

Universidad Nacional de Colombia Sede Medellín



Programación Orientada a Objetos

Memoria Escrita

HealthTech-Grupo11 Entrega I

Semestre 2020-2

Presentado a:

Profesor Jaime Alberto Guzmán Luna

Por:

Richard Alexis Montoya Londoño

Iván Santiago Rojas Martínez

Daniel Julián Cardona Álvarez

Luis Santiago Maya Restrepo

Medellín

30 de octubre de 2020

Contenido:

- ***Descripción de la solución: análisis, diseño e implementación.***
- ***Diagramas UML: Vista estática del modelo UML y diagrama de Objetos.***
- ***Implementación de las características de programación orientada a objetos en el proyecto.***
- ***Manual de uso de la aplicación y de las 5 funcionalidades.***

Descripción de la solución: análisis, diseño e implementación.

Análisis:

El equipo 11 propone un software llamado HealthTech para administrar un hospital. La elección de esta propuesta se hace basados en las ricas posibilidades de actores que presenta el dominio del problema. Tales posibilidades hacen que sea un dominio propenso para implementar las características de la programación orientada a objetos. Conceptos tales como el de Hospital, Persona, Administrador, Medico, Paciente, Habitación, Historia Clínica y las interacciones entre sí, nos dieron luces para poner en práctica los conceptos aprendidos durante las clases de programación Orientada a Objetos con el profesor Jaime Guzmán Luna.

El equipo empieza por preguntarse sobre el objetivo de un programa informático, y se llega a la conclusión de que necesariamente tiene que ser la solución a un problema (o varios problemas) de un contexto específico. De esta forma, procedemos a preguntarnos qué problemas podría necesitar solucionar la administración de un hospital, y así poder empezar a pensar en funcionalidades que agregaran valor a la aplicación, específicamente las 5 funcionalidades que pide el profesor de la materia. Se pensó en los siguientes problemas a solucionar, problemas que podemos llamar como **requisitos del sistema**:

1. Poder tener a disposición los detalles básicos del hospital, tales como la ocupación de camas, la cantidad de pacientes inscritos en el sistema, información sobre el administrador e información sobre los médicos que atienden y su respectiva especialidad.
2. Tener la posibilidad de registrar pacientes en el sistema, e identificarlos de forma única, a la vez que se le crean sus respectivas historias clínicas para almacenar el historial del paciente en el hospital.
3. Que un paciente, por medio de un administrador, tenga la posibilidad de crear una solicitud para ser atendido en algún área de la medicina, ya sea Cardiología, Nefrología, Neurología, Oftalmología, entre otras.
4. Que el administrador pueda revisar todas las solicitudes que hay pendientes, para proceder a aprobarlas o no y ponerles un respectivo valor. Si se aprueban se iniciaría un procedimiento médico, se le asignaría una habitación y un médico experto en el área solicitada.
5. Ya que los procedimientos tienen costos se debe tener la opción de pagar los mismos para poner el paciente a paz y salvo.
6. Poder finalizar los procedimientos iniciados.
7. Poder dar de alta al paciente y habilitar la habitación en la que estaba alojado.
8. Consultar detalles sobre los pacientes registrados en el sistema.

Diseño:

Para la fase del diseño, lo primero que se hizo fue definir ciertas **reglas de negocio**:

1. El sistema será un monousuario, y será un usuario de tipo Administrador quién lo va a manejar.
2. Sólo habrá un objeto administrador instanciado en el sistema.
3. Sólo habrá un objeto hospital instanciado en el sistema.

4. Habrá un número fijo de 25 habitaciones, serán 25 objetos de tipo Room previamente cargados al sistema.
5. Habrá un número fijo de 10 médicos, serán 10 objetos de tipo:
Persona medico = new Medico (String nombre, String especialidad),
donde especialidad será alguna de estas, una para cada médico:
Oncología, Pediatría, Urología, Oftalmología, Cardiología, Neurología,
Nefrología, Dermatología, Psiquiatría, Ginecología.
6. Al ingresar un *paciente* al sistema, a este se le asignará una identificación de forma automática, que lo diferenciará de forma inequívoca. Igualmente se creará y se le asignará su respectiva *historia clínica*, la cual tendrá como código el mismo id del paciente. Cada paciente tendrá sólo una historia clínica, y viceversa.
7. Al crear una solicitud, esta será almacenada en las solicitudes del paciente, y en las solicitudes pendientes por revisar del administrador. A la solicitud se le asociará el tipo de actividad médica que se está requiriendo como Cardiología, Nefrología, Neurología, Oftalmología, entre otras.
8. Para crear una solicitud el paciente debe estar registrado en el sistema.
9. Un paciente puede tener tantas solicitudes como desee.
10. Cuando se aprueba una solicitud, se crea un procedimiento, y el administrador le asigna un valor al mismo. También se le asigna un médico al paciente que solicitó, médico que tiene que ser especialista en la materia solicitada, por ejemplo, si se solicitó para Cardiología, el médico asignado debe ser especialista en Cardiología. También se le asigna al paciente una habitación que esté vacía.
11. La aprobación de una solicitud está sujeta a disponibilidad de habitaciones.
12. Una habitación sólo puede tener un paciente, pero un paciente puede tener varias habitaciones asociadas a él.
13. Los procedimientos de un paciente están almacenados en la historia clínica.
14. Al pagar las deudas, se pagan todos los procedimientos asociados al paciente.
15. Para finalizar un procedimiento se debe estar a paz y salvo en todas las deudas, para esto se puede hacer uso de la funcionalidad de *pagar deudas*.
16. Para dar de alta a un paciente debe haber finalizado todos los procedimientos que tenga pendientes.
17. Dar de alta implica desocupar la habitación o habitaciones que el paciente tenía ocupadas en su estadía en el hospital.

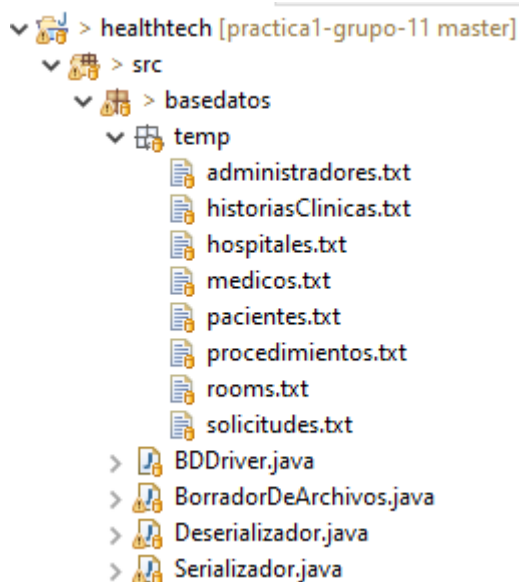
Implementación:

Detalles de implementación e código:

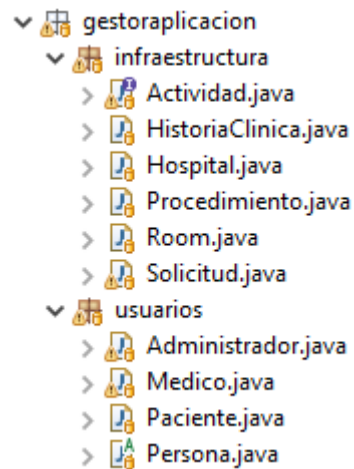
El programa fue implementado en el lenguaje de programación Java 1.8, más conocido como Java 8. Para el desarrollo del código fuente se utilizó el IDE Eclipse. Para el trabajo colaborativo en equipo utilizamos la tecnología de Git para repositorios locales y el sitio web de GitHub para el repositorio remoto.

El programa está dividido en 3 partes:

1. **Una capa de persistencia de datos:** Se encarga de cargar el estado del programa antes de la última vez que se cerrara de forma correcta, y se encarga de guardar el estado de los objetos del programa en ejecución cuando es cerrado de forma correcta. En esta capa se encuentran archivos .txt donde por medio de serialización de objetos (interfaz Serializable de java) se guardan listas de los mismos. Está estructurada de la siguiente forma:

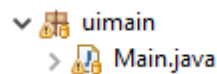


- El paquete temp contiene los archivos .txt donde se van a guardar los streams de bytes de los objetos respectivos.
 - La clase BDDriver va a contener los objetos en listas durante el tiempo de ejecución antes de ser guardados.
 - La clase BorradorDeArchivos es utilizada con fines de desarrollo, no agrega valor a la aplicación.
 - La clase Deserializador es la encargada de cargar los objetos al momento de iniciar la ejecución.
 - La clase Serializador es la encargada de guardar los objetos al momento de cerrar la aplicación correctamente.
2. **Una capa donde está el modelo lógico:** Es la capa que contiene todas las clases relevantes para el modelo de negocio, está descrita con detalle en el diagrama UML que se encuentra después en este mismo documento. Tales clases contienen los métodos y atributos necesarios para cumplir los requisitos de la aplicación.



- La capa lógica está contenida en el paquete gestoraplicacion, que a su vez se divide en dos subpaquetes, infraestructura y usuarios.

3. **La capa de la vista:** En esta capa es donde se encuentra el método main() y desde donde se hacen los llamados a los métodos correspondientes. Aquí se muestra toda la información por pantalla al usuario.



La estructura general del main está soportada por una estructura do-while y sentencias switch-case donde el usuario podrá escoger entre varias opciones:

```
public static void main (String[] args)
{
    do{
        switch(){
            case 1:
                break;
            case 2:
                break;
            case 3:
                break;
            default:
        }
    }
    while();
}
```

Las opciones serán las siguientes:

```

=====
Ingrese 0 para:
-->Mostrar detalles basicos del hospital
=====
Ingrese 1 para:
-->Ingresar Paciente al sistema
=====
Ingrese 2 para:
-->Crear Solicitud de aprobacion de procedimineto para paciente existente
Recuerde que el paciente debe estar registrado en nuestra base de datos
=====
Ingrese 3 para:
-->Aprobar solicitud (Sujeto a disponibilidad de habitaciones)

Si no hay disponibilidad es posible que quiera darle de alta a algun paciente con la opcion 6
=====
Ingrese 4 para:
-->Pagar deudas de paciente.
=====
Ingrese 5 para:
-->Finalizar procedimiento

Recuerde que solo pueden finalizar procedimiento aquellos pacientes que esten a paz y salvo en el respectivo procedimiento,
para ello posiblemente quiera hacer uso de la opcion 4
=====
Ingrese 6 para:
-->Dar de alta

Recuerde que para dar de alta al paciente, debe haber finalizado todos los procedimientos que tenga pendientes,
para ello tal vez quiera hacer uso de la opcion 5.
=====
Ingrese 7 para:
-->Ver detalles de pacientes
=====
Ingrese 8 para:
-->Salir del sistema de forma segura
=====

```

Al iniciar el main lo primero que se hará será invocar el método estático que deserializa todos los objetos:

```

public static void main (String[] args)
{
    Deserializador.deserializar();
}

```

Al finalizar la aplicación, lo que se hará es invocar al método estático salir del sistema para guardar los objetos:

```

private static void salirDelsistema() {
    System.out.println("Vuelva pronto");
    Serializador.serializar();
    System.exit(0);
}

```

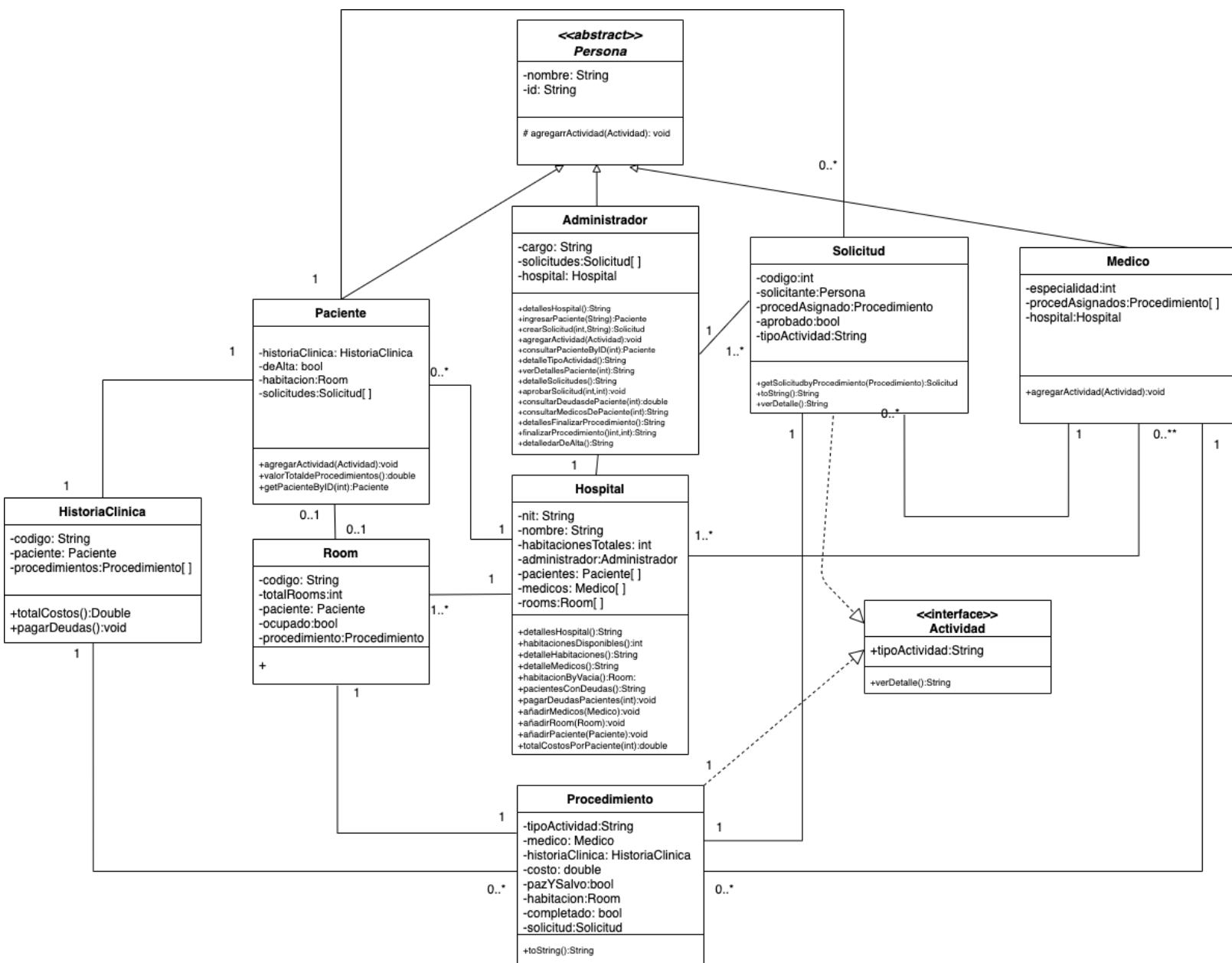
Como información adicional, en el sistema se precargaron las habitaciones, el hospital, el administrador y los médicos, luego el código que hacía esto se comentó:

```

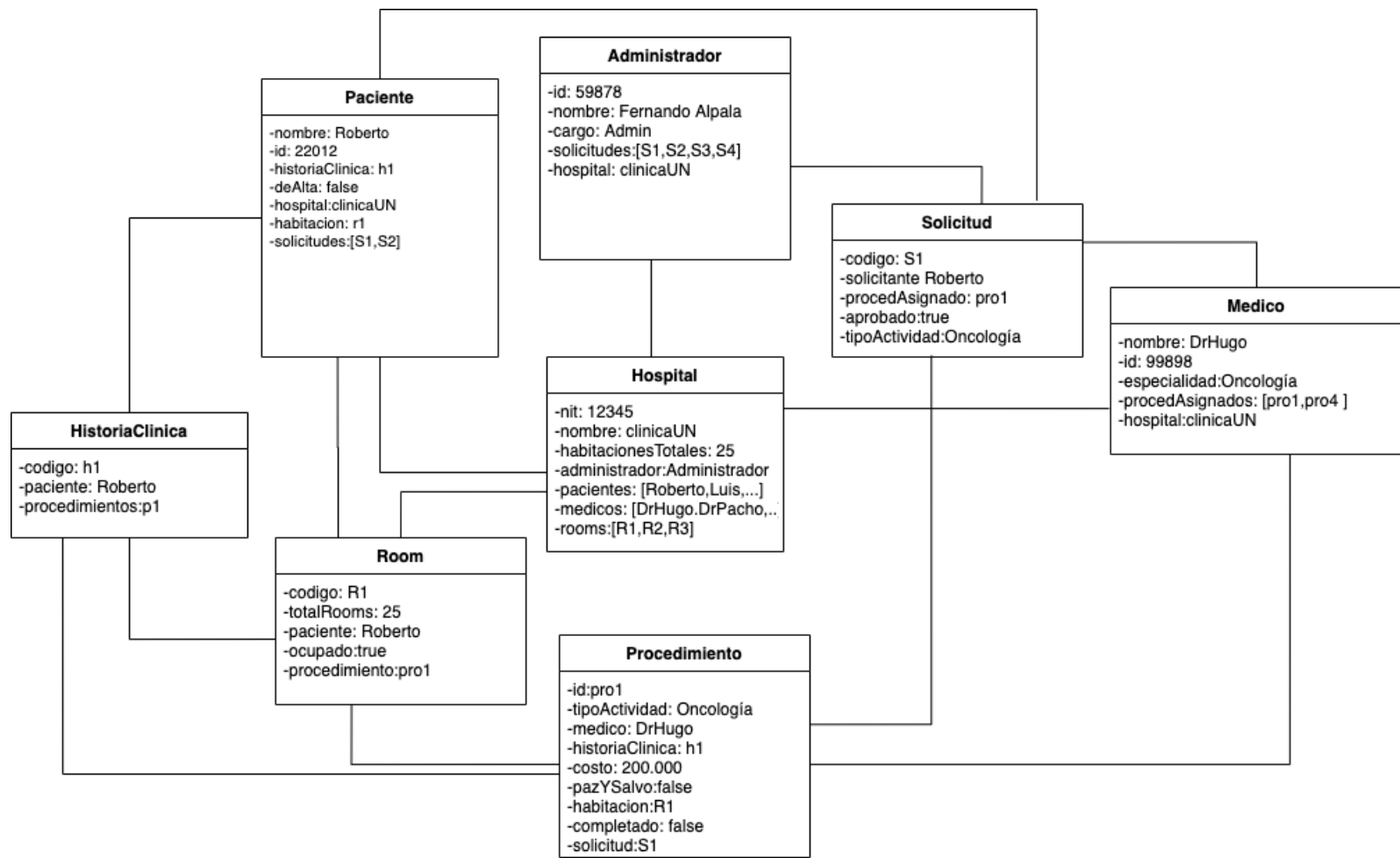
39 //      Room r1 = new Room();
40 //      Room r5 = new Room();
41 //      Room r6 = new Room();
42 //      Room r7 = new Room();
43 //      Room r8 = new Room();
44 //      Room r9 = new Room();
45 //      Room r10 = new Room();
46 //      Room r11 = new Room();
47 //      Room r12 = new Room();
48 //      Room r13 = new Room();
49 //      Room r14 = new Room();
50 //      Room r15 = new Room();
51 //      Room r16 = new Room();
52 //      Room r17 = new Room();
53 //      Room r18 = new Room();
54 //      Room r19 = new Room();
55 //      Room r20 = new Room();
56 //      Room r21 = new Room();
57 //      Room r22 = new Room();
58 //      Room r23 = new Room();
59 //      Room r24 = new Room();
60 //      Room r25 = new Room();
61 //
62 //      Persona m1 = new Medico("Carlos Mejia", "Oncologia");
63 //      Persona m2 = new Medico("Jorge Ramirez", "Pediatría");
64 //      Persona m3 = new Medico("Julian Moreno", "Urologia");
65 //      Persona m4 = new Medico("Jose Gomez", "Oftalmologia");
66 //      Persona m5 = new Medico("Hugo Restrepo", "Cardiologia");
67 //      Persona m6 = new Medico("Alejandro Henao", "Neurologia");
68 //      Persona m7 = new Medico("Bibiana Lopez", "Neftrologia");
69 //      Persona m8 = new Medico("Francisco Diaz", "Dermatologia");
70 //      Persona m9 = new Medico("Claudia Jimenez", "Psiquiatria");
71 //      Persona m10 = new Medico("Blanca Cardona", "Ginecologia");
72 //      Serializador.serializar();
73 //

```

Vista estática del modelo:



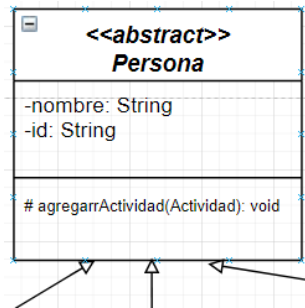
Presentación del Diagrama de Clases



Implementación de características de programación orientada a objetos en el proyecto :

1. Clase abstracta y método abstracto:

Tenemos una clase abstracta en el proyecto que se llama Persona. Se puede encontrar en la ruta src/gestoraplicacion/usuarios/Persona.java.



Tal clase se encarga de abstraer el concepto de persona en el contexto del hospital, donde define atributos de uso común en las clases hijas que son Administrador, Paciente y Medico. La clase fue declarada de la siguiente manera:

```
public abstract class Persona implements Serializable {
```

La clase es abstracta porque contiene un método sin implementación que es parte de un comportamiento común en Administrador, Paciente y Medico. Está declarado de la siguiente manera:

```
public abstract void agregarActividad(Actividad actividad);
```

Este método existe porque cada una de estas clases (Administrador, Paciente y Medico) posee una lista de actividades que les corresponde almacenar: solicitudes y procedimientos, y deben ser agregados a sus respectivas listas.

En la clase Medico se encuentra el método abstracto implementado así:

```
@Override
public void agregarActividad(Actividad actividad) {
    procedAsignados.add((Procedimiento) actividad);
}
```

En la clase Paciente se encuentra el método implementado así:

```
@Override
public void agregarActividad(Actividad actividad) {
    solicitudes.add((Solicitud) actividad);
}
```

En la clase Administrador se encuentra el método implementado así:

```
@Override
public void agregarActividad(Actividad actividad) {
    solicitudes.add((Solicitud) actividad);
}
```

2. Interfaz:

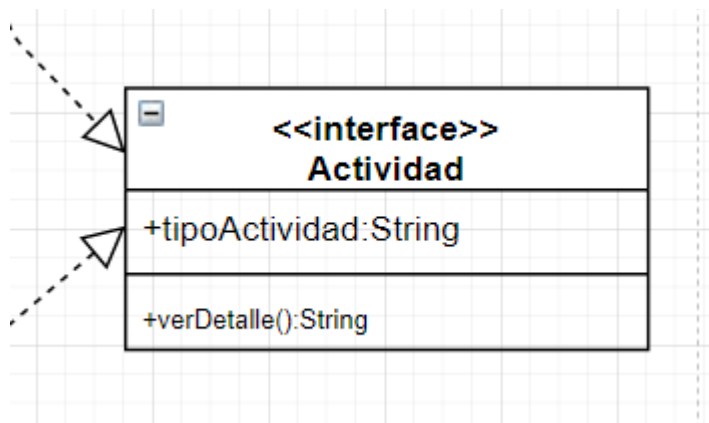
La interfaz se puede encontrar en la ruta src/gestoraplicacion/infraestructura/Actividad.java.

La interfaz que implementamos se llama Actividad, la cual define un nuevo tipo de dato y un comportamiento común a las clases de Solicitud y Procedimiento, las cuales implementan esta interfaz. Se definió de la siguiente manera:

```
public interface Actividad extends Serializable{

    public static final String[] tipoActividad= {"Oncologia", "Pediatria", "Urologia",
        "Oftalmologia", "Cardiologia", "Neurologia", "Nefrologia", "Dermatologia", "Psiquiatria", "Ginecologia"};

    public abstract String verDetalle();
}
```



El método abstracto que contiene es para conocer ciertas características de la Actividad, ya sea de Solicitud o de Procedimiento. El atributo static final tipoActividad es donde se definen los tipos de Actividades que se desarrollan en el hospital, como por ejemplo Cardiología, Oftalmología, Neurología, etc.

3. Herencia:

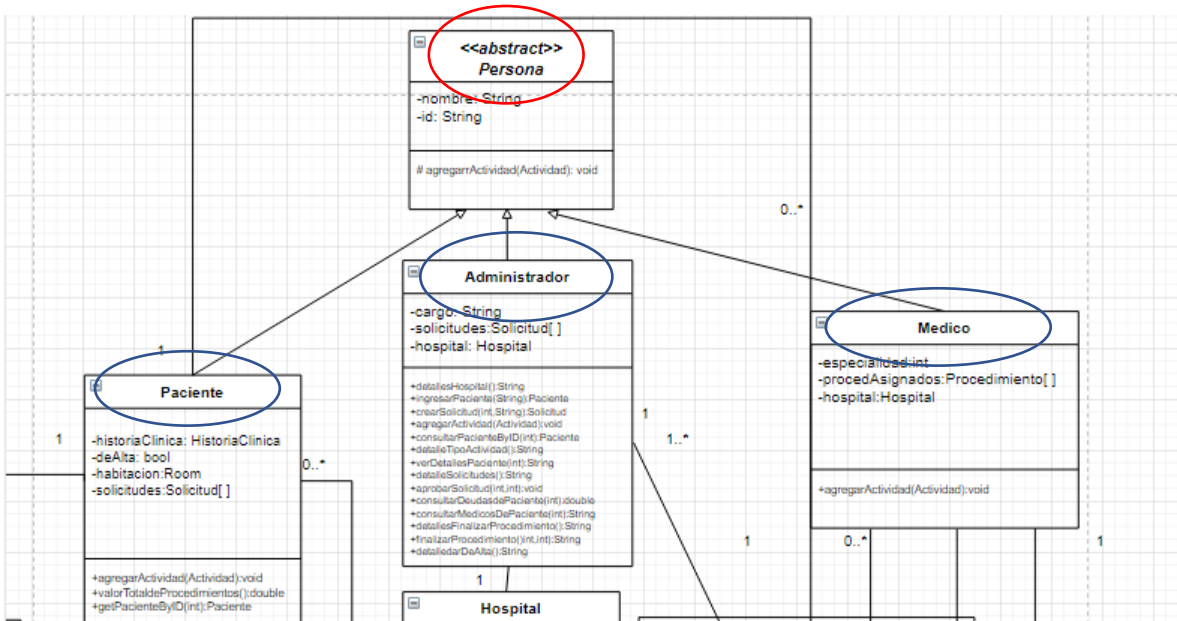
Podemos encontrar herencia en las clases de Administrador, Medico y Paciente, las cuales se encuentran en las rutas src/gestoraplicacion/usuarios/Administrador.java, src/gestoraplicacion/usuarios/Medico.java, y src/gestoraplicacion/usuarios/Paciente.java, respectivamente. Se declararon tales clases de la siguiente forma:

```
public class Administrador extends Persona {
```

```
public class Medico extends Persona{
```

```
public class Paciente extends Persona {
```

En el siguiente diagrama UML se puede ver la relación de herencia entre las clases con su clase padre Persona.



4. Ligadura dinámica:

- **Ligadura dinámica 1:** En la clase Paciente tenemos el método heredado agregarActividad():

```

99 @Override
100 public void agregarActividad(Actividad actividad) {
101     solicitudes.add((Solicitud) actividad);
102 }

```

El cual es heredado desde la clase Persona.

Luego en la clase solicitud está el siguiente método estático de fábrica:

```

95 public static Solicitud crearSolicitud(Persona persona) {
96     Actividad solicitud = new Solicitud(persona);
97     persona.agregarActividad(solicitud);
98     return (Solicitud) solicitud;

```

El cual recibe un parámetro de tipo Persona, y dentro de sí, en la línea 97, se hace un llamado desde este parámetro al método agregarActividad(). Según el parámetro que reciba, se buscará la implementación más específica del método.

Ahora en la clase Administrador tenemos el método de instancia crearSolicitud(), el cual en la línea 82, declara un apuntador de tipo Paciente, llamado pacienteAux. Luego en la línea 97 se hace llamado del método estático crearSolicitud() de la clase Solicitud. Así, el método agregarActividad() buscaría la definición más específica del método, que sería en la clase Paciente.

```
81 public Solicitud crearSolicitud(int idPaciente, String tipoActividad) {
82     Paciente pacienteAux = null;
83
84     // Este for busca si el paciente existe
85     for (Paciente p : hospital.getPacientes()) {
86         if (p.getId() == idPaciente) {
87             pacienteAux = p;
88             break;
89         }
90     }
91
92     if (pacienteAux == null) {
93         System.out.println("El paciente con la identificacion ingresada no existe");
94         return null;
95     } else {
96         Solicitud nuevaSolicitud = Solicitud.crearSolicitud(pacienteAux);
97         nuevaSolicitud.setTipoActividad(tipoActividad);
98         this.agregarActividad(nuevaSolicitud);
99         return nuevaSolicitud;
100     }
101 }
102 }
```

- **Ligadura dinámica 2:** En la clase Administrador tenemos el método de instancia verDetallesPaciente() en la cual, en la línea 180, tenemos un procedimiento de tipo Actividad:

```
165 public String verDetallesPaciente(int id) {
166     Optional<Paciente> pacienteOptional = Paciente.getPacienteById(id);
167     Paciente paciente = pacienteOptional.get();
168     String salida;
169     salida = "Paciente\n";
170     salida += " id: " + paciente.getId() + "\n";
171     if (paciente.getHabitacion() == null) { // No existe habitacion
172         salida += " habitacion actual: No tiene habitacion asignada\n";
173     } else {
174         salida += " habitacion actual: " + paciente.getHabitacion().getCodigo() + "\n";
175     }
176     salida += " paz y salvo: " + (paciente.valorTotaldeProcedimientos() == 0 ? "Verdadero" : "Falso") + "\n";
177     salida += " Estado solicitudes: \n";
178     ArrayList<Solicitud> solicitudes = paciente.getSolicitudes();
179     for (Solicitud solicitud : solicitudes) {
180         Actividad procedimiento = solicitud.getProcedimiento();
181         salida += solicitud.verDetalle();
182         if (procedimiento != null) {
183             salida += procedimiento.verDetalle();
184         } else {
185             salida += " No hay procedimiento asociado\n";
186         }
187         salida += "=====\n";
188     }
189     if (solicitudes.size() == 0) {
190         salida += " No hay solicitudes asociadas a este paciente. :(\n";
191     }
192     return salida;
193 }
```

Luego en la línea 183, a este procedimiento se le hace un llamado a método verDetalle(), el cual es heredado e implementado de la interfaz Actividad. Luego por ligadura dinámica, se buscará la implementación más específica en el contexto del objeto real que está llamándolo, que es de tipo Procedimiento.

5. Atributos de clase y métodos de clase:

- **Atributo de clase:**

Encontramos un atributo de clase, es decir, con el modificador static en la clase Procedimiento:

```
8 public class Procedimiento implements Serializable, Actividad{
9     /*
10     *Atributos
11     */
12     private static final long serialVersionUID = 7092617778020316714L;
13
14     private int id;
15
16     private static int totalProcedimientos=BDDriver.procedimientos.size();
17
```

Este nos permite saber la cantidad actual de procedimientos en el sistema, para poder así asignar códigos únicos en el momento de la creación de los mismos.

- **Método de clase:**

```
93
94 public static Solicitud crearSolicitud(Persona persona) {
95     Actividad solicitud = new Solicitud(persona);
96     persona.agregarActividad(solicitud);
97     return (Solicitud) solicitud;
98 }
```

Este método de clase, con modificador estático, está en la clase Solicitud, más específicamente, es un método estático de fábrica, que devuelve un objeto de tipo Solicitud. Por ser de clase se puede invocar de la siguiente manera: Solicitud.crearSolicitud(). Se crea de esta manera por buenas prácticas, para mantener los constructores de la clase Solicitud limpios y reutilizables, ya que al crear un objeto solicitud con este método estático, también se hace otro llamado a otro método agregarActividad().

6. Uso de constantes:

```
11 public class Hospital implements Serializable {
12
13     /*
14     * Atributos
15     */
16     private static final long serialVersionUID = 8558031300117756825L;
17     private final int nit;
18     private String nombre;
19     private static final int habitacionesTotales = 25;
20     private Administrador administrador;
21     private ArrayList<Paciente> pacientes = BDDriver.getPacientes();
22     private ArrayList<Medico> medicos = BDDriver.medicos;
23     private ArrayList<Room> rooms = BDDriver.rooms;
24
25     /*
```

En la clase Hospital, utilizamos el atributo de tipo static final para poder mostrar al usuario que hay sólo 25 habitaciones en el hospital y que los programadores no utilicen esta constante para cambios variables.

7.Encapsulamiento:

En el proyecto hicimos uso del encapsulamiento de los atributos en todas las clases, mediante el modificador private, y uso de getters y setters para acceder a ellos, los cuales tienen el modificador public. Justo como en las siguientes imágenes:

```
public class Paciente extends Persona {

    /*
     * Atributos
     */
    private static final long serialVersionUID = 2109574389975012203L;
    private HistoriaClinica historiaClinica;
    private boolean deAlta = false;
    private Room habitacion;
    private ArrayList<Solicitud> solicitudes = new ArrayList<Solicitud>();

    ..

    /*
     * Getters y Setters
     */
    public HistoriaClinica getHistoriaClinica() {
        return historiaClinica;
    }

    public void setHistoriaClinica(HistoriaClinica historiaClinica) {
        this.historiaClinica = historiaClinica;
    }

    public boolean isDeAlta() {
        return deAlta;
    }

    public void setDeAlta(boolean deAlta) {
        this.deAlta = deAlta;
    }

    public Room getHabitacion() {
        return habitacion;
    }

    public void setHabitacion(Room habitacion) {
        this.habitacion = habitacion;
    }

    public ArrayList<Solicitud> getSolicitudes() {
        return solicitudes;
    }
}
```

El modificador protected lo utilizamos en los constructores de la clase Persona, ya que como es una clase abstracta, sólo los pueden utilizar sus clases hijas. Lo mostramos en la siguiente imagen:

```

public abstract class Persona implements Serializable {

    /*
     * Atributos
     */
    private static final long serialVersionUID = -735928645591952061L;
    private String nombre;
    private int id;

    /*
     * Constructores
     */
    protected Persona() {
        this.id=BDDriver.totalPersonas()+1;
    }

    protected Persona(String nombre) {
        this();
        this.nombre = nombre;
    }
}

```

8.Sobrecarga de métodos: En la clase Administrador tenemos una sobrecarga de métodos de la siguiente manera:

Un método que recibe como parámetro un String:

```

/*
 * Metodo sobrecargado. Para ver detalle de las solicitudes por filtros:
 * A=aprobado, N=No aprobado.
 */
public void detalleSolicitudes(String param) {
    if (param.equalsIgnoreCase("A")) {
        for (Solicitud elemento : solicitudes) {
            if (elemento.isAprobado() == true) {
                System.out.println(elemento);
            }
        }
    } else if (param.equalsIgnoreCase("N")) {
        for (Solicitud elemento : solicitudes) {
            if (elemento.isAprobado() == false) {
                System.out.println(elemento);
            }
        }
    } else {
        System.out.println("Parametro no valido");
    }
}
}

```

Y otro método que no recibe parámetros:

```

/*
 * Ver detalle solicitud --> Recorre lista de solicitudes de la clase
 * administrador y mostrar detalle de cada solicitud. llama metodo toString de
 * la clase solicitud. Retorna un string con todas las solicitudes con salto de
 * linea.
 */
public String detalleSolicitudes() {
    String detalle = "";
    for (Solicitud elemento : solicitudes) {
        if (elemento.isAprobado() == false) {
            detalle += elemento + "\n";
        }
    }
    return detalle;
}
}

```

Como ambos tienen el mismo nombre pero su lista de parámetros difiere, hay una correcta sobrecarga de métodos, ya que la firma de ambos métodos difiere.

9. Sobrecarga de constructores:

Ejemplo 1: en src/gestoraplicacion/infraestructura/Solicitud.java

```
24-  /*
25-   * Constructores
26-   */
27-  public Solicitud() {
28-      cantidadTotal = BDDriver.solicitudes.size()+1;
29-      this.codigo = cantidadTotal;
30-  }
31-
32-  private Solicitud(Persona solicitante) {
33-      this();
34-      this.solicitante = solicitante;
35-      BDDriver.solicitudes.add(this);
36-  }
37-
```

Ejemplo 2: en src/gestoraplicacion/usuarios/Persona.java

```
22-  protected Persona() {
23-      this.id = BDDriver.totalPersonas()+1;
24-  }
25-
26-  protected Persona(String nombre) {
27-      this();
28-      this.nombre = nombre;
29-  }
30-
```

10. Manejo de referencias this y this() y super():

- **Ejemplo 1 de this:** En src/gestoraplicacion/infraestructura/Hospital.java podemos ver que se usa en la línea 31 el this para desambiguar entre la variable parámetro y la variable atributo.

```
25-  /*
26-   * Constructores
27-   */
28-
29-  public Hospital(String nombre) {
30-      nit = 123456789;
31-      this.nombre = nombre;
32-      BDDriver.hospitales.add(this);
33-  }
34-  }
```

- **Ejemplo 2 this:** En src/gestoraplicacion/infraestructura/Room.java podemos ver que se utiliza la palabra clave this en la línea 24 para hacer referencia al mismo objeto que se crea en el constructor y agregarlo a la base de datos:

```

20-  /*
21-   * Constructores
22-   */
23-  public Room() {
24-      BDDriver.rooms.add(this);
25-      totalRooms++;
26-      this.codigo=totalRooms;
27-  }

```

- **Ejemplo this():** En la clase Persona tenemos el uso del this() en la línea 27 para que la ejecución del código vaya al constructor sin argumentos, el cuál ejecuta una tarea de darle un código único al objeto que está siendo creado.

```

22-  protected Persona() {
23-      this.id=BDDriver.totalPersonas()+1;
24-  }
25-
26-  protected Persona(String nombre) {
27-      this();
28-      this.nombre = nombre;
29-  }

```

- **Ejemplo super() y this():**

La clase Paciente en su constructor, lo primero que hace es llamar al constructor de su papá, que es Persona, en la línea 29:

```

23
24-  /*
25-   * Constructores
26-   */
27-
28-  public Paciente(String nombre) {
29-      super(nombre);
30-
31-      BDDriver.agregarPaciente(this);
32-  }
33-

```

A su vez, cuándo llega a la clase Persona, éste, mediante un this(), hace llamado al constructor vacío:

```

22-  protected Persona() {
23-      this.id=BDDriver.totalPersonas()+1;
24-  }
25-
26-  protected Persona(String nombre) {
27-      this();
28-      this.nombre = nombre;
29-  }
30-
31-

```

A blue curved arrow originates from the `this();` line in the `Persona(String nombre)` constructor and points to the `protected Persona()` constructor, illustrating the recursive call to the no-argument constructor.


Manual de Uso:

Nota: Las funcionalidades interesantes son las que corresponden a las opciones de menú de la 2 a la 6.

Los siguientes pasos le permitirán ejecutar de manera correcta el aplicativo.

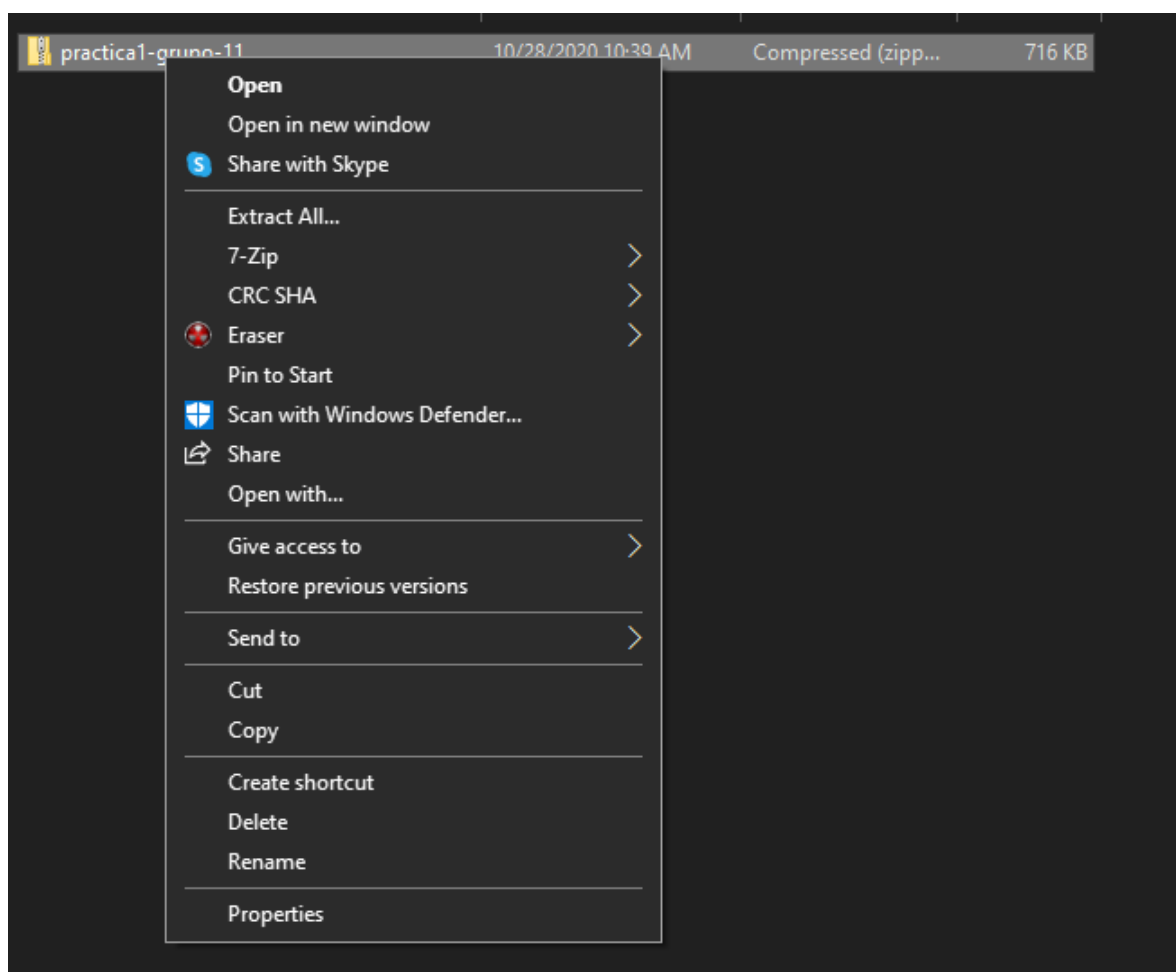
1. Ejecutar nuestro software HealhTech.

Se le entregara al cliente un archivo .zip llamado **practica1-grupo11**

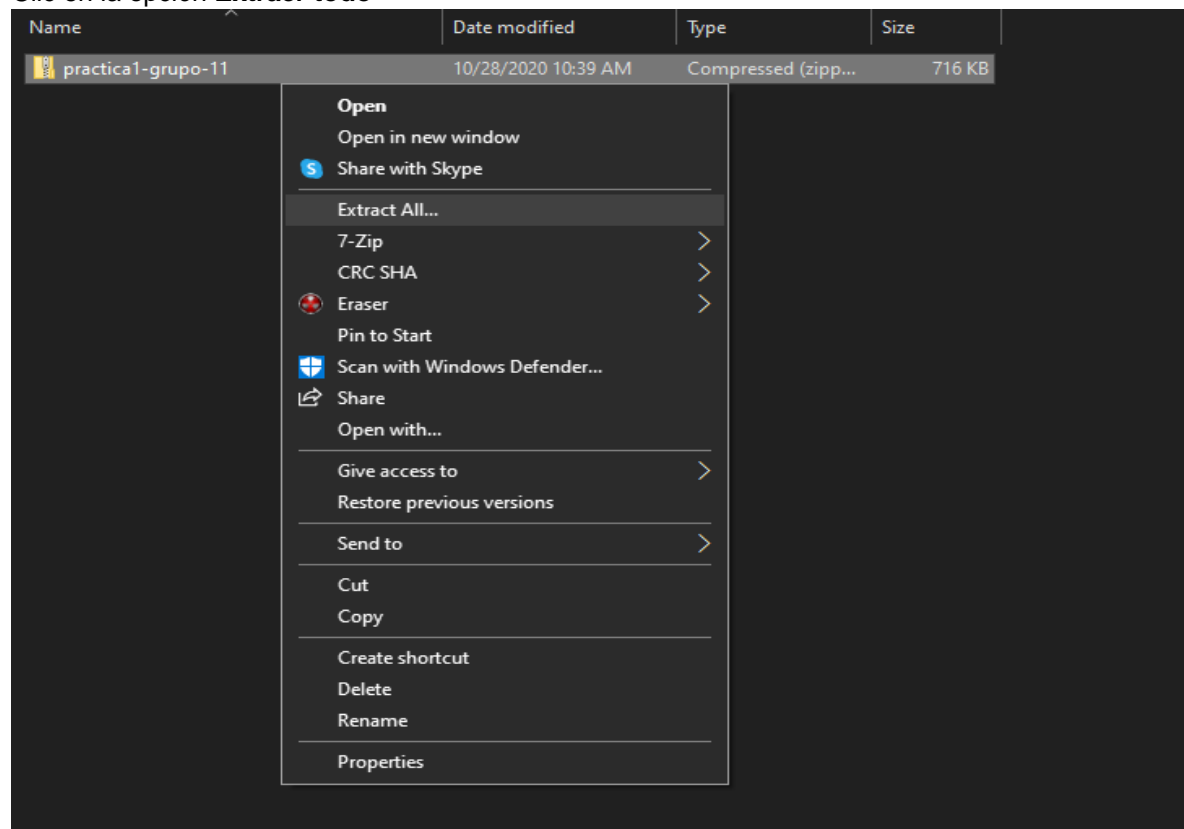
Name	Date modified	Type	Size
 practica1-grupo-11	10/28/2020 10:39 AM	Compressed (zipp...	716 KB

Dicho archivo.zip debe ser extraído.

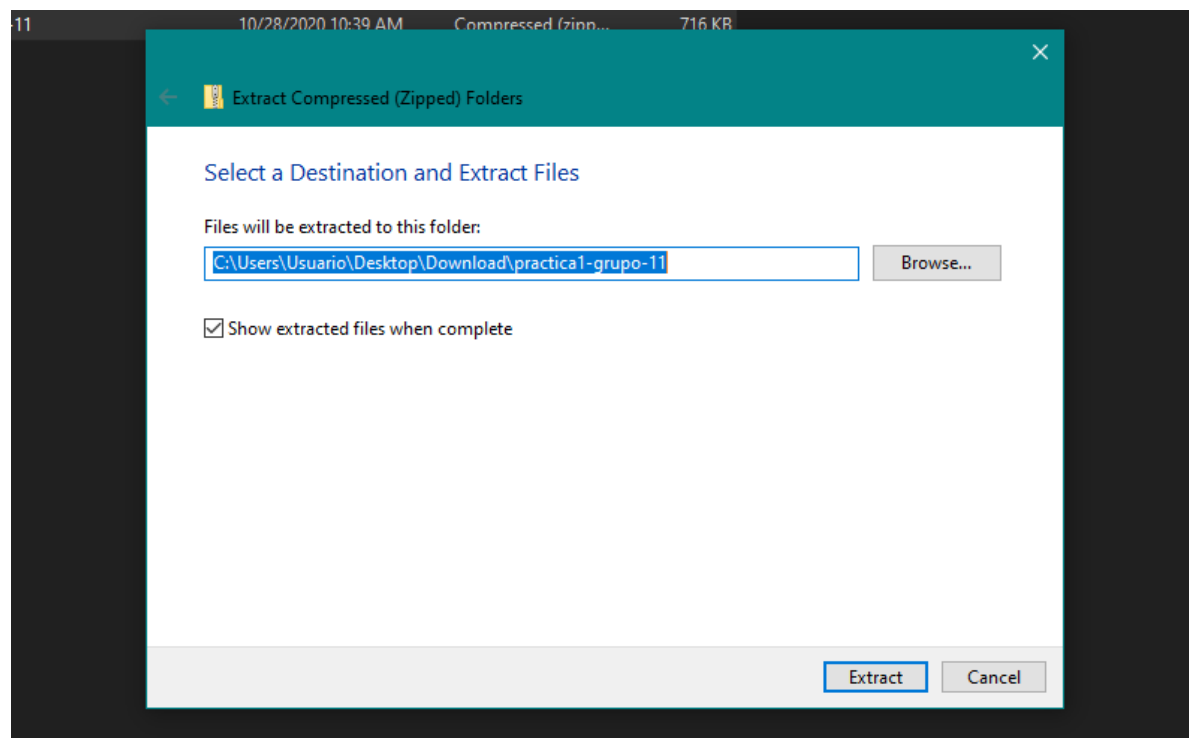
Para extraer el archivo basta con dale clic derecho al archivo **practical1-grupo11**.



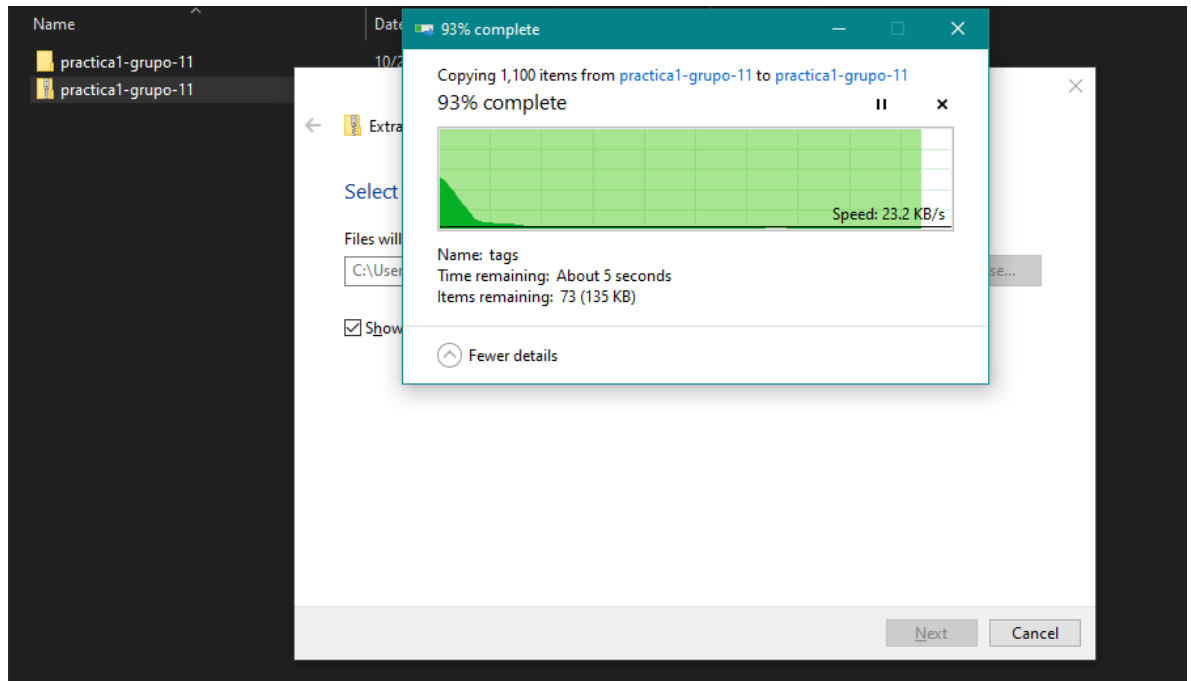
Clic en la opción **Extraer todo**



Se le abrirá la siguiente ventana



Dándole en la opción **extraer** empezará el proceso de extracción del .zip. Hay que esperar unos 15 segundos aproximadamente.



En el directorio actual tendremos **dos** archivos:

- Un archivo directorio
- Un archivo .zip

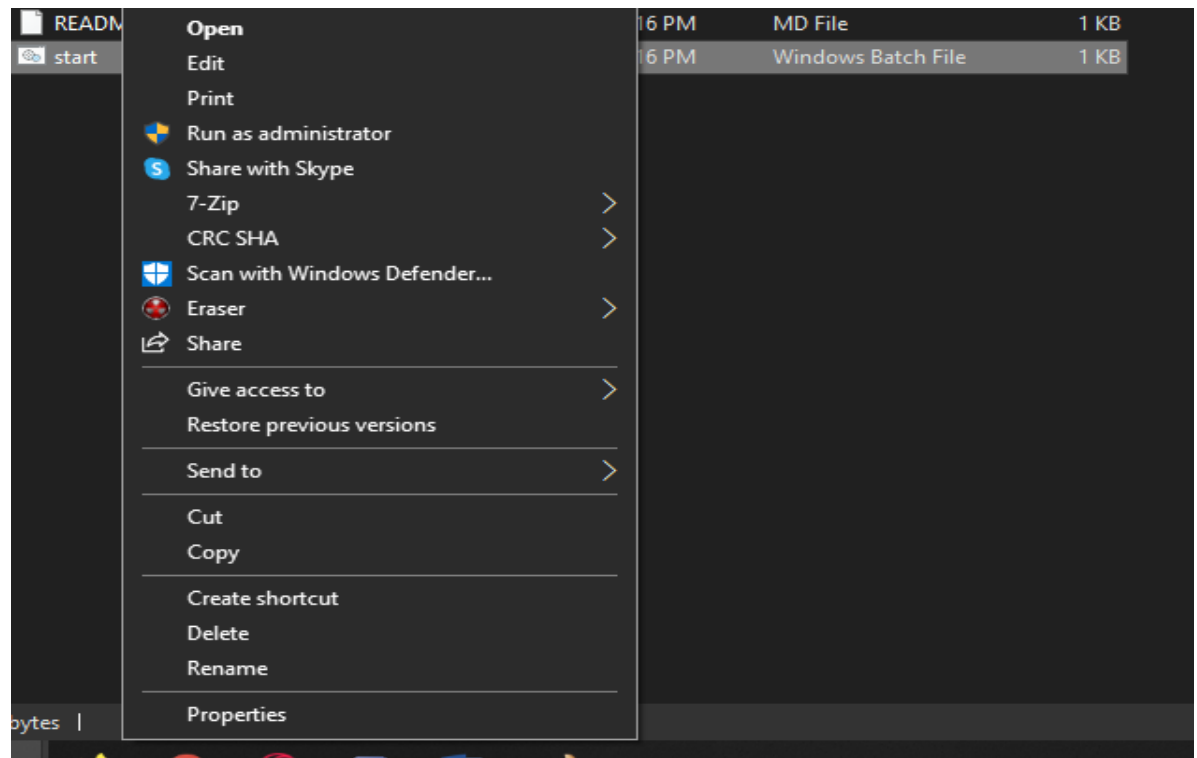
Name	Date modified	Type	Size
practica1-grupo-11	10/28/2020 7:16 PM	File folder	
practica1-grupo-11	10/28/2020 10:39 AM	Compressed (zipp...	716 KB

Abriendo el **archivo directorio** practica1-grupo11
Tendremos los siguiente:

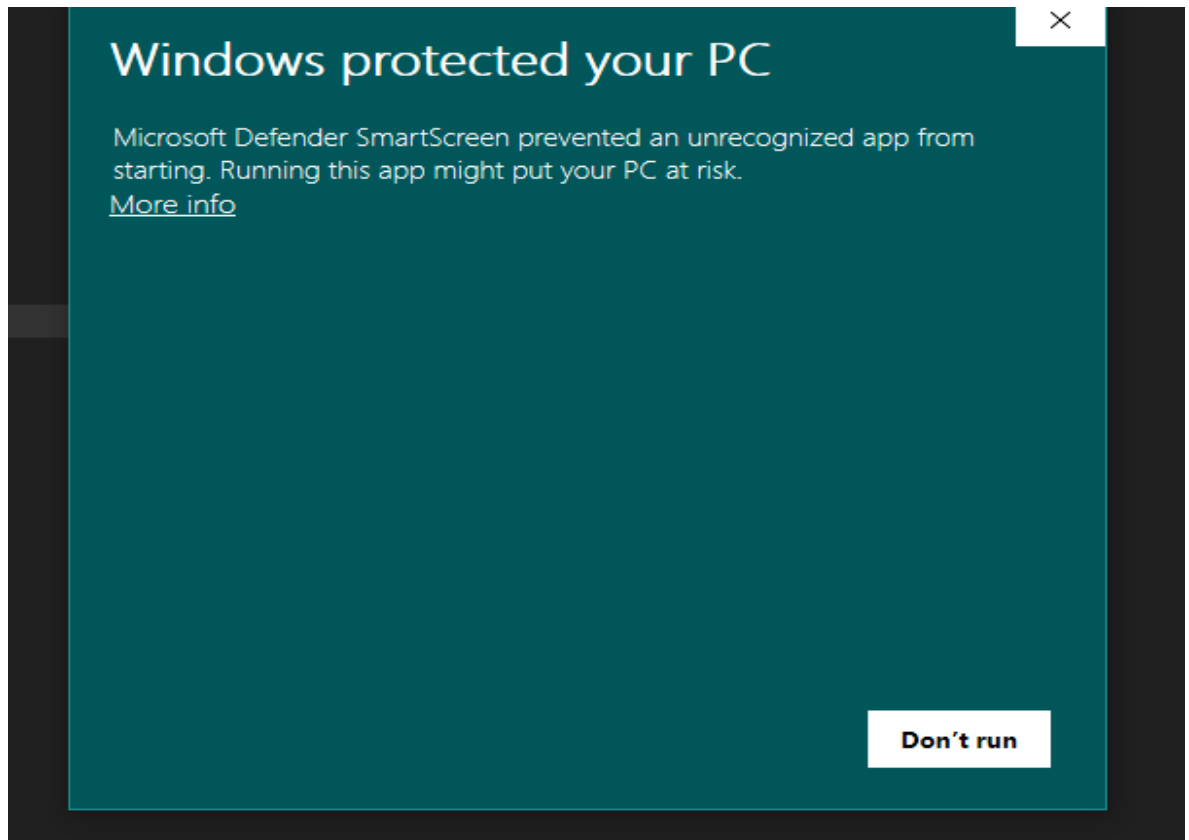
> Download > practica1-grupo-11 > practica1-grupo-11 >				
Name	Date modified	Type	Size	
bin	10/28/2020 7:16 PM	File folder		
src	10/28/2020 7:16 PM	File folder		
.classpath	10/28/2020 7:16 PM	CLASSPATH File	1 KB	
.gitignore	10/28/2020 7:16 PM	Text Document	1 KB	
.project	10/28/2020 7:16 PM	PROJECT File	1 KB	
HealthTech	10/28/2020 7:16 PM	Executable Jar File	42 KB	
MenuDeConsola	10/28/2020 7:16 PM	Text Document	5 KB	
README.md	10/28/2020 7:16 PM	MD File	1 KB	
start	10/28/2020 7:16 PM	Windows Batch File	1 KB	

Al cliente solo le debe importar el archivo llamado **start**

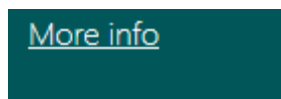
Dándole clic **derecho** luego **abrir**



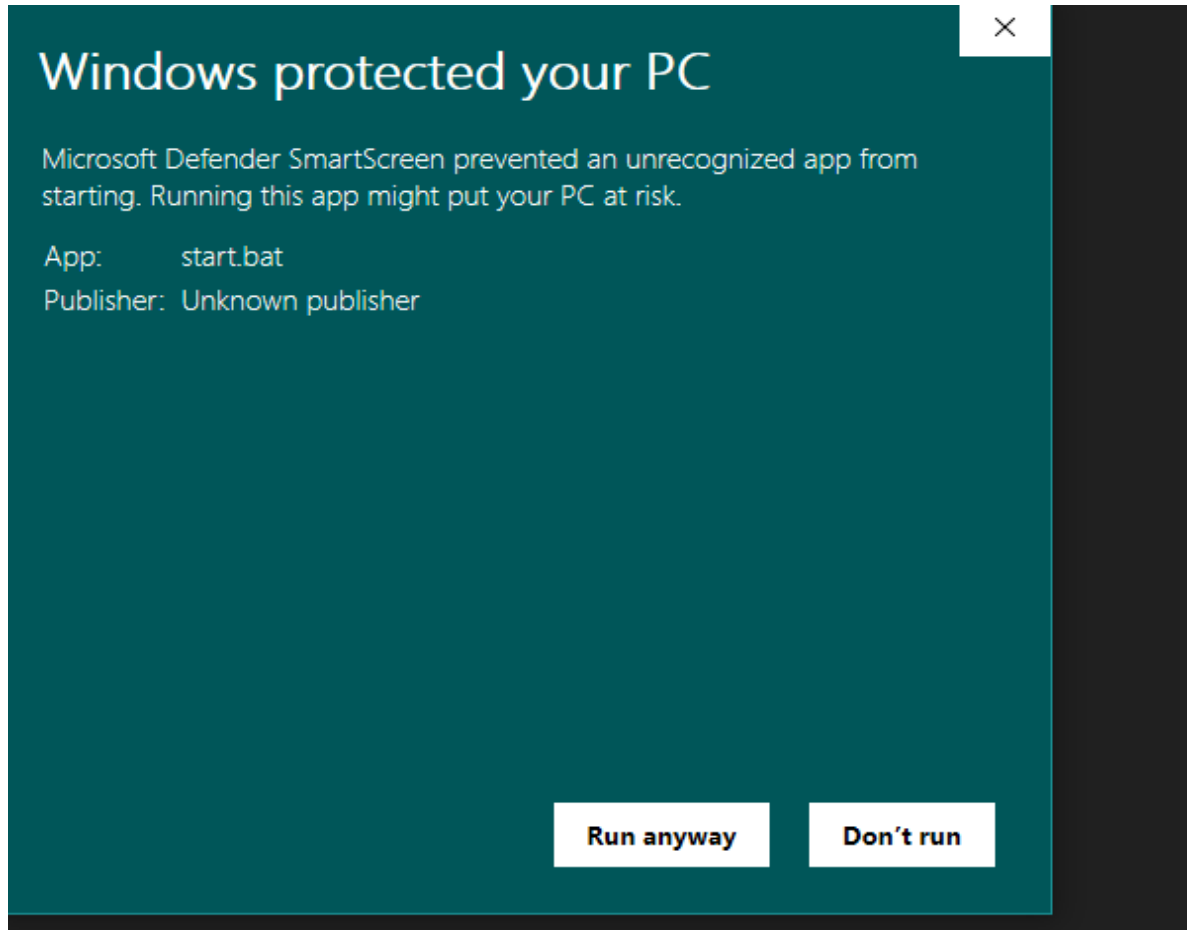
El **sistema operativo** te mostrara la siguiente ventana:



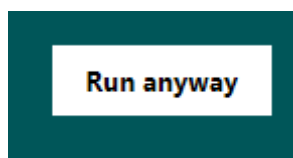
Dándole clic **derecho** en más información.



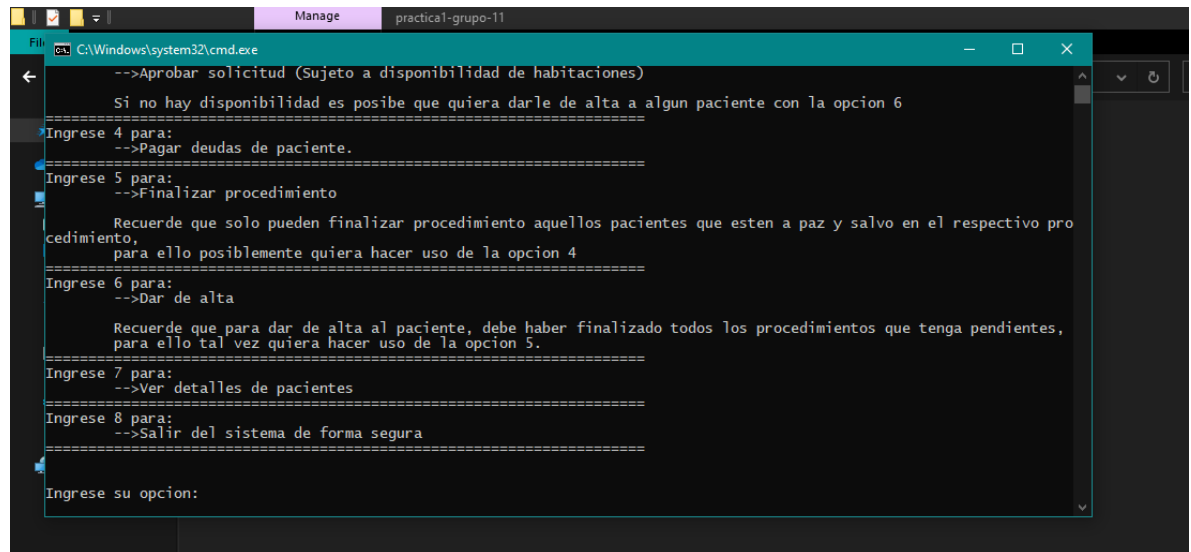
Tendrás la siguiente ventana:



Y luego dándole clic derecho **ejecutar de igual manera**



Estará el aplicativo **HealhTech** listo para usarse



```
File Edit View Help C:\Windows\system32\cmd.exe Manage practica1-grupo-11
-->Aprobar solicitud (Sujeto a disponibilidad de habitaciones)
Si no hay disponibilidad es posible que quiera darle de alta a algun paciente con la opcion 6
=====
Ingrese 4 para:
-->Pagar deudas de paciente.
=====
Ingrese 5 para:
-->Finalizar procedimiento
Recuerde que solo pueden finalizar procedimiento aquellos pacientes que esten a paz y salvo en el respectivo procedimiento,
para ello posiblemente quiera hacer uso de la opcion 4
=====
Ingrese 6 para:
-->Dar de alta
Recuerde que para dar de alta al paciente, debe haber finalizado todos los procedimientos que tenga pendientes,
para ello tal vez quiera hacer uso de la opcion 5.
=====
Ingrese 7 para:
-->Ver detalles de pacientes
=====
Ingrese 8 para:
-->Salir del sistema de forma segura
=====
Ingrese su opcion:
```

Nota importante: Tenga en cuenta que el programa sólo va a ejecutar si el .jar y la carpeta src están en el mismo directorio, esto es porque

2. Abrir el aplicativo:

-Después de abrir el aplicativo tendrá un menú igual a este.

```
Hola Administrador Jaime Alberto Guzman Luna!.
Bienvenido al Software HealthTech para Administrar el Hospital Universitario

Aqui esta su menu:

MENU:

=====
Ingrese 0 para:
-->Mostrar detalles basicos del hospital
=====
Ingrese 1 para:
-->Ingresar Paciente al sistema
=====
Ingrese 2 para:
-->Crear Solicitud de aprobacion de procedimiento para paciente existente
Recuerde que el paciente debe estar registrado en nuestra base de datos
=====
Ingrese 3 para:
-->Aprobar solicitud (Sujeto a disponibilidad de habitaciones)

Si no hay disponibilidad es posible que quiera darle de alta a algun paciente con la opcion 6
=====
Ingrese 4 para:
-->Pagar deudas de paciente.
=====
```

```

Ingrese 5 para:
-->Finalizar procedimiento

Recuerde que solo pueden finalizar procedimiento aquellos pacientes que esten a paz y salvo en el respectivo procedimiento,
para ello posiblemente quiera hacer uso de la opcion 4
=====
Ingrese 6 para:
-->Dar de alta

Recuerde que para dar de alta al paciente, debe haber finalizado todos los procedimientos que tenga pendientes,
para ello tal vez quiera hacer uso de la opcion 5.
=====
Ingrese 7 para:
-->Ver detalles de pacientes
=====
Ingrese 8 para:
-->Salir del sistema de forma segura
=====

```

Ingrese su opcion:



Dicho menú tendrá diferentes funcionalidades que van desde la **0** hasta la **8**. A lo largo de este manual nos iremos familiarizando con cada una de las funcionalidades.

3. Detalles Básicos del Hospital:

- Lo más seguro es que quieras saber detalles del hospital. Ingresando la **opción 0** en nuestro menú:

Ingrese su opcion:

0

Se le desplegará la siguiente información de interés.

Nombre del Hospital, Nombre del Administrador, Habitaciones totales, Habitaciones Desocupadas, Cantidad de pacientes en el Sistema y los médicos que están registrado en la base de datos con su id, nombre y especialidad.

```

Nombre del hospital: Hospital Universitario

Nombre del administrador: Jaime Alberto Guzman Luna

Habitaciones totales: 25

Habitaciones desocupadas: 25

Cantidad de pacientes en el sistema: 0

Detalle de medicos:
Id      Nombre      Especialidad
=====
2  Carlos Mejia      Oncologia
3  Jorge Ramirez     Pediatria
4  Julian Moreno     Urologia
5  Jose Gomez        Oftalmologia
6  Hugo Restrepo     Cardiologia
7  Alejandro Henao   Neurologia
8  Bibiana Lopez     Nefrologia
9  Francisco Diaz    Dermatologia
10 Claudia Jimenez   Psiquiatria
11 Blanca Cardona   Ginecologia

```



Si dese volver al menu principal ingrese 1 , si desea cerrar el sistema de forma segura ingrese 9:

Para regresar al menú principal basta con **ingresar 1**.

4. Ingresar pacientes al Sistema:

-Ingresando la **opción 1**. Se le pedirá el nombre del paciente.

```
Ingrese su opcion:
```

```
1
```

```
Ingrese el nombre del paciente que desea ingresar:
```

```
Hector Lopez
```

```
El paciente que acaba de ingresar se llama Hector Lopez y tiene el id 12
```

```
Su historia clinica tiene codigo: 12
```

```
Si desea volver al menu principal ingrese 1 , si desea cerrar el sistema de forma segura ingrese 9:
```

Al ingresar un paciente, se crea un objeto paciente, a la vez que se le asigna un id único de forma automática y se le crea una historia clínica también de forma automática.

5. Crear solicitud de aprobación de procedimiento:

Luego de ingresar un paciente, el usuario tiene la opción de crear la solicitud de aprobación procedimiento para cualquier paciente que esté registrado en la base de datos del hospital. Dicha opción permite que el paciente **PIDA** ser atendido, es decir, que pida que se le asigne un médico especialista en la opción ingresada (Cardiología, Oncología, etc.), se le inicie un procedimiento y se le asigne una cama.

```
Ingrese su opcion:
```

```
2
```

```
Aqui esta la lista de pacientes ingresados en el sistema para que elija uno de sus IDs:
```

NOMBRE	ID
Hector Lopez	12
Samuel gomez	13
Valentina Mora	14
Camilo Osorio	15
Alejandro Valencia	16

```
Aqui esta la lista de tipos de procedimiento posible, también debe elegir uno:
```

```
TIPO DE PROCEDIMEINTOS POSIBLES
```

```
Oncologia  
Pediatria  
Urologia  
Oftalmologia  
Cardiologia  
Neurologia  
Nefrologia  
Dermatologia  
Psiquiatria  
Ginecologia
```

```
Ingrese el id del paciente al cual le desea crear una solicitud: |
```

-Ingresando la **opción 2** del menú principal:
Podrá visualizar los pacientes con su id y los tipos de procedimientos que se les pueden aplicar.

Ingresando el id y luego el tipo de procedimiento se creará la solicitud de aprobación y esta estará en un estado de espera hasta ser aprobada por el administrador.

```
Ingrese el id del paciente al cual le desea crear una solicitud:
12
Ingrese el tipo de solicitud exactamente como se le mostro en la tabla:
Urologia
Acaba de crear una solicitud para el paciente:
Hector Lopez
Con el id:
12
El codigo de la solicitud es:
1
Si desea volver al menu principal ingrese 1 , si desea cerrar el sistema de forma segura ingrese 9:
```

6. Aprobar solicitud de procedimiento: -Ingresando la **opcion3**.

Podrá visualizar todas solicitudes que se han realizado y algunos datos sobre la misma.

```
Ingrese su opcion:
3

Aqui esta la lista de las solicitudes sin aprobar, elija un codigo:

Codigo de solicitud: 1
Solicitante: Hector Lopez
Tipo Procedimiento: Urologia
Aprobado: false

Ingrese el codigo de la solicitud que desea aprobar:
```

Ingresando el código de la solicitud y el costo del procedimiento. Automáticamente se seleccionará un médico especialista en el área, una habitación disponible, será aprobada la solicitud, iniciado el procedimiento y éste último agregado a la historia clínica. El usuario también deberá especificar el valor del procedimiento a realizar.

Ingrese el codigo de la solicitud que desea aprobar:

1

Ingrese el Costo del procedimiento a realizar:

300

Se ha aprobado con exito la solicitud para Urologia

El medico asignado es: Julian Moreno de Urologia

El numero de habitacion asignada es: 1

Luego de realizar el procedimiento al paciente se deberá pagar dicho procedimiento.

7. Pagar deudas de pacientes:

-Ingresando la **opción 4**.

Se desplegará una tabla con los pacientes deudores. En la columna DEBE se le mostrará la suma de todos los costos de los procedimientos que tiene iniciados el paciente.

Ingrese su opcion:

4

Aquí esta la lista de pacientes con deudas:

PACIENTE	ID	DEBE
=====	=====	=====
Hector Lopez	12	300.0

Ingresando el id del paciente procederá a pagar la deuda con el hospital

Ingrese el id del paciente que quiere pagar:

12

PAZ Y SALVO TOTAL

Luego de que el paciente este a paz y salvo con el hospital este podrá finalizar procedimiento.

8. Finalizar procedimiento:

- Ingresando la **opción 5**.

Se desplegará una tabla con los pacientes que están a paz con el hospital y se les puede finalizar el procedimiento.

Ingrese su opcion:

5

Esta es la lista de pacientes que pueden finalizar procedimientos asociados:

NOMBRE	ID	PROCEDIMIENTO	ID
=====	=====	=====	=====
Hector Lopez	12	Urologia	1

Ingrese el id del paciente y del procedimiento que quiere finalizar:

Ingresando el id del paciente y el id del procedimiento finalizara dicho procedimiento.

Ingrese id Paciente:

12

Ingrese id Procedimiento:

1

Se finalizo el procedimiento con el id: 1
del paciente: Hector Lopez Exitosamente

Luego de finalizar el procedimiento se podrá dar de alta al paciente. Lo cual implica desalojar la habitación en la que está.

9. Dar de alta:

-Ingresando la **opción 6**.

Se desplegará una tabla con el nombre de los pacientes y el id que tienen procedimientos asociados finalizados

Ingrese su opcion:

6

Esta es la lista de pacientes que pueden ser dados de Alta

NOMBRE	ID
=====	=====
Hector Lopez	12

Ingresando el id del paciente podrá dar de alta a dicho paciente.

Ingrese el id del paciente que se quiere dar de alta:

Ingrese id Paciente:

12

Se dio de alta al paciente: Hector Lopez con id: 12 de forma exitosa

10. Mostrar detalles de pacientes:

-Ingresando la **opción 7**.

Se desplegará una lista con los pacientes registrados en el hospital a los cuales podrás ver detalles de interés

```
Ingrese su opcion:
7
Esta es la lista de pacientes:
```

NOMBRE	ID
=====	=====
samuel rojas	12
manuel gomez	13
camilo osorio	14
raul Morales	15
Hector lopez	16

Ingresando el ID del paciente podrá ver detalles de interés

```
Ingrese el ID de paciente de ver detalles:
```

```
12
Paciente
  id: 12
  habitacion actual: No tiene habitacion asignada
  paz y salvo: Verdadero
  Estado solicitudes:
    Tipo: Oncologia
   Codigo: 1
    Aprobado: No
    No hay procedimiento asociado
=====
```

```
Si desea volver al menu principal ingrese 1 , si desea cerrar el sistema de forma segura ingrese 8:
```

-Ingresando la **opción 1** podrá ir al menú principal

-Ingresando la **opción 8** podrá cerrar el sistema de forma segura, es decir, se guardará el estado actual del sistema y los cambios que haya hecho.

11. Cerrar aplicativo:

-Ingresando la **opción 8** podemos finalizar de forma correcta la ejecución del aplicativo **HealthTech**.

```
Ingrese su opcion:
8
Vuelva pronto
```