UNIVERSIDAD DE BURGOS

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

Diseño de un subsistema de persistencia para una aplicación de Agenda

Diseño y mantenimiento de software

Alumnos Oscar Fernández Armengol
Daniel Puente Gabarri
Jorge Zamora Marques
Javier Martínez Riberas
Jaime Sagüillo Revilla

Tutor Carlos López Nozal
DEPARTAMENTO DE INGENIERÍA CIVIL
Área de Lenguajes y Sistemas Informáticos

Índice

Índice de ilustraciones

Descripción de la aplicación

Se quiere implementar una aplicación que gestione los datos de una agenda (contactos, tipo de contactos, llamadas). La funcionalidad de la aplicación tanto en modo consola como en gráfico, permite:

- Elegir sistema de persistencia (base de datos o fichero binario)
- Insertar
    - Un nuevo contacto
    - Una nueva llamada
    - Un nuevo tipo de contacto
- Actualizar
    - Un contacto
    - Una llamada
    - Un tipo de contacto
- Consultar
    - Todos los contactos
        - Filtrados/ordenados por apellido
        - Filtrados/ordenados por nombre
    - Todas Llamadas
        - Filtrados/ordenados por contacto
        - Filtrados/ordenados por fecha de realización
    - Tipos de contacto

Diseño arquitectónico

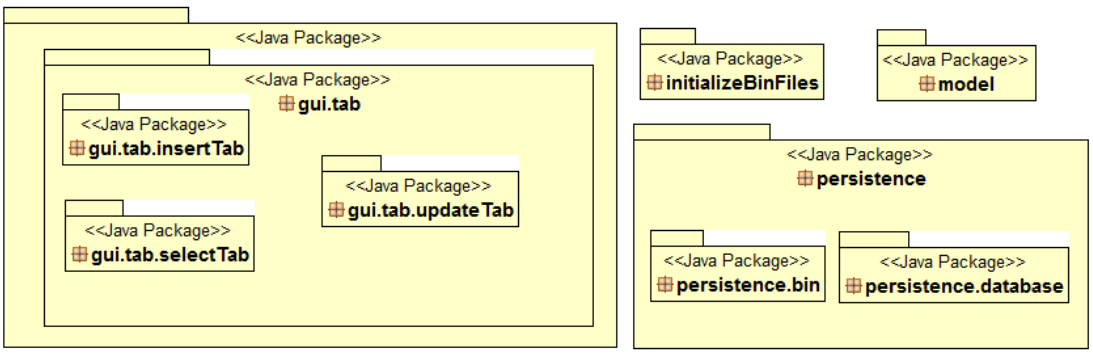

Ilustración 1 - Diseño arquitectónico

Diseño detallado del subsistema main.java.persistence

Patrones creacionales

Singleton

Problema

Deseamos obtener una única conexión a la base de datos. De manera que todos los objetos utilicen la misma conexión.

Solución

Para ello aplicamos el patrón creacional Singleton, el cual nos permitirá controlar que solo tendremos una única instancia a la base de datos.

Participantes

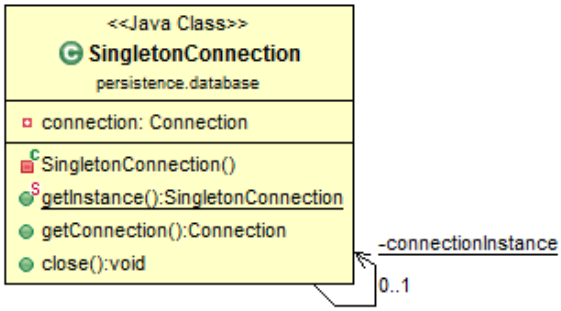

Ilustración 2 - Diagrama de clase – SingletonConnection

Además, para la clase que contiene todas las operaciones sobre la base de datos nos interesa mantener una única instancia, ya que estas operaciones no cambian y pueden ser compartidas por las distintas persistencias de contactos, llamadas y tipos de contactos.

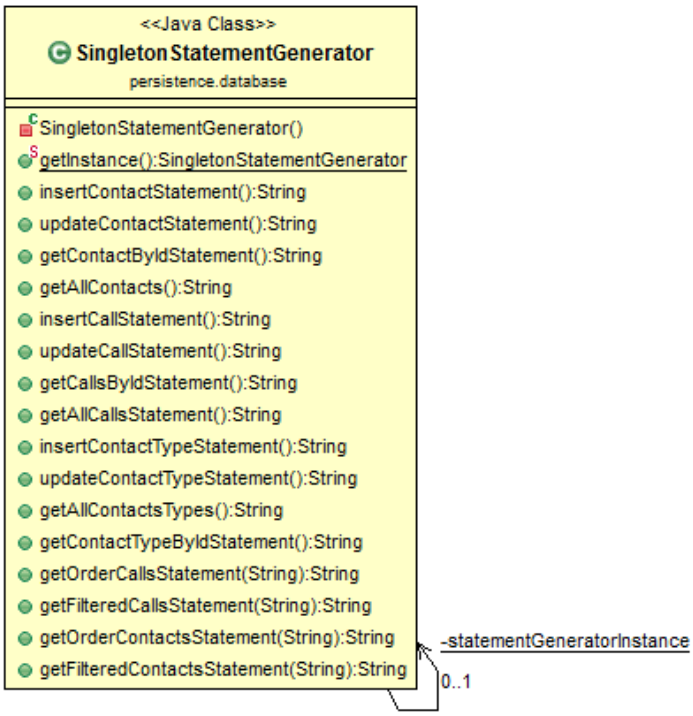

Ilustración 3 - Diagrama de clase – SingletonStamentGenerator

Factory Method

Problema

Deseamos abstraernos de la instanciación de la persistencia para los diferentes tipos de objetos, es decir, la persistencia para las llamadas, contactos, tipos de contactos.

Solución

La clase FactoryDataBase y la clase FactoryBinFile, las cuales implementan la interfaz IFactoryPersistence, se ocuparán de la creación de la persistencia para las llamadas, contactos y tipos de contacto por nosotros. De esta manera nos abstraeremos de que acceso a datos debemos de instanciar, sino que simplemente los obtendremos cuando les necesitemos.

Participantes

- La factoría de BinFiles:

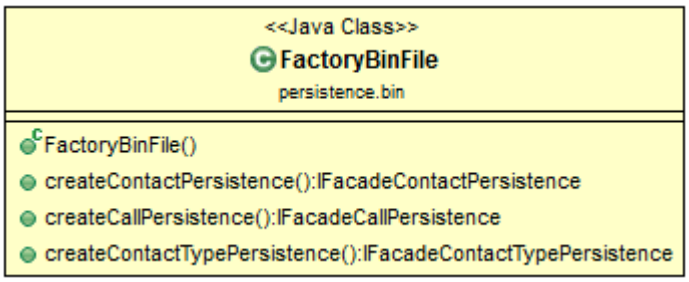

Ilustración 4 – FactoryBinFile

- La factoría de la base de datos

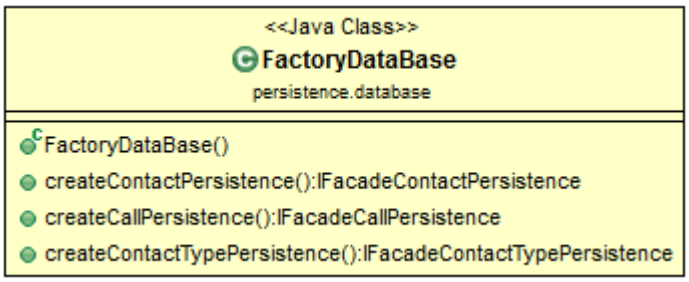

Ilustración 5 – FactoryDataBase

Abstract Factory

Problema

En este sistema tenemos 2 métodos de persistencia, como ya sabemos. En cada momento desearemos trabajar con una persistencia determinada, ya sea con la base de datos o con los ficheros binarios.

Solución

Así, Abstract Factory nos proporciona una metodología de trabajar en cada momento con uno de estos conjuntos de productos, usando a la vez todo el conjunto, y no permitiéndonos de esta manera que utilicemos parte de un conjunto y parte del otro. Lo cual es lo que deseamos con nuestro sistema de persistencia.

Participantes

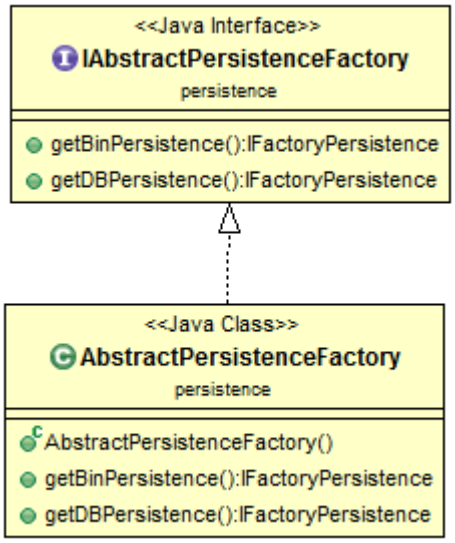

Ilustración 6 - Diagrama de clases – AbstractPersistenceFactory

Patrones estructurales

Facade

Problema

Como todos sabemos el acceso a las persistencias depende de apertura y cierre de la conexión o de los stream a los ficheros, nombres concretos de los ficheros, etc. Lo cual hace que debamos tener conocimientos para saber cómo interactuar con el sistema. Y deseamos que esto no sea así, abstrayendo al cliente de conocimientos internos del sistema, simplificando el acceso a las distintas funcionalidades.

Solución

De esta manera, Facade nos proporciona una manera sencilla de implementar una interfaz que simplifique el conocimiento que el cliente debe de tener sobre el sistema. Ocultando prácticamente toda la complejidad, y permitiendo al cliente que solo tenga que interactuar con esta instancia.

Participantes

Como hemos venido contando, existen 2 tipos de persistencia, por lo cual queremos que los 2 tipos implementen los mismos métodos. Mediante las siguientes interfaces es precisamente lo que conseguimos:

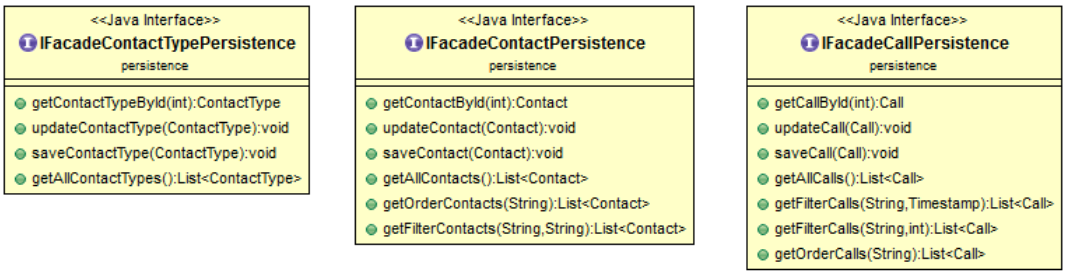

Ilustración 7 - Diagrama de interfaces – Facades

A continuación, tenemos las 2 implementaciones de las interfaces anteriores. Implementaciones que resultan ser los Facades para nuestros sistemas de persistencia.

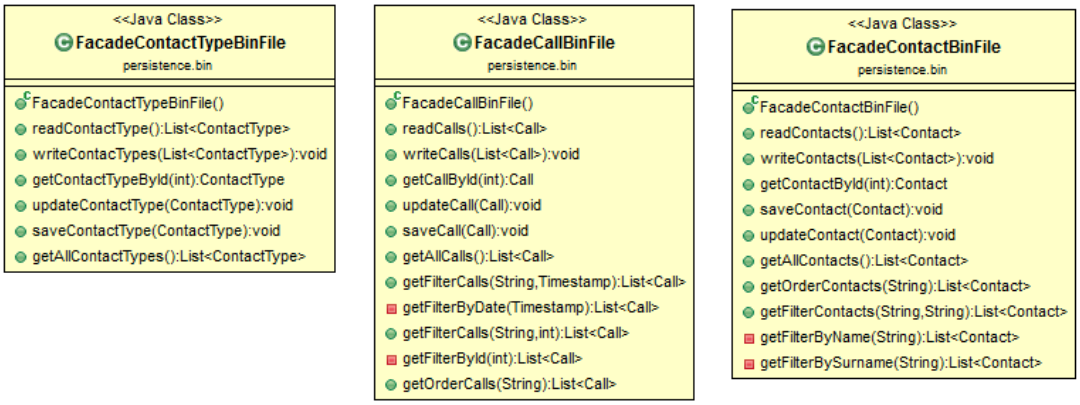

Ilustración 8 - Diagrama de clases - Facade - Persintencia tipo binaria

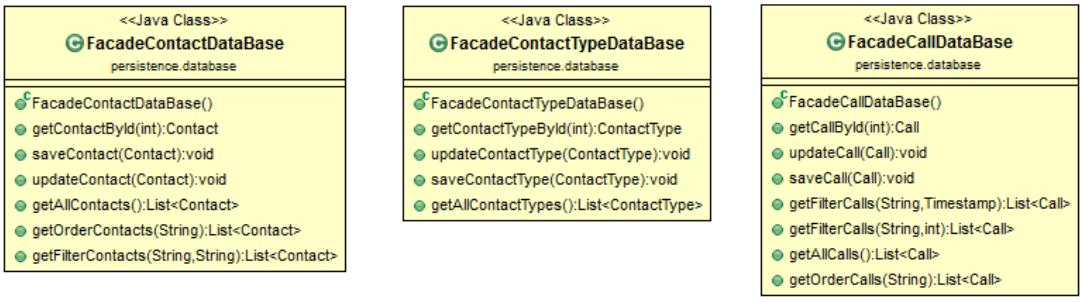

Ilustración 9 - Diagrama de clases - Facade - Persistencia base de datos

Composite

Problema

Existe una estructura jerárquica que se beneficia de usar este patrón para facilitar la actualización de la persistencia en todos los niveles de la aplicación.

Solución

Aplicar el patrón composite, de manera que cuando la clase padre decida cambiar de tipo de persistencia, sean las clases hijas las encargadas de llevar a cabo estos cambios de manera correcta.

Participantes

A continuación, mostramos la interfaz y las clases que participan en este patrón de diseño:

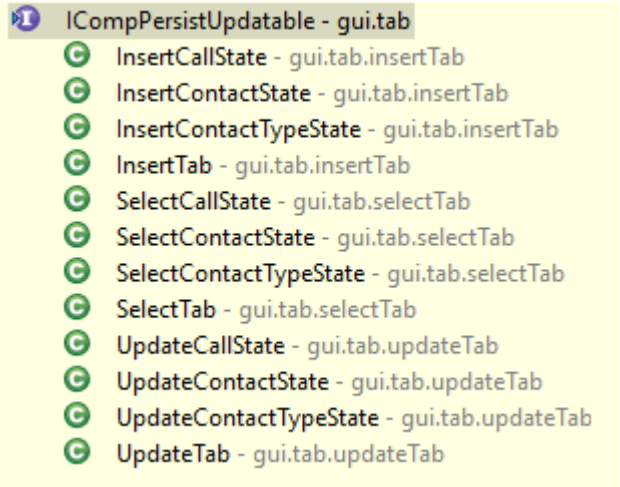

Ilustración 10 - Participantes del patrón composite

Patrones de comportamiento

State

Problema

Las distintas operaciones que llevamos a cabo (insertar, actualizar y consultar) sobre nuestra agenda se deben de realizar sobre tres tipos de tablas (contactos, llamadas y tipos de contacto), de manera que en función de la tabla a utilizar dichas operaciones varían.

Solución

Utilizar el patrón state, de manera que en función de la tabla a utilizar (estado) las operaciones a utilizar cambien para trabajar sobre la tabla adecuada.

Participantes

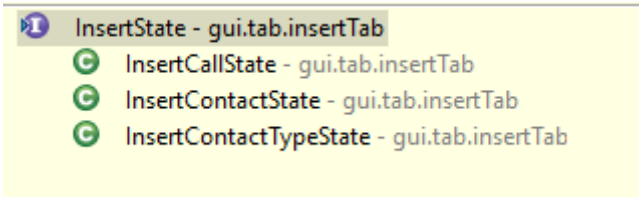

Ilustración 11 - Participantes del patrón state para la operación insertar

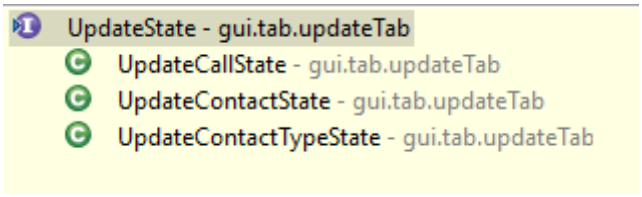

Ilustración 12 - Participantes del patrón state para la operación actualizar

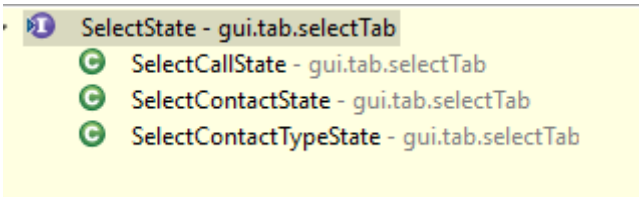

Ilustración 13 - Participantes del patrón state para la operación consultar

A continuación, mostramos uno de los esquemas que siguen nuestras implementaciones del patrón de diseño state:

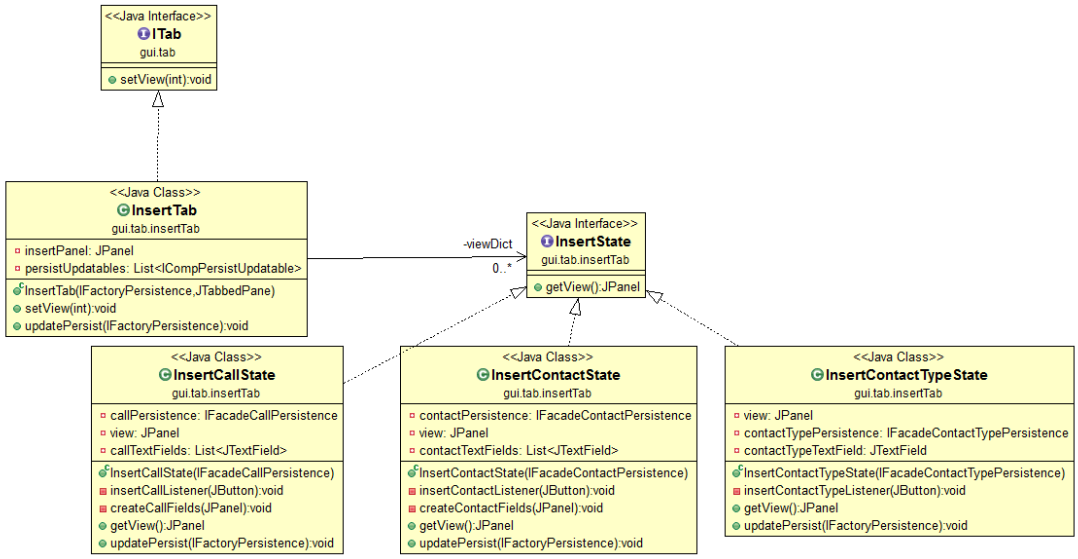

Ilustración 14 - Diagrama UML de la implementación del patrón state para insertar

Manual del programador

Carpeta src

Esta parte del proyecto albergará toda la lógica de la aplicación. Además, contendrá toda la estructura de paquetes anteriormente mostrada *Diseño Arquitectónico,* la parte de test de la aplicación enfocada únicamente a la persistencia de los datos, junto con los diagramas de clases de dicha carpeta.

Carpeta BinFiles

Esta parte se encuentran los ficheros binarios *Calls.dat, Contacts.dat* y *ContactTypes.dat* sobre los que se llevarán a cabo las operaciones. Cabe destacar que se incluye dentro de la carpeta anteriormente explicada *"src/initializeBinFiles",* una clase java denominada *BinFilesInitialize* la cual podremos ejecutar si queremos inicializar estos ficheros binarios a su estado inicial, es decir con la misma información de ambas tablas proporcionada. Si queremos llevar a cabo este paso antes de ejecutar dichos ficheros tendremos que editarlos vaciando todo su contenido y a continuación, ejecutar el *initializeBinFiles.*

Carpeta info

Esta carpeta contendrá el enunciado del proyecto.

Carpeta rsc

Esta carpeta contendrá el fichero *.sql* con las tablas a crear en la base de datos junto con los inserts proporcionados. Además, dicha carpeta almacenará el archivo *server.properties* necesario para el correcto funcionamiento de la base de datos *hsqldb.*

Carpeta útil

En esta parte del proyecto se encuentra un archivo *.dat* que contendrá la sentencia necesaria para arrancar la base de datos, para ello desde el *Explorador de archivos* ejecutaremos dicho archivo y arrancaremos la base de datos *hsqdbl*

Manual de usuario

Esta aplicación esta implementada tanto para su uso con interfaz de consola como por interfaz gráfica, por lo que dicha aplicación podrá ser ejecutada dentro del propio entorno de *Eclipse* (u otro entorno de desarrollo), desde la consola de *Windows* o desde su interfaz gráfica.

Desde eclipse

Al ejecutar la aplicación, nuestra aplicación posibilitará al usuario que elija el tipo de persistencia que desea utilizar como podemos observar en la siguiente ilustración.

```
* Elija el sistema de persistencia:
** 1) Base datos SQL
** 2) Sistema de persistencia mediante ficheros binarios
** 0) Salir de la aplicación
```

Ilustración 15 - Menú inicial

Además, dicho menú permitirá en todo momento volver a la sección anterior o incluso finalizar la ejecución de la aplicación seleccionando como opción el número cero. Una vez seleccionado el tipo de persistencia a utilizar el menú nos permitirá realizar todas las tareas pedidas en el apartado Diseño arquitectónico.

Desde la consola de Windows

1. Para ejecutar la aplicación desde la consola de Windows tendremos que arrancar dicha consola.
2. Posicionarnos en la ruta del proyecto como podemos ver en la siguiente ilustración.

```
Microsoft Windows [Versión 10.0.14393]
(c) 2016 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Daniel>cd C:\Users\Daniel\Desktop\DMSW\PersistentAgenda
```

Ilustración 16 - Consola de comandos - ruta proyecto

3. En este punto tendremos dos opciones de ejecución:
   a. Ejecutar únicamente la aplicación

   Para llevar a cabo esta acción bastará con ejecutar el siguiente comando.

```
C:\Users\Daniel\Desktop\DMSW\PersistentAgenda>gradle run
```

Ilustración 17 - Consola de comandos - gradle run

   b. Crear el javadoc, ejecutar los test y aplicación.

   Para llevar a cabo esta acción bastará con ejecutar el siguiente comando.

```
C:\Users\Daniel\Desktop\DMSW\PersistentAgenda>gradle release
```

Ilustración 18 - Consola de comandos - gradle reléase

Desde la interfaz gráfica

Para ejecutar la interfaz gráfica bastaría con ejecutar el *main* situado en el paquete *gui y en la clase*: ***MainGui.java.*** Y de esta manera obtendríamos la siguiente ventana*:*

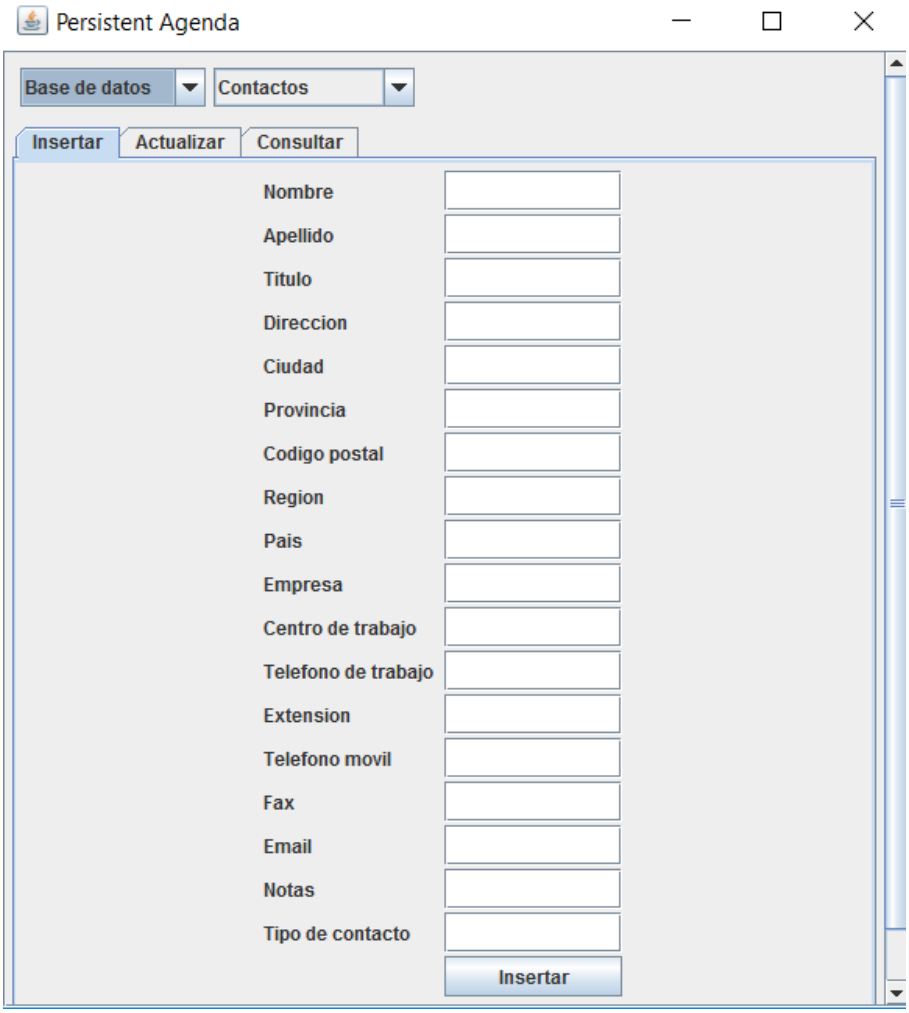

Ilustración 19 - Ventana inicial de la aplicación

La aplicación está compuesta por dos desplegables en su parte más superior, en las que se permite la elección entre las persistencias y los datos sobre los que se quiere trabajar. Además, justo debajo se encuentran tres pestañas con las distintas operaciones que se pueden realizar sobre nuestros datos, como mostramos en la siguiente ilustración:

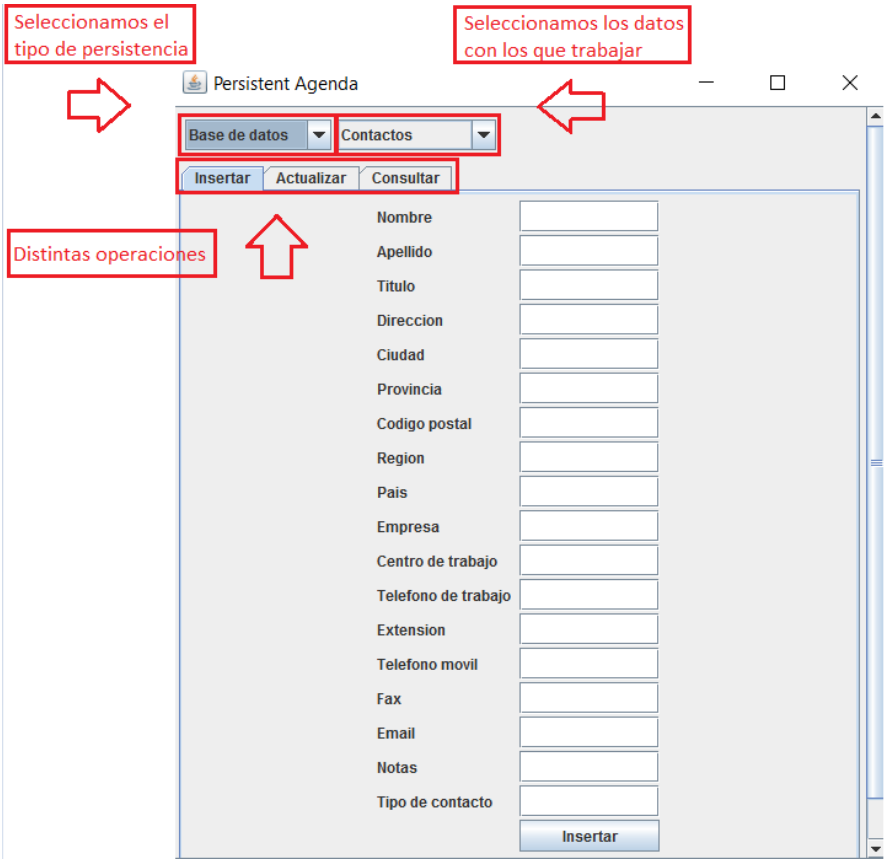

Ilustración 20 - Descripción de los elementos de la aplicación

Para finalizar, cuando realizamos una consulta se nos mostrara la información obtenida en una nueva ventana, como la mostrada a continuación:

Ilustración 21 - Ejemplo de ventana de una consulta sobre los tipos de contacto