClearConsole.clearConsole();

        // If there are no pacients waiting for diagnostic, throw exception
        if (pacientsAwaitingDiagnotic.size() == 0) {
            throw new NoPacientsWaitingForDiagnosticException("Não existem pacientes à espera
de diagnóstico.");
```

```
    } // If there are pacients waiting for diagnostic, print them to the console
    else {

      System.out.println("Pacientes à espera de diagnóstico\n");

      for (Person pacient : pacientsAwaitingDiagnotic) {
        System.out.println(pacient.toString());
      }
    }

    // Waits for user input
    AwaitsUserInput.awaitsUserInput();

  }

  public void listPacientsInHospitalQueue(Hospital hospital) throws
NoPacientsInWaitingQueueException {
    // * Also used in the administrator menu

    ClearConsole.clearConsole();

    // If there aren't any pacients in the hospital waiting queue, throw exception
    if (hospital.getPacientQueue().size() == 0) {
      throw new NoPacientsInWaitingQueueException("Não existem pacientes na lista de
espera.");
    } // If there are pacients in the hospital waiting queue, print them to the
    // console
    else {

      // Prints pacients information to console
      System.out.println("Pacientes na lista de espera do hospital\n");
      for (Person pacient : hospital.getPacientQueue()) {
```

```
            System.out.println(pacient.toString() + "\n");
        }

    }

    // Waits for user input
    AwaitsUserInput.awaitsUserInput();

  }

  public void listPacientsAwaitingDischarge() throws NoPacientsAwaitingDischargeException {
    // Lists all pacients waiting for discharge

    ClearConsole.clearConsole();

    // If the medic doesn't have pacients awaiting discharge
    if (pacientsAwaitingDischarge.size() == 0) {
      throw new NoPacientsAwaitingDischargeException("O médico não tem pacientes a
aguardar alta.");
    } // If the medic has pacients awaiting discharge
    else {

      System.out.println("Pacientes à espera de alta\n");

      // Prints all pacients waiting for discharge
      for (Person pacient : pacientsAwaitingDischarge) {

        System.out.println(pacient.toString() + "\n");

      }
```

```
        }

        // Waits for user input
        AwaitsUserInput.awaitsUserInput();
    }

    public void pacientDiagnostic(Hospital hospital)
            throws NoPacientsToDiagnoseException,
NoSpecialistNursesAttributedToMedicException, MaximumCapacityFilled {
        // Starts the pacient's diagnostic process. The process needs a specialist
        // nurse, but its up to the medic to decide if there is a need to request
        // auxiliary nurses beforehand

        // If the medic doesn't have a specialist nurse attributed
        if (specialistNurses.size() == 0) {
            throw new NoSpecialistNursesAttributedToMedicException(
                    "Não tem enfermeiros especialistas atribuídos para começar o diagnóstico.");
        }

        // If the medic already has 3 pacients associated
        if (pacientsAwaitingDiagnotic.size() == 3) {
            throw new MaximumCapacityFilled("O médico já tem 3 pacientes.");
        }

        pacientsAwaitingDiagnotic.add(hospital.getPacientQueue().poll());
        Person currentPacient = pacientsAwaitingDiagnotic.poll();

        // If there are no pacients to diagnose, throw an exception
        if (currentPacient == null) {
            throw new NoPacientsToDiagnoseException("Não há pacientes por diagnosticar.");
        } // If there are pacients in the hospital waiting queue, diagnose the first
```

```
        else {

            // Generate the person symptoms
            currentPacient.infect();

            // Sends the pacient to a specialist nurse, to help with the diagnostic
            specialistNurses.get(0).getPacientsWaitingForDiagnostic().add(currentPacient);

            ClearConsole.clearConsole();
            System.out.println(
                    "Paciente enviado para diagosticar ao enfermeiro " + specialistNurses.get(0).getID()
+ ".");
            Menu.scanner.next();
        }

    }

    public void dischargePacient() throws NoPacientsAwaitingDischargeException {
        // Sends the pacient to a specialist nurse, to verify if the pacient still has
        // any symptoms

        // If there are no pacients awaiting discharge, throw an exception
        if (pacientsAwaitingDischarge.size() == 0) {
            throw new NoPacientsAwaitingDischargeException("Não existem pacientes à espera de
alta.");
        } // If there are pacients awaiting discharge, send the first to a specialist
        // nurse, for a new diagnostic
        else {

            Person currentPacient = pacientsAwaitingDischarge.poll();

            // Generate the person symptoms
```

```
        currentPacient.infect();

        // Sends the pacient to a specialist nurse
        specialistNurses.get(0).getPacientsWaitingForDiagnostic().add(currentPacient);
        ClearConsole.clearConsole();
        System.out.println("Paciente enviado de volta para diagosticar ao enfermeiro "
                + specialistNurses.get(0).getID() + ", para que possa dar alta.");
        Menu.scanner.next();

    }

}

public void requestAuxiliaryNurses(Hospital hospital)
        throws IDNotFoundException, NotEnoughAuxiliaryNursesException {
    // Sends a request for auxiliary nurses to a chief nurse

    ClearConsole.clearConsole();

    // Input the chief nurse to send the request to
    System.out.println("ID do chefe enfermeiro: ");
    int chiefNurseID = Menu.scanner.nextInt();

    // Check if the chief nurse exists
    SpecialistNurse chiefNurse = null;
    for (SpecialistNurse tempChiefNurse : hospital.getChiefNurses()) {
        if (tempChiefNurse.getID() == chiefNurseID) {
            chiefNurse = tempChiefNurse;
        }
    }
```

```
        // If the chief nurse doesn't exist, throw an exception
        if (chiefNurse == null) {
            throw new IDNotFoundException("Não existe nenhum enfermeiro chefe com o ID
inserido.");
        } // If the chief nurse exists, send the request for auxiliary nurses
        else {
            ClearConsole.clearConsole();
            System.out.println("Quantos enfermeiros auxiliares necessita: ");
            int auxiliaryNursesRequested = Menu.scanner.nextInt();

            // If there are not enough free auxiliary nurses to complete the request, throw
            // an
            // exception
            int numberOfFreeAuxiliaries = 0;

            // Counts the number of auxiliary nurses with no medics associated
            for (AuxiliaryNurse tempAuxiliaryNurse : hospital.getAuxiliaryNurses()) {
                if (tempAuxiliaryNurse.getAssociatedMedic() == null) {
                    numberOfFreeAuxiliaries++;
                }
            }

            // If there are not enough auxiliary nurses to complete the request, send the
            // request to the hospital
            if (auxiliaryNursesRequested > numberOfFreeAuxiliaries) {
                hospital.getAuxiliaryRequests().put(this, auxiliaryNursesRequested);
                throw new NotEnoughAuxiliaryNursesException(
                        "Não existem enfermeiros auxiliares suficientes. O pedido foi enviado para o
hospital.");
            } // If there are enough auxiliary nurses to complete the request, send the
            // request
            else {
```

```
            chiefNurse.getMedicAuxiliaryRequests().put(this, auxiliaryNursesRequested);
            ClearConsole.clearConsole();
            System.out.println("Foram requisitados " +
Integer.toString(auxiliaryNursesRequested)
                    + " enfermeiros auxiliares\n");
            AwaitsUserInput.awaitsUserInput();
        }

    }
  }

} package Hospital_Database.Person;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.Random;
import java.util.Set;

import Hospital_Database.Hospital;
import Hospital_Database.Menu;
import Hospital_Database.Remedy;
import Hospital_Database.Exceptions.IDNotFoundException;
import Hospital_Database.Exceptions.NoPacientsAwaitingCureException;
import Hospital_Database.Exceptions.NoPacientsToDiagnoseException;
import Hospital_Database.UserInterface.AwaitsUserInput;
import Hospital_Database.UserInterface.ClearConsole;

public class Nurse extends Person {
```

```
private Medic associatedMedic;
private HashMap<Person, ArrayList<Remedy>> schedule;
private int careerYears;
private Queue<Person> pacientsWaitingForDiagnostic;

// ! Constructor
public Nurse(int ID, String name, int birthdayYear, int careerYears) {
   super(ID, name, birthdayYear);
   this.careerYears = careerYears;
   schedule = new HashMap<Person, ArrayList<Remedy>>();
   pacientsWaitingForDiagnostic = new LinkedList<>();

}

// ! Getters & Setters
public int getCareerYears() {
   return careerYears;
}

public void setCareerYears(int newCareerYears) {
   careerYears = newCareerYears;
}

public HashMap<Person, ArrayList<Remedy>> getSchedule() {
   return schedule;
}

public Medic getAssociatedMedic() {
   return associatedMedic;
}
```

```
    public void setAssociatedMedic(Medic newAssociatedMedic) {
        associatedMedic = newAssociatedMedic;
    }

    // ! Getters
    public Queue<Person> getPacientsWaitingForDiagnostic() {
        return pacientsWaitingForDiagnostic;
    }

    // ! Nurse menu related methods
    public void helpsDiagnostic(Hospital hospital) throws NoPacientsToDiagnoseException {

        ClearConsole.clearConsole();

        // If there aren't any pacients waiting for diagnostic, throw an exception
        if (pacientsWaitingForDiagnostic.size() == 0) {
            throw new NoPacientsToDiagnoseException("Não existem pacientes à espera de
diagnóstico.");
        } // If there are pacients waiting for diagnostic, diagnose the first one
        else {
            Person currentPacient = pacientsWaitingForDiagnostic.poll();

            System.out.println(currentPacient + ".");
            System.out.println("Temperatura: " + currentPacient.getTemperature() + ".");
            System.out.println("Nível de glóbulos brancos: " +
currentPacient.getWhiteBloodCellLevels() + ".");

            if (currentPacient.hasGastrointestinalSymptoms()) {
                System.out.println("Sintomas gastrointestinais: Sim.");
            } else {
                System.out.println("Sintomas gastrointestinais: Não.");
```

```
}

// Verify the symptoms and add the pacient to the schedule, to apply the various
// cures
ArrayList<Remedy> remediesToApply = new ArrayList<>();
boolean hasSymptoms = false;

if (currentPacient.getTemperature() > 37.5) {
   hasSymptoms = true;
   remediesToApply.add(new Remedy("Covid"));
}

if (currentPacient.getWhiteBloodCellLevels() < 0.5) {
   hasSymptoms = true;

   remediesToApply.add(new Remedy("HIV"));
}

if (currentPacient.hasGastrointestinalSymptoms()) {
   hasSymptoms = true;
   remediesToApply.add(new Remedy("Ebola"));
}

// If the pacient doesn't have symptoms, remove it from the hospital
if (!hasSymptoms) {
   hospital.getRegistry().put(currentPacient, remediesToApply);
   System.out.println("O paciente não tinha sintomas, e foi removido do hospital\n");
   AwaitsUserInput.awaitsUserInput();
}

// If the pacient has symptoms, add it to schedule
```

```
            if (hasSymptoms) {
                schedule.put(currentPacient, remediesToApply);
                System.out.println("O paciente tem sintomas, e foi adicionado à agenda de curativos.\n");
                AwaitsUserInput.awaitsUserInput();
            }

        }
    }

    public void applyCureToPacient(Hospital hospital) throws NoPacientsAwaitingCureException {
        // Applies cure to the first pacient in the nurses waiting schedule

        ClearConsole.clearConsole();

        // List<Person> pacientsWaitingCure = new ArrayList<>(schedule.keySet());
        Queue<Person> pacientsWaitingCure = new LinkedList<>(schedule.keySet());

        // If there are no pacients waiting for cure, throw an exception
        if (pacientsWaitingCure.size() == 0) {
            throw new NoPacientsAwaitingCureException("Não existe nenhum paciente à espera de cura.\n");
        } // If there are pacients waiting for cure, apply cure to the first one
        else {

            Person currentPacient = pacientsWaitingCure.poll();

            // Apply cure to first pacient
            System.out.println("Introduza a data do curativo: ");
            String cureDate = Menu.scanner.next();
```

```
        // Sets the date of the remedy
        for (Remedy remedyToApply : schedule.get(currentPacient)) {
            remedyToApply.setDateApplied(cureDate);
        }

        Random random = new Random();
        int probabilityOfDeath = random.nextInt(10 - 1 + 1) + 1;

        // Pacient died after administring cure, save to hospital history
        if (probabilityOfDeath == 1) {
            currentPacient.setDead(true);

            hospital.getRegistry().put(currentPacient, schedule.get(currentPacient));

            System.out.println("O paciente morreu após serem aplicados os curativos.");
            Menu.scanner.next();

        } // If the cure is sucessful, send to medic for discharge
        else {
            associatedMedic.getPacientsAwaitingDischarge().add(currentPacient);
            System.out.println("Os curativos foram bem sucedidos, o paciente foi enviado de
volta ao médico.");
            Menu.scanner.next();
        }

    }
  }

  public void listPacientsWaitingForCure() {
    // Prints the pacients awaiting cure to the console
```

```
    ClearConsole.clearConsole();

    Set<Person> pacientsWaitingCure = schedule.keySet();

    System.out.println("Pacientes a aguardar cura");

    for (Person pacient : pacientsWaitingCure) {

        System.out.println(pacient.toString());

    }

    // Awaits for user input
    AwaitsUserInput.awaitsUserInput();
}

public void listPacientsWaitingForDiagnostic() {
    // Prints the pacients awaiting diagnostic to the console

    ClearConsole.clearConsole();

    System.out.println("Pacientes a aguardar diagnóstico");

    for (Person pacient : pacientsWaitingForDiagnostic) {

        System.out.println(pacient.toString());

    }

    // Awaits for user input
    AwaitsUserInput.awaitsUserInput();
```

```
}

public void listMedicNurses(Hospital hospital) throws IDNotFoundException {
   // Lists all nurses associated with a medic

   ClearConsole.clearConsole();

   System.out.println("ID do médico: ");
   int medicID = Menu.scanner.nextInt();

   // Check if a medic with the ID exists
   Medic medic = null;
   for (Medic tempMedic : hospital.getMedics()) {

      if (tempMedic.getID() == medicID) {
         medic = tempMedic;
      }

   }

   // If medic doesn't exist, throws an exception
   if (medic == null) {
      throw new IDNotFoundException("Não existe um médico com o ID " + medicID + ".");
   } // If medic exists, prints it's nurses to the console
   else {

      ClearConsole.clearConsole();

      List<AuxiliaryNurse> medicAuxiliaryNurses = medic.getAuxiliaryNurses();
      List<SpecialistNurse> medicSpecialistNurses = medic.getSpecialistNurses();
```

```
            System.out.println("Médico (ID: " + medicID + ").\n");
            System.out.println("Enfermeiros auxiliares\n");

            for (AuxiliaryNurse tempAuxiliaryNurse : medicAuxiliaryNurses) {

                System.out.println(tempAuxiliaryNurse.toString() + "\n");

            }

            System.out.println("Enfermeiros especialistas\n");

            for (SpecialistNurse tempSpecialistNurse : medicSpecialistNurses) {

                System.out.println(tempSpecialistNurse.toString() + "\n");

            }

        }

        // Waits for user input
        AwaitsUserInput.awaitsUserInput();

    }

    @Override
    public String toString() {

        // If the nurse isn't associated to a medic
        if (associatedMedic == null) {
            return "ID: " + super.getID() + "\n" + "Nome: " + super.getName() + "\n" + "Anos de
carreira: "
```

```
                + careerYears;

        } // If the nurse is associated to a medic
        else {
            return "ID: " + super.getID() + "\n" + "Nome: " + super.getName() + "\n" + "Anos de
carreira: "
                + careerYears + "\nAlocado ao médico:\n" + "ID: " + associatedMedic.getID() + "\n"
+ "Nome: "
                + associatedMedic.getName();
        }

    }

    @Override
    public boolean equals(Object object) {
        return super.equals(object);
    }
}
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Hospital_Database.Person;

import java.util.Random;

/**
 *
 * @author mistakx
 */
public class Person implements Infectable {
```

```
private int ID;
private String name;
private int birthdayYear;
private double temperature;
private double whiteBloodCellLevels;
private boolean gastrointestinalSymptoms;
private boolean dead;

public Person(int ID, String name, int birthdayYear) {
   this.ID = ID;
   this.name = name;
   this.birthdayYear = birthdayYear;
   this.dead = false;

}

// ! Getters and setters
public boolean isDead() {
   return dead;
}

public void setDead(boolean dead) {
   this.dead = dead;
}

public boolean hasGastrointestinalSymptoms() {
   return gastrointestinalSymptoms;
}

public void setGastrointestinalSymptoms(boolean gastrointestinalSymptoms) {
```

```
    this.gastrointestinalSymptoms = gastrointestinalSymptoms;
}

public double getWhiteBloodCellLevels() {
    return whiteBloodCellLevels;
}

public void setWhiteBloodCellLevels(double whiteBloodCellLevels) {
    this.whiteBloodCellLevels = whiteBloodCellLevels;
}

public double getTemperature() {
    return temperature;
}

public void setTemperature(double temperature) {
    this.temperature = temperature;
}

public void setID(int iD) {
    this.ID = iD;
}

public int getID() {
    return ID;
}

public int getBirthdayYear() {
    return birthdayYear;
}
```

```
    public void setBirthdayYear(int birthdayYear) {
        this.birthdayYear = birthdayYear;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    // ! Overriding methods
    @Override
    public void infect() {
        // Generate the person symptoms
        Random random = new Random();

        temperature = 35 + random.nextInt(7);
        whiteBloodCellLevels = 0.05 + (Math.random() * (0.96));
        gastrointestinalSymptoms = Math.random() < 0.5;

    }

    @Override
    public String toString() {
        return "ID: " + ID + "\n" + "Nome: " + name + "\n" + "Ano de nascimento: " + birthdayYear
+ ".";
    }

    @Override
```

```
    public boolean equals(Object object) {
        if (this == object) {
            return true;
        }

        if (object == null) {
            return false;
        }

        if (this.getClass() != object.getClass()) {
            return false;
        }

        Person person = (Person) object;
        return ID == person.getID();
    }

}
package Hospital_Database.Person;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Queue;

import Hospital_Database.Hospital;
import Hospital_Database.Menu;
import Hospital_Database.Exceptions.IDNotFoundException;
import Hospital_Database.Exceptions.NoMedicRequestsExistException;
import Hospital_Database.UserInterface.AwaitsUserInput;
import Hospital_Database.UserInterface.ClearConsole;
```

```
public class SpecialistNurse extends Nurse implements ChiefNurse {

   // If the specialist nurse is a chief nurse, it can receive requests from a
   // medic for auxiliary nurses
   public HashMap<Medic, Integer> medicRequests;

   // ! Constructor
   public SpecialistNurse(int ID, String name, int birthdayYear, int careerYears) {
      super(ID, name, birthdayYear, careerYears);
      medicRequests = new HashMap<>();
   }

   // ! Getter
   @Override
   public HashMap<Medic, Integer> getMedicAuxiliaryRequests() {
      return medicRequests;
   }

   // ! Overriding the interface methods
   @Override
   public void listMedicAuxiliaryRequests() throws NoMedicRequestsExistException {
      // Prints all medic requests for auxiliares.
      // This method can only be called by a chief nurse.

      ClearConsole.clearConsole();

      // If there no medic requests for auxiliaries, throw an exception
      if (medicRequests.size() == 0) {
         throw new NoMedicRequestsExistException("Não existe nenhum requisito de
auxiliares.");
```

```
    } // If there medic requests for auxiliaries, print them to the console
    else {

      ArrayList<Medic> medics = new ArrayList<>(medicRequests.keySet());

      System.out.println("Pedidos de enfermeiros auxiliares\n");

      for (Medic medic : medics) {
        System.out.println(
            medic.toString() + "\nNúmero de pedidos: " +
String.valueOf(medicRequests.get(medic)) + "\n");
      }

    }

    // Waits for user input
    AwaitsUserInput.awaitsUserInput();

  }

  @Override
  public void fulfilMedicAuxiliaryRequest(Hospital hospital) throws
NoMedicRequestsExistException {
    // Fulfils a medics request for auxiliary nurses.
    // This method can only be called by a chief nurse.

    ClearConsole.clearConsole();

    // Input medic ID to fulfil request
    System.out.println("ID do médico que requisitou auxiliares: ");
    int medicID = Menu.scanner.nextInt();
```

```
Queue<Medic> medicsAwaitingRequests = new LinkedList<>(medicRequests.keySet());

Medic medic = null;
for (Medic tempMedic : medicsAwaitingRequests) {
   if (tempMedic.getID() == medicID) {
      medic = tempMedic;
   }
}

// If medic doesn't exist, throw an exception
if (medic == null) {
   throw new NoMedicRequestsExistException(
         "Não existe nenhum médico com o ID introduzido à espera de auxiliares.");
} // If medic exists, fulfils his requests
else {
   int numberOfRequestedAuxiliaries = medicRequests.get(medic);
   int numberOfAuxiliariesAttributed = 0;

   for (AuxiliaryNurse tempAuxiliaryNurse : hospital.getAuxiliaryNurses()) {

      // If the associated nurse doesn't have an associated medic already, attributes
      // it to the medic
      if (tempAuxiliaryNurse.getAssociatedMedic() == null) {
         medic.getAuxiliaryNurses().add(tempAuxiliaryNurse);
         tempAuxiliaryNurse.setAssociatedMedic(medic);
         numberOfAuxiliariesAttributed++;

         if (numberOfAuxiliariesAttributed == numberOfRequestedAuxiliaries) {
            break;
         }
```

```
            }
        }

        System.out.println("Auxiliares atribuídos");
        AwaitsUserInput.awaitsUserInput();

    }

}

@Override
public void attributeSpecialistNurseToMedic(Hospital hospital) throws IDNotFoundException
{ //
    // Attributes a specialist nurse to a medic. This method can only be called by a
    // chief nurse.

    ClearConsole.clearConsole();

    // ! Asks the user for the specialist nurse to attribute to a medic
    System.out.println("ID do enfermeiro especialista a atribuir: ");
    int specialistNurseID = Menu.scanner.nextInt();

    SpecialistNurse specialistNurse = null;

    // Check if a specialist nurse with the ID exists
    for (SpecialistNurse tempSpecialistNurse : hospital.getSpecialistNurses()) {

        if (tempSpecialistNurse.getID() == specialistNurseID) {
            specialistNurse = tempSpecialistNurse;
            break;
        }
```

```
        }

        // If the specialist nurse doesn't exist, throw an exception
        if (specialistNurse == null) {
            throw new IDNotFoundException("Não existe um enfermeiro especialista com o ID " +
specialistNurseID + ".");
        } // ! If the specialist nurse exists, asks the user for Medic and checks if it
        // exists
        else {
            ClearConsole.clearConsole();
            System.out.println("ID do médico: ");
            int medicID = Menu.scanner.nextInt();

            // Check if a medic with the ID exists
            Medic medic = null;
            for (Medic tempMedic : hospital.getMedics()) {

                if (tempMedic.getID() == medicID) {
                    medic = tempMedic;
                    break;
                }

            }

            // If the medic doesn't exist, throws an exception
            if (medic == null) {
                throw new IDNotFoundException("Não existe um médico com o ID " + medicID + ".");
            } // If the medic exists, attribute the specialist to the found medic
            else {
                specialistNurse.setAssociatedMedic(medic);
```

```
                medic.getSpecialistNurses().add(specialistNurse);
                ClearConsole.clearConsole();
                System.out.println("Enfermeiro especialista atribuído.");
                Menu.scanner.next();
            }

        }

    }

    // ! Overriding object methods
    @Override
    public boolean equals(Object object) {
        return super.equals(object);
    }

    @Override
    public String toString() {
        return super.toString();
    } package Hospital_Database.UserInterface;

import java.util.Scanner;

public class AwaitsUserInput {

    private static Scanner scanner = new Scanner(System.in);

    public static void awaitsUserInput() {
        scanner.next();
    }
```

```
}
} package Hospital_Database.UserInterface;

public class ClearConsole {

   public static void clearConsole() {

      System.out.print("\033[H\033[2J");
      System.out.flush();

   }

};
package Hospital_Database;

import Hospital_Database.UserInterface.ClearConsole;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.Queue;
import java.util.Scanner;

import Hospital_Database.Person.AuxiliaryNurse;
import Hospital_Database.Person.Medic;
import Hospital_Database.Person.Person;
import Hospital_Database.Person.SpecialistNurse;

public class Files {

   public static void populateHospital(String fileLocation, Hospital hospital) {
```

```
// At the beginning of the program execution, reads a file and populates the
// hospital accordingly.
// The hospital last attributed ID is also synced with the file's max ID.
List<Medic> medics = hospital.getMedics();
List<AuxiliaryNurse> auxiliaryNurses = hospital.getAuxiliaryNurses();
List<SpecialistNurse> specialistNurses = hospital.getSpecialistNurses();
List<SpecialistNurse> chiefNurses = hospital.getChiefNurses();

Queue<Person> pacientsQueue = hospital.getPacientQueue();

File file = new File(fileLocation);

try {
   file.createNewFile(); // Creates file if it doesn't already exist
   Scanner scanner = new Scanner(file);

   // Each line from the populate file is a different person
   while (scanner.hasNextLine()) {

      // Reads the person's class from the file
      String person = scanner.nextLine();
      String[] parsedPerson = person.split(",");
      String personClass = parsedPerson[0];

      // Decides on what to do, based on the person's class
      switch (personClass) {
         case "Person":

            int pacientID = Integer.parseInt(parsedPerson[1]);
            String pacientName = parsedPerson[2];
            int pacientBirthdayYear = Integer.parseInt(parsedPerson[3]);
```

```
    Person newPerson = new Person(pacientID, pacientName, pacientBirthdayYear);

    pacientsQueue.add(newPerson);

    // Increments the hospital's last attributed ID, if necessary
    if (pacientID > Hospital.getLastIDAttributed()) {
        Hospital.setLastIDAttributed(Hospital.getLastIDAttributed() + 1);
    }

    break;

case "Medic":

    int medicID = Integer.parseInt(parsedPerson[1]);
    String medicName = parsedPerson[2];
    int medicBirthdayYear = Integer.parseInt(parsedPerson[3]);

    Medic newMedic = new Medic(medicID, medicName, medicBirthdayYear);

    medics.add(newMedic);

    // Increments the hospital's last attributed ID, if necessary
    if (medicID > Hospital.getLastIDAttributed()) {
        Hospital.setLastIDAttributed(Hospital.getLastIDAttributed() + 1);
    }

    break;

case "AuxiliaryNurse":
```

```
                int auxiliaryNurseID = Integer.parseInt(parsedPerson[1]);
                String auxiliaryNurseName = parsedPerson[2];
                int auxiliaryNurseBirthdayYear = Integer.parseInt(parsedPerson[3]);
                int auxiliaryNurseCareerYears = Integer.parseInt(parsedPerson[4]);

                AuxiliaryNurse newAuxiliary = new AuxiliaryNurse(auxiliaryNurseID,
auxiliaryNurseName,
                        auxiliaryNurseBirthdayYear, auxiliaryNurseCareerYears);

                auxiliaryNurses.add(newAuxiliary);

                // Increments the hospital's last attributed ID, if necessary
                if (auxiliaryNurseID > Hospital.getLastIDAttributed()) {
                   Hospital.setLastIDAttributed(Hospital.getLastIDAttributed() + 1);
                }

                break;

             case "SpecialistNurse":
                int specialistNurseID = Integer.parseInt(parsedPerson[1]);
                String specialistNurseName = parsedPerson[2];
                int specialistNurseBirthdayYear = Integer.parseInt(parsedPerson[3]);
                int specialistNurseCareerYears = Integer.parseInt(parsedPerson[4]);

                SpecialistNurse newSpecialistNurse = new SpecialistNurse(specialistNurseID,
specialistNurseName,
                        specialistNurseBirthdayYear, specialistNurseCareerYears);

                specialistNurses.add(newSpecialistNurse);

                // Increments the hospital's last attributed ID, if necessary
                if (specialistNurseID > Hospital.getLastIDAttributed()) {
```

```
                Hospital.setLastIDAttributed(Hospital.getLastIDAttributed() + 1);
            }

            break;

        case "ChiefNurse":
            int chiefNurseID = Integer.parseInt(parsedPerson[1]);
            String chiefNurseName = parsedPerson[2];
            int chiefNurseBirthdayYear = Integer.parseInt(parsedPerson[3]);
            int chiefNurseCareerYears = Integer.parseInt(parsedPerson[4]);

            SpecialistNurse newChiefNurse = new SpecialistNurse(chiefNurseID,
chiefNurseName,
                    chiefNurseBirthdayYear, chiefNurseCareerYears);

            chiefNurses.add(newChiefNurse);

            // Increments the hospital's last attributed ID, if necessary
            if (chiefNurseID > Hospital.getLastIDAttributed()) {
                Hospital.setLastIDAttributed(Hospital.getLastIDAttributed() + 1);
            }

            break;

        default:
            break;
      }

    }

    scanner.close();
```

```
        } catch (IOException exception) {
            ClearConsole.clearConsole();
            System.out.println(exception.getMessage());
            Menu.scanner.next();
        }

    }
}
package Hospital_Database;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.Random;

import Hospital_Database.Exceptions.IDNotFoundException;
import Hospital_Database.Exceptions.NoMedicRequestsExistException;
import Hospital_Database.Exceptions.NoMedicsExistException;
import Hospital_Database.Exceptions.NoNursesExistException;
import Hospital_Database.Exceptions.NoPacientsInWaitingQueueException;
import Hospital_Database.Exceptions.NotEnoughAuxiliaryNursesException;
import Hospital_Database.Exceptions.NotEnoughCareerYearsException;
import Hospital_Database.Person.AuxiliaryNurse;
import Hospital_Database.Person.Medic;
import Hospital_Database.Person.Person;
import Hospital_Database.Person.SpecialistNurse;
import Hospital_Database.UserInterface.AwaitsUserInput;
import Hospital_Database.UserInterface.ClearConsole;
```

```
public class Hospital {

   // ! Class-based singleton implementation
   private static Hospital INSTANCE;

   public static Hospital getInstance() {
      if (INSTANCE == null) {
         INSTANCE = new Hospital();
      }

      return INSTANCE;
   }

   // ! Instance variables
   private static int lastIDAttributed = 0;
   final private int NECESSARY_YEARS_FOR_PROMOTION = 20;
   private List<Medic> medics;
   private List<AuxiliaryNurse> auxiliaryNurses;
   private List<SpecialistNurse> specialistNurses;
   private List<SpecialistNurse> chiefNurses;
   private Queue<Person> pacientsQueue;
   private HashMap<Medic, Integer> auxiliaryRequests;
   private HashMap<Person, ArrayList<Remedy>> registry;

   // ! Contrustor (private because its a singleton)
   private Hospital() {
      medics = new ArrayList<>();
      auxiliaryNurses = new ArrayList<>();
      specialistNurses = new ArrayList<>();
      chiefNurses = new ArrayList<>();
```

```
    pacientsQueue = new LinkedList<>();
    auxiliaryRequests = new HashMap<>();
    registry = new HashMap<>();
}

// ! Getters and setters
public HashMap<Person, ArrayList<Remedy>> getRegistry() {
    return registry;
}

public int getNECESSARY_YEARS_FOR_PROMOTION() {
    return NECESSARY_YEARS_FOR_PROMOTION;
}

public static int getLastIDAttributed() {
    return lastIDAttributed;
}

public static void setLastIDAttributed(int lastID) {
    Hospital.lastIDAttributed = lastID;
}

public List<Medic> getMedics() {
    return medics;
}

public List<AuxiliaryNurse> getAuxiliaryNurses() {
    return auxiliaryNurses;
}

public List<SpecialistNurse> getSpecialistNurses() {
```

```
    return specialistNurses;
}

public List<SpecialistNurse> getChiefNurses() {
    return chiefNurses;
}

public Queue<Person> getPacientQueue() {
    return pacientsQueue;
}

public HashMap<Medic, Integer> getAuxiliaryRequests() {
    return auxiliaryRequests;
}

// ! Populate hospital method
protected void populate(String hospitalDatabaseFilePath) {
    Files.populateHospital(hospitalDatabaseFilePath, this);
}

// ! Administrator menu related methods
public void addMedic() { // Creates and adds new medic to the hospital

    ClearConsole.clearConsole();

    // Input medic name
    System.out.println("Nome do médico: ");
    String name = Menu.scanner.next();

    // Input medic birthday year
    System.out.println("Ano de nascimento do médico: ");
```

```
        int birthdayYear = Menu.scanner.nextInt();

        // Create medic, add medic to the list, and add 1 to the last ID
        Medic newMedic = new Medic(lastIDAttributed + 1, name, birthdayYear);
        medics.add(newMedic);
        lastIDAttributed++;

    }

    public void addSpecialistNurse() { // Creates and adds specialist nurse to the hospital

        ClearConsole.clearConsole();

        // Input specialist nurse name
        System.out.println("Nome do enfermeiro especialista: ");
        String name = Menu.scanner.next();

        // Input specialist nurse birthday year
        System.out.println("Ano de nascimento do enfermeiro especialista: ");
        int birthdayYear = Menu.scanner.nextInt();

        // Input specialist nurse career years
        System.out.println("Anos de carreira do enfermeiro especialista: ");
        int careerYears = Menu.scanner.nextInt();

        // Create new specialist nurse, add it to the list and add 1 to last ID.
        SpecialistNurse newSpecialistNurse = new SpecialistNurse(lastIDAttributed + 1, name,
birthdayYear, careerYears);
        specialistNurses.add(newSpecialistNurse);
        lastIDAttributed++;
```

```
    }

    public void addAuxiliaryNurse() { // Creates and adds auxiliary nurse to the hospital

        ClearConsole.clearConsole();

        // Input auxiliary nurse name
        System.out.println("Insira o nome do enfermeiro auxiliar: ");
        String name = Menu.scanner.next();

        // Input auxiliary nurse birthday year
        System.out.println("Ano de nascimento do enfermeiro auxiliar: ");
        int birthdayYear = Menu.scanner.nextInt();

        // Input auxiliary nurse career years
        System.out.println("Ano de carreira do enfermeiro auxiliar: ");
        int careerYears = Menu.scanner.nextInt();

        // Create new auxiliary nurse, add it to the list, and add 1 to the last ID.
        AuxiliaryNurse newAuxiliaryNurse = new AuxiliaryNurse(lastIDAttributed + 1, name,
birthdayYear, careerYears);
        auxiliaryNurses.add(newAuxiliaryNurse);
        lastIDAttributed++;

    }

    public void addNewPacient() { // Adds new pacient to the hospital

        ClearConsole.clearConsole();

        // Input pacient name
```

```
        System.out.println("Insira o nome do paciente: ");
        String name = Menu.scanner.next();

        // Input pacient birthday year
        System.out.println("Ano de nascimento do novo paciente: ");
        int birthdayYear = Menu.scanner.nextInt();

        // Create pacient, add pacient to list, and add 1 to the last ID generated
        Person newPacient = new Person(lastIDAttributed, name, birthdayYear);
        pacientsQueue.add(newPacient);
        lastIDAttributed++;

    }

    public void promoteSpecialistNurseToChief() throws NotEnoughCareerYearsException,
IDNotFoundException {
        // Promotes a specialist nurse to chief

        ClearConsole.clearConsole();

        // Input ID of specialist nurse to be promoted
        int specialistNurseID;
        System.out.println("ID do enfermeiro especialista que deseja promover a chefe: ");
        specialistNurseID = Menu.scanner.nextInt();

        // Check if a specialist nurse with the ID exists
        boolean specialistNurseExists = false;

        for (SpecialistNurse tempSpecialistNurse : specialistNurses) {

            // If specialist nurse exists
```

```
            if (tempSpecialistNurse.getID() == specialistNurseID) {
                specialistNurseExists = true;

                int nurseCareerYears = tempSpecialistNurse.getCareerYears();

                // Promote specialist nurse to chief nurse
                if (nurseCareerYears >= NECESSARY_YEARS_FOR_PROMOTION) {

                    chiefNurses.add(tempSpecialistNurse);
                    specialistNurses.remove(tempSpecialistNurse);

                } else {
                    throw new NotEnoughCareerYearsException(
                            "O enfermeiro especialista não tem anos de carreira suficientes.");
                }

                break;
            }

        }

        if (!specialistNurseExists) {
            throw new IDNotFoundException("Não existe um enfermeiro especialista com o ID " +
specialistNurseID + ".");
        }

    }

    public void progressNursesCareerYears() {
        // Add 1 to all nurse career years
```

```
        // Progress specialist nurses career years
        for (int i = 0; i < specialistNurses.size(); i++) {
            SpecialistNurse tempSpecialistNurse = specialistNurses.get(i);
            tempSpecialistNurse.setCareerYears(tempSpecialistNurse.getCareerYears() + 1);
        }

        // Progress auxiliary nurses career years
        for (int i = 0; i < auxiliaryNurses.size(); i++) {
            AuxiliaryNurse tempAuxiliaryNurse = auxiliaryNurses.get(i);
            tempAuxiliaryNurse.setCareerYears(tempAuxiliaryNurse.getCareerYears() + 1);
        }

        // Progress chief nurses career years
        for (int i = 0; i < chiefNurses.size(); i++) {
            SpecialistNurse tempChiefNurse = chiefNurses.get(i);
            tempChiefNurse.setCareerYears(tempChiefNurse.getCareerYears() + 1);
        }

        ClearConsole.clearConsole();
        System.out.println("Foi adicionado 1 ano de carreira a todos os enfermeiros existentes.");
        Menu.scanner.next();

    }

    public void listNurses() throws NoNursesExistException { // Prints all nurses in the hospital to
the console

        ClearConsole.clearConsole();

        // If no nurses exist in the hospital, throw an exception
        if ((specialistNurses.size() == 0) && (auxiliaryNurses.size() == 0) && (chiefNurses.size() ==
0)) {
```

```
            throw new NoNursesExistException("Não existem enfermeiros no hospital.");
        } // If there are nurses in the hospital, print them to the console
        else {

            // List auxiliary nurses
            System.out.println("Enfermeiros auxiliares\n");
            for (AuxiliaryNurse tempAuxiliaryNurse : auxiliaryNurses) {
                System.out.println(tempAuxiliaryNurse.toString() + "\n");
            }

            // List specialist nurses
            System.out.println("\n\nEnfermeiros especialista\n");
            for (SpecialistNurse tempSpecialistNurse : specialistNurses) {
                System.out.println(tempSpecialistNurse.toString() + "\n");
            }

            // List chief nurses
            System.out.println("\n\nEnfermeiros chefe\n");
            for (SpecialistNurse tempChiefNurse : chiefNurses) {
                System.out.println(tempChiefNurse.toString() + "\n");
            }

        }

        // Waits for user input
        Menu.scanner.next();

    }

    public void listMedics() throws NoMedicsExistException { // Prints all medic in the hospital to
the console
```

```
    ClearConsole.clearConsole();

    // If there are no medics in the hospital, throw an exception
    if (medics.size() == 0) {
        throw new NoMedicsExistException("Não existem médicos no hospital.");
    } // If there are medics in the hospital, print them to the console
    else {

        // Prints all medics to the console
        System.out.println("Médicos no hospital\n");
        for (int i = 0; i < medics.size(); i++) {
            Medic tempMedic = medics.get(i);
            System.out.println(tempMedic.toString() + "\n");
        }
    }

    // Waits for user input
    Menu.scanner.next();

}

public void listRequestsForAuxiliaryNurses() throws NoMedicRequestsExistException {
    // Lists all requests for auxiliary nurses sent to the hospital, in the case
    // they couldn't be completed by a chief nurse

    ClearConsole.clearConsole();

    // If there are no requests in the hospital, throw an exception
    if (auxiliaryRequests.size() == 0) {
```

```
        throw new NoMedicRequestsExistException("Não existem pedidos de enfermeiros
auxiliares no hospital.");
    } else {

        ArrayList<Medic> medics = new ArrayList<>(auxiliaryRequests.keySet());
        System.out.println("Pedidos de enfermeiros auxiliares\n");

        for (Medic medic : medics) {
            System.out.println(medic.toString() + "\nNúmero de pedidos: "
                    + String.valueOf(auxiliaryRequests.get(medic)) + "\n");
        }

    }
    // Wait for user input
    Menu.scanner.next();

  }

  public void trashRequestsForAuxiliaryNurses() throws NoMedicRequestsExistException {
    // Trash a random amount of requests for auxiliary nurses made to the hospital

    // If there aren't any requests, throw exception
    if (auxiliaryRequests.size() == 0) {
        throw new NoMedicRequestsExistException("Não existem pedidos de enfermeiros
auxiliares.");
    } // If there are requests in the hospital, trash a random number of them
    else {

        Random random = new Random();

        ArrayList<Medic> medics = new ArrayList<>(auxiliaryRequests.keySet());
```

```
        int requestsToTrash = random.nextInt(auxiliaryRequests.size() - 1 + 1) + 1;

        for (int i = 0; i < requestsToTrash; i++) {

            auxiliaryRequests.remove(medics.get(i));

        }

        ClearConsole.clearConsole();
        System.out.println("Foram removidos " + requestsToTrash + " pedidos de
auxiliares.\n");

        // Waits for user input
        Menu.scanner.next();
    }
  }

  public void virusOutbreak() {
    // Infects random amount of staff

    Random random = new Random();
    int probabilityOfInfection;

    int peopleInfected = 0;

    // Infects medics
    for (Medic medic : medics) {
        probabilityOfInfection = random.nextInt(10 - 1 + 1) + 1;

        if (probabilityOfInfection == 1) {
            medic.infect();
```

```
            pacientsQueue.add(medic);
            medics.remove(medic);
            peopleInfected++;
        }

    }

    // Infects auxiliary nurses
    for (AuxiliaryNurse auxiliaryNurse : auxiliaryNurses) {
        probabilityOfInfection = random.nextInt(10 - 1 + 1) + 1;

        if (probabilityOfInfection == 1) {
            auxiliaryNurse.infect();
            pacientsQueue.add(auxiliaryNurse);
            auxiliaryNurses.remove(auxiliaryNurse);
            peopleInfected++;

        }

    }

    for (SpecialistNurse specialistNurse : specialistNurses) {
        probabilityOfInfection = random.nextInt(10 - 1 + 1) + 1;

        if (probabilityOfInfection == 1) {
            specialistNurse.infect();
            pacientsQueue.add(specialistNurse);
            specialistNurses.remove(specialistNurse);
            peopleInfected++;

        }
```

```
    }

    for (SpecialistNurse chiefNurse : chiefNurses) {
        probabilityOfInfection = random.nextInt(10 - 1 + 1) + 1;

        if (probabilityOfInfection == 1) {
            chiefNurse.infect();
            pacientsQueue.add(chiefNurse);
            chiefNurses.remove(chiefNurse);
            peopleInfected++;

        }

    }

    System.out.println("Foram infetadas " + peopleInfected + " pessoas.");
    AwaitsUserInput.awaitsUserInput();

}

public void hospitalReports() {
    // Generates a hospital report, and prints it to the console

    ClearConsole.clearConsole();

    System.out.println("--------------------------------------------------------------------------");
    System.out.println("                    **RELATORIO HOSPITALAR**");
    System.out.println("--------------------------------------------------------------------------");

    int numberOfDeaths = 0;
```

```
        int numberOfMedicalDischarges = 0;
        int numberOfRemediesApplied = 0;

        ArrayList<Person> pacientsInRegistry = new ArrayList<>(registry.keySet());

        for (Person pacient : pacientsInRegistry) {

            numberOfRemediesApplied += registry.get(pacient).size();

            if (pacient.isDead()) {
                numberOfDeaths++;
            } else {
                numberOfMedicalDischarges++;
            }
        }

        int totalPacients = numberOfDeaths + numberOfMedicalDischarges;
        System.out.println("Total de pacientes recebidos: " + totalPacients + ".\n");
        System.out.println("Altas: " + numberOfMedicalDischarges + ".\n");
        System.out.println("Óbitos: " + numberOfDeaths + ".\n");
        System.out.println("Numero de curativos administrados: " +
String.valueOf(numberOfRemediesApplied) + "\n");

        AwaitsUserInput.awaitsUserInput();
    }

    public void listPacientsInHospitalQueue() throws NoPacientsInWaitingQueueException {
        // List all pacients in the hospital queue

        ClearConsole.clearConsole();
```

```
        // If there aren't any pacients in the hospital waiting queue, throw exception
        if (pacientsQueue.size() == 0) {
            throw new NoPacientsInWaitingQueueException("Não existem pacientes na lista de espera.");
        } // If there are pacients in the hospital waiting queue, print them to the
        // console
        else {

            // Prints pacients information to console
            System.out.println("Pacientes na lista de espera do hospital\n");
            for (Person pacient : pacientsQueue) {
                System.out.println(pacient.toString() + "\n");
            }

        }

        // Waits for user input
        AwaitsUserInput.awaitsUserInput();

    }

    public void fulfilMedicAuxiliaryRequest() throws NoMedicRequestsExistException, NotEnoughAuxiliaryNursesException {
        // Fulfils a medics request for auxiliary nurses.

        ClearConsole.clearConsole();

        // Input medic ID to fulfil request
        System.out.println("Introduza o ID do médico que requisitou auxiliares: ");
        int medicID = Menu.scanner.nextInt();

        Queue<Medic> medicsAwaitingRequests = new LinkedList<>(auxiliaryRequests.keySet());
```

```
        Medic medic = null;
        for (Medic tempMedic : medicsAwaitingRequests) {
            if (tempMedic.getID() == medicID) {
                medic = tempMedic;
            }
        }

        // If medic doesn't exist, throw an exception
        if (medic == null) {
            throw new NoMedicRequestsExistException(
                    "Não existe nenhum médico com o ID introduzido à espera de auxiliares.");
        } // If medic exists, fulfils his requests
        else {

            int numberOfFreeAuxiliaries = 0;

            for (AuxiliaryNurse auxiliaryNurse : auxiliaryNurses) {
                if (auxiliaryNurse.getAssociatedMedic() == null) {
                    numberOfFreeAuxiliaries++;
                }
            }

            int numberOfRequestedAuxiliaries = auxiliaryRequests.get(medic);

            // If there aren't enough auxiliares, throw error
            if (numberOfFreeAuxiliaries < numberOfRequestedAuxiliaries) {
                throw new NotEnoughAuxiliaryNursesException("Não existem auxiliares
suficientes.");
            } // If there are enough auxiliares, fulfil request
            else {
```

```
            int numberOfAuxiliariesAttributed = 0;
            for (AuxiliaryNurse tempAuxiliaryNurse : auxiliaryNurses) {
               // If the associated nurse doesn't have an associated medic already, attributes
               // it to the medic
               if (tempAuxiliaryNurse.getAssociatedMedic() == null) {
                  medic.getAuxiliaryNurses().add(tempAuxiliaryNurse);
                  tempAuxiliaryNurse.setAssociatedMedic(medic);
                  numberOfAuxiliariesAttributed++;

                  if (numberOfAuxiliariesAttributed == numberOfRequestedAuxiliaries) {
                     break;
                  }
               }
            }

            System.out.println("Auxiliares atribuídos");
            AwaitsUserInput.awaitsUserInput();

         }
      }

   }

}
package Hospital_Database;

public class Main {

   public static void main(String[] args) {

      String hospitalDatabaseFilePath = "HospitalDB.txt";
```

```
        Hospital.getInstance().populate(hospitalDatabaseFilePath);

        Menu.mainMenuUserInterface(Hospital.getInstance());

    }

}
package Hospital_Database;

import Hospital_Database.UserInterface.ClearConsole;
import java.util.Scanner;

import Hospital_Database.Exceptions.IDNotFoundException;
import Hospital_Database.Exceptions.NotEnoughPermissionsException;
import Hospital_Database.Person.AuxiliaryNurse;
import Hospital_Database.Person.ChiefNurse;
import Hospital_Database.Person.Medic;
import Hospital_Database.Person.Nurse;
import Hospital_Database.Person.SpecialistNurse;

public class Menu {

    public static Scanner scanner = new Scanner(System.in);
    private static int option;

    public static void mainMenuUserInterface(Hospital hospital) {
        // Prints the main menu user interface to the console

        while (true) {

            try {
```

```
    ClearConsole.clearConsole();
    System.out.println("Selecione o menu que deseja:");
    System.out.println("1 - Menu Médico");
    System.out.println("2 - Menu Enfermeiro");
    System.out.println("3 - Menu Administrador\n");

    option = scanner.nextInt();

    switch (option) {

      case 1:
        medicMenuUserInterface(hospital);
        break;
      case 2:
        nurseMenuUserInterface(hospital);
        break;
      case 3:
        administratorMenuUserInterface(hospital);
        break;

      default:
        ClearConsole.clearConsole();
        System.out.println("Opção inválida.\n");
        scanner.next();
        break;
    }

  } catch (IDNotFoundException exception) {
    ClearConsole.clearConsole();
    System.out.println(exception.getMessage());
    scanner.next();
```

```
        }

    }

}

private static void administratorMenuUserInterface(Hospital hospital) {
    // Prints administrator menu user interface to the console

    while (true) {

        try {

            ClearConsole.clearConsole();
            System.out.println("Selecione uma opção.");
            System.out.println("1 - Criar Médico.");
            System.out.println("2 - Criar Enfeirmeiro-especialista.");
            System.out.println("3 - Criar Enfermeiro-Auxiliar.");
            System.out.println("4 - Criar novo paciente.");
            System.out.println("5 - Promover a enfermeiro-chefe.");
            System.out.println("6 - Aumentar anos de carreira de todos os enfermeiros.");
            System.out.println("7 - Listar enfermeiros.");
            System.out.println("8 - Listar médicos.");
            System.out.println("9 - Listar pedidos para enfermeiros-auxiliares, feitos ao
hospital.");
            System.out.println("10 - Listar pacientes em espera no hospital.");
            System.out.println("11 - Atirar pedidos para enfermeiros-aulixiares para
trituradora.");
            System.out.println("12 - Virus Outbreak.");
            System.out.println("13 - Gerar relatório hospitalar.");
            System.out.println("14 - Atende ao pedido para enfermeiros-auxiliares");
```

```
System.out.println("15 - Sair do programa");
System.out.println("0 - Voltar ao menu anterior.\n");

boolean exitMenuUserInterface = false;
int option;
option = scanner.nextInt();
switch (option) {
   case 1:
      hospital.addMedic();
      break;
   case 2:
      hospital.addSpecialistNurse();
      break;
   case 3:
      hospital.addAuxiliaryNurse();
      break;
   case 4:
      hospital.addNewPacient();
      break;
   case 5:
      hospital.promoteSpecialistNurseToChief();
      break;
   case 6:
      hospital.progressNursesCareerYears();
      break;
   case 7:
      hospital.listNurses();
      break;
   case 8:
      hospital.listMedics();
      break;
```

```
case 9:
   hospital.listRequestsForAuxiliaryNurses();

   break;
case 10:
   hospital.listPacientsInHospitalQueue();

   break;
case 11:
   hospital.trashRequestsForAuxiliaryNurses();
   break;
case 12:
   hospital.virusOutbreak();
   break;
case 13:
   hospital.hospitalReports();
   break;
case 14:
   hospital.fulfilMedicAuxiliaryRequest();
   break;
case 15:
   System.exit(0);
   break;

case 0:
   exitMenuUserInterface = true;
   break;

default:
   ClearConsole.clearConsole();
   System.out.println("Opção inválida.\n");
```

```
                scanner.next();
                break;

            }

            if (exitMenuUserInterface) {
               break;
            }
         } catch (Exception exception) {
            ClearConsole.clearConsole();
            System.out.println(exception.getMessage());
            scanner.next();
         }
      }

   }

   private static void medicMenuUserInterface(Hospital hospital) throws IDNotFoundException
{
      // Prints medic menu user interface to the console

      ClearConsole.clearConsole();

      // Asks the user for ID
      System.out.println("Insira o seu ID: ");
      int medicID = Integer.parseInt(scanner.next());

      // Checks if a medic with the ID exists
      Medic medic = null;
      for (Medic tempMedic : hospital.getMedics()) {
```

```
            if (tempMedic.getID() == medicID) {
                medic = tempMedic;
            }

        }

        // If medic with the ID doesn't exist
        if (medic == null) {

            throw new IDNotFoundException("Não existe nenhum médico com o ID " +
String.valueOf(medicID) + ".");

        } // If medic with the ID exists
        else {

            while (true) {

                try {

                    ClearConsole.clearConsole();
                    System.out.println("Selecione uma opção.");
                    System.out.println("1 - Listar pacientes em espera no hospital.");
                    System.out.println("2 - Listar pacientes a aguardar alta.");
                    System.out.println("3 - Diagnóstico ao paciente.");
                    System.out.println("4 - Dar alta hospitalar.");
                    System.out.println("5 - Requerimento de auxiliares.");
                    System.out.println("0 - Voltar ao menu anterior.\n");

                    boolean exitMenuUserInterface = false;
                    option = scanner.nextInt();
```

```
switch (option) {

   case 1:
      medic.listPacientsInHospitalQueue(hospital);
      break;
   case 2:
      medic.listPacientsAwaitingDischarge();
      ;
      break;
   case 3:
      medic.pacientDiagnostic(hospital);
      break;
   case 4:
      medic.dischargePacient();
      break;
   case 5:
      medic.requestAuxiliaryNurses(hospital);
      break;
   case 0:
      exitMenuUserInterface = true;
      break;
   default:
      ClearConsole.clearConsole();
      System.out.println("Opção inválida.\n");
      scanner.next();
      break;

}

if (exitMenuUserInterface) {
   break;
```

```
                }

            } catch (Exception exception) {
                ClearConsole.clearConsole();
                System.out.println(exception.getMessage());
                scanner.next();
            }
        }

    }

}

    private static void nurseMenuUserInterface(Hospital hospital) throws IDNotFoundException
{
    // Prints nurse menu user interface to the console

    ClearConsole.clearConsole();

    // Asks the user for ID
    System.out.println("Insira o seu ID: ");
    int nurseID = Integer.parseInt(scanner.next());

    // Search for a nurse with the ID in all the nurses lists
    Nurse nurse = null;
    boolean nurseIsChief = false;
    for (AuxiliaryNurse tempAuxiliaryNurse : hospital.getAuxiliaryNurses()) {
        if (tempAuxiliaryNurse.getID() == nurseID) {
            nurse = tempAuxiliaryNurse;
            break;
        }
```

```
}
for (SpecialistNurse tempSpecialistNurse : hospital.getSpecialistNurses()) {
   if (tempSpecialistNurse.getID() == nurseID) {
      nurse = tempSpecialistNurse;
      break;
   }
}
for (SpecialistNurse tempChiefNurse : hospital.getChiefNurses()) {
   if (tempChiefNurse.getID() == nurseID) {
      nurse = tempChiefNurse;
      nurseIsChief = true;
      break;
   }
}

// If the nurse doesn't exist, throw an exception
if (nurse == null) {
   throw new IDNotFoundException("Não existe nenhum enfermeiro com o ID inserido.");
}

while (true) {

   try {
      ClearConsole.clearConsole();
      System.out.println("Selecione uma opção.");
      System.out.println("1 - Listar enfermeiros de um médico.");
      System.out.println("2 - Listar pacientes a aguardar curativo.");
      System.out.println("3 - Atribuir enfermeiro-especialista a médico.");
      System.out.println("4 - Aplicar curativo a um paciente.");
      System.out.println("5 - Listar requisitos de enfermeiros auxiliares.");
      System.out.println("6 - Atender aos requisitos de enfermeiros auxiliares.");
```

```
System.out.println("7 - Diagnóstico ao paciente.");
System.out.println("8 - Listar pacientes à espera de diagnóstico.");
System.out.println("0 - Voltar ao menu anterior.");

boolean exitMenuUserInterface = false;
option = scanner.nextInt();

switch (option) {
   case 1:
      nurse.listMedicNurses(hospital);
      break;
   case 2:
      nurse.listPacientsWaitingForCure();
      break;

   case 3:
      if (nurseIsChief) {
         ((ChiefNurse) nurse).attributeSpecialistNurseToMedic(hospital);

      } else {
         throw new NotEnoughPermissionsException(
               "Apenas um enfermeiro chefe pode atribuír um enfermeiro especialista.");
      }
      break;
   case 4:
      nurse.applyCureToPacient(hospital);

      break;
   case 5:
      if (nurseIsChief) {
         ((ChiefNurse) nurse).listMedicAuxiliaryRequests();
```

```
                } else {
                    throw new NotEnoughPermissionsException(
                            "Apenas um enfermeiro chefe pode ver os pedidos por auxiliares.");
                }
                break;
            case 6:
                if (nurseIsChief) {
                    ((ChiefNurse) nurse).fulfilMedicAuxiliaryRequest(hospital);
                } else {
                    throw new NotEnoughPermissionsException(
                            "Apenas um enfermeiro chefe pode atender ao pedido de enfermeiros
auxiliares.");
                }
                break;
            case 7:
                nurse.helpsDiagnostic(hospital);
                break;
            case 8:
                nurse.listPacientsWaitingForDiagnostic();
                break;
            case 0:
                exitMenuUserInterface = true;
                break;

            default:
                ClearConsole.clearConsole();
                System.out.println("Opção inválida.\n");
                scanner.next();
                break;

        }
```

```
            if (exitMenuUserInterface) {
                break;
            }

        } catch (Exception exception) {
            ClearConsole.clearConsole();
            System.out.println(exception.getMessage());
            scanner.next();

        }
    }

  }

}
package Hospital_Database;

public class Remedy {

  private String name;
  private String dateApplied;

  public Remedy(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public void setDateApplied(String dateApplied) {
        this.dateApplied = dateApplied;
    }

    public String getDateApplied() {
        return dateApplied;
    }
}
```