Diseño de un subsistema de interfaz gráfica de usuario para una aplicación de Agenda

Índice de Contenido

1 Requisitos Software

- Usar un IDE, recomendado Eclipse IDE for Java Developers http://www.eclipse.org/downloads/
- Plugin Eclipse Window Builder https://eclipse.org/windowbuilder/
- Herramientas de modelado para diagramas de clase. Por ejemplo podeis usar ObjectAiD http://www.objectaid.com/ u otras herramientas que os permitan dibujar diagramas UML como http://draw.io/.

2 Objetivos Especificos

- Diseñar interfaces gráficas de usuario usando patrones de diseño.
- Diseñar software fáciles de mantener.
- Aplicar 2 patrones de diseño que pueden ser creacionales, estructurales y de **comportamiento (al menos 1 de comportamiento)** sobre Java.

3 Enunciado

Se quiere implementar una aplicación que gestione los datos de una agenda (Contactos, Tipo de contactos, Llamadas). La funcionalidad de la aplicación 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

3.1 Diseño Arquitectónico

El diseño arquitectónico de la aplicación está compuesto por tres subsistemas de la aplicación y uno de pruebas. Las dependencias de los subsistemas de la agenda se muestran en la *Ilustración 1*. La descripción de los subsistemas es la siguiente:

- Subsistema `main.java.persistence`
    - Contiene las clases que permiten acceder al sistema de persistencia de la agenda.
    - Implementa dos formas de persistencia, una basada en persistencia de base de datos y otra basada en persistencia de flujos de ficheros binarios.
    - Permite configurar al cliente con uno de los dos subsistemas sin que tenga dependencias sobre las implementaciones concretas. El cliente usa una propiedad de la aplicación para seleccionar la implementación concreta.
- Subsistema `main.java.gui`
    - Contiene las clases que permiten al usuario interactuar con la agenda (consultar, insertar, actualizar).

- Los diseños de las clases de interfaz gráfica de usuario:
  - Son reutilizables. Minimizando acoplamientos entre sus clases y maximizando su cohesión.
  - Especifican, encolan y ejecutan órdenes en momentos diferentes.
  - Un cambio en uno de los objetos del modelo requiere cambiar otros objetos, y no se conocen cuántos objetos deben ser cambiados

Se supone que en el constructor de la ventana principal (`JFrame`) se crean:

- Una `Collection<Contact>` contactos añadiendo un conjunto de instancias `Contact`.
- Una `Collection<Call>` llamadas añadiendo un conjunto de instancias `Call`.

Todas las operaciones de gestión de datos desde la GUI (*Graphical User Interface*) se harán sobre esas colecciones (consultar, navegar, insertar, borrar, actualizar).
Descripción de algunos problemas generales y específicos del diseño del GUI:

- Gran número de atributos en la clase `Contact`
- Datos de distintos tipos: Fecha, Hora, String
- Validar campos de entrada
- Minimizar el número de interacciones de usuario para acceder a la funcionalidad
- Utilizar sistemas de navegación conocidos por el usuario
- Incorporar *tooltips* en los componentes
- Seguir estándares básicos (Nombre de botón “Aceptar”, “Cancelar”. Pares de botones “cancelar” “aceptar" siempre en el mismo orden: izquierda derecha, etc…)
- Colores suaves y atractivos

- Subsistema `main.java.modelo`
  - Contiene las clases de tipo entidad (*entity class*) del contexto de la aplicación de agenda.
- Subsistema `test.java`
  - Contiene las clases que sirven para probar la funcionalidad del subsistema.

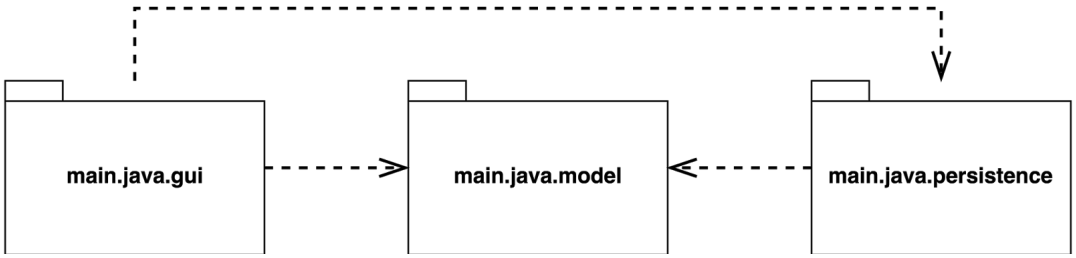

Ilustración 1: Diseño arquitectónico de la aplicación Agenda

3.2 Diseño de Datos

En el diagrama de clases de la *Ilustración 2* se muestran las clases de tipo entidad y sus relaciones en la aplicación agenda. Un contacto realiza cero o varias llamadas siempre asociadas a un contacto. Los contactos tienen asociado un único tipo de contacto.

Ilustración 2: Diagrama de clases del subsistema ubu.lsi.dms.agenda.modelo

En la *Tabla 1* se muestran los comandos SQL concretos para crear una base de datos relacional que sirva para soportar la persistencia del subsistema `main.java.modelo`.

```
CREATE TABLE Contacts (
  id INT NOT NULL PRIMARY KEY IDENTITY,
  name VARCHAR(50) NULL,
  surname VARCHAR(50) NULL,
  title VARCHAR(50) NULL,
  address VARCHAR(255) NULL,
  city VARCHAR(50) NULL,
  province VARCHAR(20) NULL,
  postal_code VARCHAR(20) NULL,
  region VARCHAR(50) NULL,
  country VARCHAR(50) NULL,
  company_name VARCHAR(50) NULL,
  workstation VARCHAR(50) NULL,
  work_phone VARCHAR(30) NULL,
  work_extension VARCHAR(20) NULL,
  mobile_phone VARCHAR(30) NULL,
  fax_number VARCHAR(30) NULL,
  email VARCHAR(50) NULL,
  contact_type_id INT NULL,
  notes VARCHAR(512) NULL
```

);
 	
CREATE TABLE Calls (
  id INT NOT NULL PRIMARY KEY IDENTITY,
  contact_id INT NOT NULL,
  call_date TIMESTAMP NOT NULL,
  subject VARCHAR(255) NOT NULL,
  notes VARCHAR(255) NULL,
);
 	
CREATE TABLE ContactsTypes (
  id INT NOT NULL PRIMARY KEY IDENTITY,
  contact_type VARCHAR(50) NOT NULL,
);

Tabla 1: Comandos SQL de creación en el modelo relacional de datos

En UbuVirtual puedes obtener el código fuente de la primera versión de la práctica.

4 Productos a Entregar

Un proyecto de Gradle (bibliotecas, recursos y fuentes Java) con la implementación de los subsistemas :

- `main.java.persistence`
- `main.java.modelo`
- `main.java.gui`

Todo el código fuente debe estar comentado y bien formateado.

Un documento de memoria con el formato de la plantilla disponible en UBUVirtual y donde se presenten los siguientes apartados:

- Descripción de la aplicación
- Diseño arquitectónico
- Diseño detallado del subsistema `main.java.gui`. La documentación de diseño debe especificarse desde la perspectiva de patrones de diseño (problema, solución y participantes, todos ellos en el contexto específico del software que estamos creando). Se deben **utilizar** por lo menos **dos patrones de diseño**.
- Manual del programador donde se explique el contenido de cada carpeta (src, res, lib, etc), cómo compilar y ejecutar la aplicación.
- Manual de usuario donde se explique como funciona la aplicación.

5 Bibliografia

- Carlos López. Apuntes de la asignatura: Diseño y Mantenimiento del software, 2014.
- López, Carlos, Raúl Marticorena, Judith Antolín, y Ignacio Cruzado. «Inclusión de patrones de diseño en un plan de estudios de Ingeniería Técnica en Informática de Gestión». En *Actas de las IX Jornadas de Enseñanza Universitaria en Informática JENUI 2003. Cádiz. ISBN 84-283-2845-5*, 265-272, 2003. http://bioinfo.uib.es/~joemiro/aenui/procJenui/Jen2003/loincl.pdf