Para modelar nuestro problema, el que concierne a un hospital, se intentó ser lo realista posible, eso sí, considerando las limitaciones temporales -tiempo de entrega- y de recursos, sin embargo ello no quiere decir que se intentó abordar el problema de una forma escueta, no, simplemente fue un aspecto que siempre tuvimos en mente.

Nuestro objetivo fue el de implementar un sistema informático para un hospital, llamado *Hospital Universidad de Antioquia*, **el cual le permitiera al usuario efectuar todo lo que estuviera relacionado con la atención de pacientes**, para ello consideramos pertinente que con el sistema se pudiesen realizar las siguientes acciones:

- Crear nuevos pacientes.
- Buscar si una persona es o no un paciente del hospital.
- Actualizar la información de un paciente.
- Eliminar la información de una persona que era paciente del hospital, pero que ha decidido cambiar de centro de atención.

Ahora bien, con la intención de complementar el sistema informático que desarrollamos, vimos que era adecuado que la información los médicos del hospital pudiese ser operada de manera análoga a la de los pacientes, es por ello que respecto a los médicos nuestro sistema le permite al usuario:

- Crear nuevos médicos.
- Buscar si una persona es o no un médico del hospital.
- Actualizar la información de un médico.
- Eliminar la información de una persona que era médico del hospital, pero que ha decidido cambiar de lugar de trabajo.

Nótese aquí que lo anterior fue hecho a propósito, esto es, con la intención de explotar una de las características de la Programación Orientada a Objetos (POO), la **herencia**. Así, tendremos un "ente abstracto" que podrá "efectuar" las acciones anteriormente señaladas y dos entes, también abstractos, que serán "capaces" de replicar tales acciones; dicho en términos precisos, contaremos con **una clase padre** y por el momento con **dos clases hijas**.

Lo anterior hace referencia a las únicas personas que harán parte de nuestro modelo, pero la forma en la que implementamos la clase padre permite que nuestra solución se pueda extender para anexar otro tipo de personas, esto es, lo hicimos con la intención de resaltar otra de las cualidades de la POO, es decir, con el objetivo de que nuestra solución sea **escalable**.

En este punto pasamos a analizar la forma en la que interactúan las personas que hacen parte de nuestro modelo, esto es, la forma en la que se relacionan los médicos y los pacientes. Aquí pudimos identificar, haciendo un paralelo con lo que ocurre en la vida real, dos aspectos fundamentales en la interacción médico-paciente, lo que son las *citas* y lo que son las *historias clínicas*, ambas modeladas en nuestra solución.

Consideramos idóneo para nuestro modelo que el usuario, por medio de la clase que manejará todo lo que tiene que ver con citas médicas, estuviera en condiciones de realizar las acciones:

- Crear nuevos citas.
- Buscar si una persona tiene una cita programada con uno de los médicos del hospital.
- Actualizar la información de una cita.
- Eliminar la cita que uno de los pacientes tiene programada con uno de los médicos del hospital.

Finalmente, pero no menos importante, no sería muy realista de nuestra parte que el modelo no le permitiera el médico conocer la historia clínica de un paciente, es por ello que decidimos implementar una clase justo con ese fin. Al usuario se le permitirá interactuar con las historias clínicas para efectuar solamente lo siguiente:

- Crear nuevas historias clínicas.
- Acceder a la historia clínica de un paciente.
- Actualizar la historia clínica de un paciente.
- Eliminar una historia clínica de un paciente.

Al ver entonces las anteriores acciones, pudimos corroborar que era adecuado que tanto la clase que modelara las citas como aquella que modelara las historias clínicas, se comportaran de manera análoga a las clases ya establecidas, esto es, nos dimos cuenta de que estas nuevas clases (citas e historia clínica) debían ser también **dos clases hijas** de la misma clase padre.

Es así que iniciamos la codificación de nuestra solución. En un principio y como se dijo previamente, modelamos la clase padre, esto es, la clase que sirve de referencia para las instancias de cada objeto, a esta clase la llamamos **generic_database** y en ella están implementadas las acciones que se habían indicado previamente, estas acciones, que de manera técnica se reconocen como **métodos**, son:

- create(): Sirve para crear una instancia del objeto, que puede ser de tipo *doctor, patient, citas, historia.*
- read(): Sirve para buscar la información referente a un registro, en caso de que exista en la base de datos, se muestra la información que pertenece a ese registro.
- update(): Sirve para actualizar uno o varios campos de un registro de la base de datos.
- delete(): Elimina la información de un registro de la base de datos.

Por lo tanto cada uno de los objetos de nuestro modelo, de manera más precisa, cada una de sus instancias, ya sea de la clase *doctor,* de la clase *patient,* de la clase *citas* o de la clase *historia,* podrá efectuar tales acciones.

A continuación esbozamos cómo funciona nuestro modelo, esto lo hacemos en abstracto.

- Para que una persona pueda ser atendida por uno de los médicos del hospital, ella debe estar en la categoría de paciente, para ello el sistema verifica si tal persona hace parte de la base de datos de pacientes, en caso de que no estuviera allí, se debe crear un registro, en la base de datos de pacientes,con la información de él.

- Para poder agendar una cita (**solo a los pacientes se le agendan citas**), el paciente no debe tener programadas otras citas, pues en caso de que ello ocurriera, el sistema le impide programar una nueva cita. Así las cosas, **un paciente sólo podrá tener activa una cita**.

- Para **buscar la información** de un paciente o de un médico, el campo que el sistema le exige al usuario es su **número de identificación**, ya sea cédula de ciudadanía o tarjeta de identidad.

- Para **eliminar la información** que se tenga de un paciente o de un médico, al igual que con la búsqueda, el sistema le exige al usuario el **número de identificación** de la persona y de igual forma, tal número puede ser la cédula de ciudadanía o la tarjeta de identidad.

- Puesto que un paciente solo podrá tener una cita activa, las citas serán manipuladas dentro del sistema por medio del **número de identificación del paciente**, luego las operaciones de **buscar**, de **actualizar** y de **eliminar una cita**, necesitarán conocer tal número.

- Al igual que ocurre con las citas, a un paciente se le asignará una única historia clínica y es por ellos que las operaciones de **buscar**, de **actualizar** y de **eliminar una historia clínica**, le exigirán al usuario que ingrese el **número de identificación del paciente**.