|  |
| --- |
|  |
| AGENDA DE CONTACTOS |
| Documentación de la práctica inicial Java para PSP y PMDM |
|  |
| **Carlos García Corpas, Javier Sánchez Gómez** |
| **04/10/2016** |

ÍNDICE

[ANÁLISIS DEL SISTEMA 3](#_Toc463272926)

[ACTIVIDAD ASI 2: ESTABLECIMIENTO DE REQUISITOS 3](#_Toc463272927)

[Tarea ASI 2.1: Obtención de requisitos 3](#_Toc463272928)

[Tarea ASI 2.2: Especificación de casos de uso 4](#_Toc463272929)

[ACTIVIDAD ASI 4: ANÁLISIS DE LOS CASOS DE USO 8](#_Toc463272930)

[Tarea ASI 4.1: Identificación de las clases asociadas a un caso de uso 8](#_Toc463272931)

[ACTIVIDAD ASI 5: ANÁLISIS DE CLASES 8](#_Toc463272932)

[Tarea ASI 5.1: Identificación de Responsabilidades y Atributos 8](#_Toc463272933)

[Tarea ASI 5.2: Identificación de Asociaciones y Agregaciones 9](#_Toc463272934)

[ACTIVIDAD ASI 8: DEFINICIÓN DE INTERFACES DE USUARIO 10](#_Toc463272935)

[Tarea ASI 8.1: Especificación de principios generales de la interfaz 10](#_Toc463272936)

[DISEÑO DEL SISTEMA 11](#_Toc463272937)

[ACTIVIDAD DSI 4: DISEÑO DE CLASES 11](#_Toc463272938)

[Tarea DSI 4.1: Identificación de clases adicionales 11](#_Toc463272939)

[Tarea DSI 4.3: Identificación de atributos de las clases 11](#_Toc463272940)

[Tarea DSI 4.6: Descripción de métodos de las operaciones 11](#_Toc463272941)

[CONSTRUCCIÓN DEL SISTEMA 12](#_Toc463272942)

# ANÁLISIS DEL SISTEMA

## ACTIVIDAD ASI 2: ESTABLECIMIENTO DE REQUISITOS

### Tarea ASI 2.1: Obtención de requisitos

El objetivo es desarrollar un software que permita a un usuario gestionar, mediante el uso de interfaces gráficas, una agenda de contactos.

El catálogo de requisitos para el desarrollo de esta aplicación es el siguiente:

**Requisitos funcionales:**

* La aplicación debe tener las siguientes funcionalidades:
  + Añadir, borrar, modificar y buscar contactos.
  + Cargar, guardar y mostrar agenda.
* Un contacto está formado por un nombre y un teléfono. Por hacer algo más realista la agenda, hemos incluido como requisito que:
  + El nombre de un contacto debe estar entre 1 o 20 caracteres.
  + El número de teléfono de un contacto debe estar entre 3 y 13 caracteres.
  + Tanto el nombre como el número pueden admitir números y letras.
* Los contactos deben almacenarse en un fichero y luego poder ser leídos de dicho fichero.

**Requisitos técnicos:**

* Los contactos de la agenda siempre estarán cargados en memoria. Cuando se lea el fichero, los contactos existentes en memoria, serán reemplazados por los contactos leídos del fichero, previo aviso al usuario.
* Es opcional trabajar con bases de datos en vez de con ficheros (usaremos ficheros en este proyecto).
* El entorno de desarrollo debe ser Eclipse.

**Requisitos de la interfaz gráfica:**

* La aplicación debe tener dos pantallas, al menos:
  + Pantalla inicial con un menú que permita acceder a las opciones “Añadir Contacto”, “Borrar Contacto”, “Modificar Contacto”, “Buscar Contacto”, “Guardar Fichero”, “Mostrar Fichero”, “Leer Fichero”
  + Para cada una de las opciones deberá haber una pantalla, que no podrá cerrarse. Debe tener un botón para volver a la pantalla inicial.

### Tarea ASI 2.2: Especificación de casos de uso

En esta tarea trataremos de analizar los casos de uso que se pueden obtener del catálogo de requisitos, identificando los actores y las posibles relaciones.

Lo primero es identificar los actores del programa. En los requisitos no se habla de ningún tipo de usuario, por lo que vamos a suponer que sólo hay un tipo de actor al que vamos a llamar usuario. El usuario tendrá acceso a todas las funcionalidades del programa.

Dichas funcionalidades se pueden definir cada una como un caso de uso individual, con lo cual quedaría el diagrama de los casos de uso de la siguiente forma:

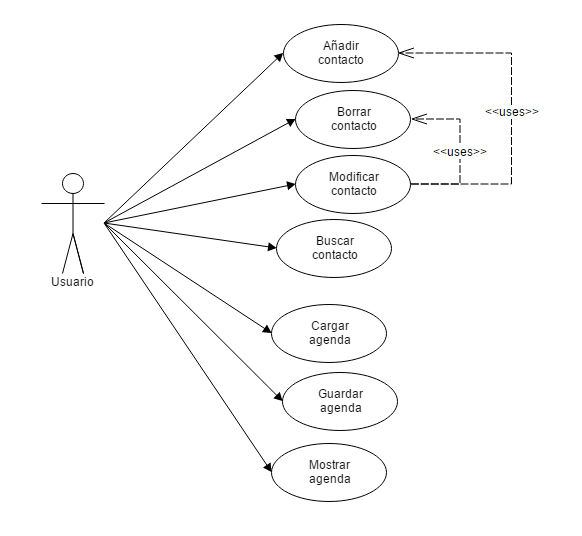


Figura 1: Diagrama de Casos de Uso

Este diagrama describe todas las funcionalidades del programa y las relaciones que van a tener con el usuario. Sin embargo, podemos tener en cuenta otros casos de uso o relaciones posibles.

Destacar que existe una relación del caso de uso Modificar que va a *usar* a los casos de uso Añadir y Borrar contacto. Esto es, cada vez que el usuario vaya a hacer uso de esta funcionalidad, el programa (tras confirmar si la operación se puede realizar) borrará el contacto sin modificar y añadirá a continuación el nuevo contacto con las modificaciones.

Uno de ellos sería un caso de uso *Leer contacto*, que haría referencia a cada vez que se accede a la memoria para obtener un contacto. Por ejemplo, el caso de uso añadir contacto necesita leer los contactos de la agenda para ver si el contacto que se desea añadir existe previamente.

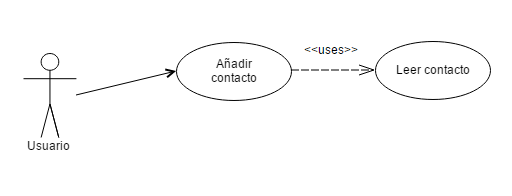


Figura 2: Caso de Uso Leer Contacto

Sin embargo, este caso de uso no responde a una funcionalidad del programa como tal. No es algo que el usuario pueda hacer, si no que es más bien un mecanismo interno, dado lo cual decidimos no incluirlo en el diagrama de casos de uso.

Otra situación a tener en cuenta es la posible relación <<uses>> o <<includes>> entre los casos de uso *Modificar* y *Borrar* *contacto* con *Mostrar* *agenda*. Todavía es un poco pronto para tenerlo en cuenta pero es posible que a la hora de modificar o borrar un contacto utilicemos la función, o parte de la función, mostrar agenda.

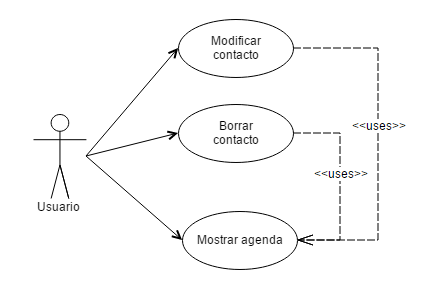


Figura 3: posible relación <<uses>> entre Casos de Uso

Una vez obtenidos los casos de uso, es necesario definirlos.

No vamos a definir todos los casos de uso, que son similares entre sí, si no que vamos a elegir uno de cada grupo, considerando como grupo los casos relacionados con contactos frente a los relacionados con agenda.

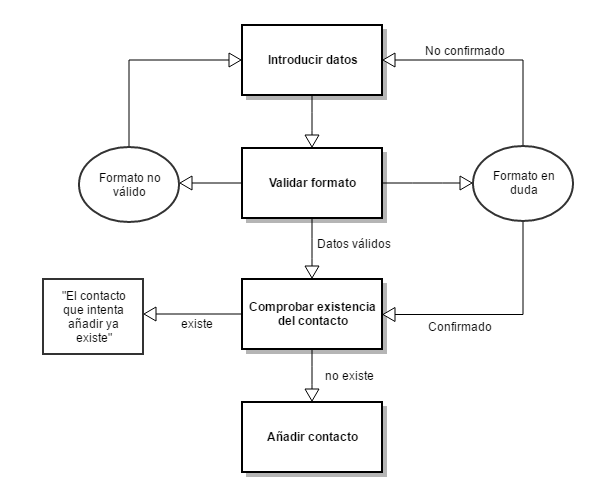
Los casos de uso elegidos son *Añadir* *contacto* y *Cargar* *agenda*.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ESPECIFICACIÓN DE CASO DE USO** | | | | | | |
| **Autor** | Carlos García, Javier Sánchez | | | | **Aplicación** | Agenda |
| **Identificador** | | 1 | **Nombre** | Añadir contacto | | |
| **DESCRIPCIÓN BREVE** | | | | | | |
| Este caso de uso describe la funcionalidad que permite añadir un contacto a la agenda. | | | | | | |
| **ACTORES** | | | | | | |
| Usuario | | | | | | |
| **TRIGGERS O DISPARADORES** | | | | | | |
| A petición del usuario. | | | | | | |
| **PRECONDICIONES** | | | | | | |
| Agenda cargada en memoria, ya sea por cargar una existente o por crear una nueva. | | | | | | |
| **FLUJO NORMAL** | | | | | | |
| 1 | Pedir al usuario mediante una ventana de entrada de datos que introduzca el nombre y el teléfono del contacto que desea añadir. | | | | | |
| 2 | Comprobar la validez del formato del contacto. Esto es, el nombre debe tener entre 1 y 20 caracteres, y el número de teléfono debe tener entre 3 y 13. Ambos admiten tanto números como letras. | | | | | |
| 3 | Comprobar si el contacto existe en la agenda cargada en memoria, para no sobrescribirlo. Se considera que un contacto existe cuando su número de teléfono está repetido, sin embargo, si pueden existir dos contactos con el mismo nombre. | | | | | |
| 4 | Se crea el contacto y se añade a la agenda. | | | | | |
| **FLUJO ALTERNATIVO** | | | | | | |
| 2.1 | En caso de que el nombre del contacto contenga números, o el número de teléfono contenga letras, se preguntará al usuario si está seguro de no haberse equivocado. Si afirma estar seguro, seguirá el flujo normal. | | | | | |
| 2.2 | Si los datos introducidos por el usuario son inválidos, el programa se lo comunicará y volverá a la ventana de entrada de datos para volver a introducir un contacto. | | | | | |
| 3.1 | Si el contacto ya existe en memoria, se le comunicará al usuario con una ventana emergente. | | | | | |
| **POSTCONDICIONES** | | | | | | |
| Contacto añadido a la agenda. | | | | | | |
| **COMENTARIOS ADICIONALES** | | | | | | |
| Al utilizar interfaces gráficas por primera vez en este proyecto nos encontramos con herramientas que pueden impedir al usuario que introduzca un valor con formato inválido. Por ejemplo, puedes establecer en una caja de entrada de texto un límite de caracteres, por lo que puedes controlar un cierto formato desde la propia interfaz. | | | | | | |

Tabla 1: Caso de Uso “Añadir contacto”

Para aclarar cómo funciona este caso de uso, vamos a ampliarlo con el siguiente esquema. Tras introducir los datos, el programa los valida, pudiendo pasar como flujo alternativo que el formato no sea válido (en cuyo caso devuelve al usuario a la pantalla de introducir datos), o que el formato sea inusual(\*), en cuyo caso se pide confirmación al usuario.

Tras validar el formato se comprueba que el contacto que se desea agregar no exista. En caso de que exista, se le comunicará al usuario. Si no existe, el contacto será creado y añadido a la agenda.



La descripción del caso de uso *Cargar* *agenda* es la siguiente:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ESPECIFICACIÓN DE CASO DE USO** | | | | | | |
| **Autor** | Carlos García, Javier Sánchez | | | | **Aplicación** | Agenda |
| **Identificador** | | 2 | **Nombre** | Cargar agenda | | |
| **DESCRIPCIÓN BREVE** | | | | | | |
| Describe la funcionalidad correspondiente a la acción de cargar un fichero con una agenda. | | | | | | |
| **ACTORES** | | | | | | |
| Usuario | | | | | | |
| **TRIGGERS O DISPARADORES** | | | | | | |
| A petición del usuario. | | | | | | |
| **PRECONDICIONES** | | | | | | |
| No hay precondiciones. | | | | | | |
| **FLUJO NORMAL** | | | | | | |
| 1 | El usuario selecciona el fichero del que quiere cargar la agenda. | | | | | |
| 2 | El sistema comprueba que el fichero exista y sea válido. Los ficheros válidos serán aquellos en los que haya una agenda guardada, incluso aunque esté vacía. | | | | | |
| 3 | El sistema lee los datos del fichero y se carga en memoria, machacando los datos de la anterior agenda cargada. | | | | | |
| **FLUJO ALTERNATIVO** | | | | | | |
| 2.1 | En caso de que el fichero no exista o no sea una agenda, el sistema se lo comunicará al usuario por pantalla. | | | | | |
| **POSTCONDICIONES** | | | | | | |
| Agenda cargada | | | | | | |
| **COMENTARIOS ADICIONALES** | | | | | | |
| Seguramente, con las opciones proporcionadas por la interfaz gráfica, será posible impedir que el usuario intente cargar un archivo que no exista o que no sea una agenda. Sin embargo, aunque esto se pueda tratar desde la parte visual del programa, debe estar controlado también en la parte del negocio. | | | | | | |

Tabla 2: Caso de Uso “Cargar Agenda”

## ACTIVIDAD ASI 4: ANÁLISIS DE LOS CASOS DE USO

### Tarea ASI 4.1: Identificación de las clases asociadas a un caso de uso

Esta tarea es bastante sencilla para este proyecto. Si se analiza el diagrama de clases se pueden dividir los tipos de casos de uso en dos apartados:

* Casos de uso referentes a la manipulación de contactos.
* Casos de uso referentes al uso de la agenda.

Lo normal viendo esto es deducir que va a haber dos clases, una referente a los contactos y otra referente a la agenda como tal.

## ACTIVIDAD ASI 5: ANÁLISIS DE CLASES

En el apartado anterior hemos deducido que vamos a necesitar dos clases, que vamos a llamar Contacto y Agenda. En este apartado lo que trataremos es, primero, de identificar los métodos y atributos de dichas clases, y segundo, identificar cómo van a estar desarrolladas entre sí.

Cabe destacar que esta actividad sólo se refiere a clases de negocio. No se contempla aquí ninguna clase referida a la interfaz gráfica ni al acceso a datos.

### Tarea ASI 5.1: Identificación de Responsabilidades y Atributos

Dados los requisitos, los atributos y métodos que en principio van a tener nuestras clases son:

1. Clase Contacto:
   1. Atributos: un nombre y teléfono. Ambos serán privados a la clase y de tipo String.
   2. Métodos: necesitará un método que valide si el contacto cumpla el formato o no. Esto posiblemente se acabe añadiendo al constructor.
2. Clase Agenda:
   1. Atributos: ninguno.
   2. Métodos:
      * Cargar agenda.
      * Guardar agenda.
      * Añadir contacto.
      * Borrar contacto.
      * Modificar contacto.
      * Buscar contacto por nombre y por teléfono.

Observar que no va a tener un método que sea mostrar agenda, ya que de eso se encargará la interfaz gráfica.

Con esta distribución de los métodos, la clase principal del negocio es claramente la agenda. Se podría ver de forma que fueran los contactos los que se añadieran, borraran y demás a la agenda. Sin embargo nos parece que facilita el desarrollo del código que sea la agenda la que tenga esta serie de funcionalidades.

### Tarea ASI 5.2: Identificación de Asociaciones y Agregaciones

Vistos los atributos y métodos de las dos clases, sólo queda ver cómo se van a relacionar entre sí.

Tal y como entendemos la aplicación, una agenda es un objeto compuesto de contactos, está formada por contactos, aunque pueda no tener ninguno. Por ello, esta es una relación de composición en la cual la clase Agenda tiene acceso a la clase Contacto, pero no al revés.

El hecho de que sea composición pero no agregación se debe a que si una agenda se borra, deberían de borrarse los contactos que haya en ella, no puede haber contactos fuera de una agenda.

La cardinalidad tiene que expresar que una agenda puede estar compuesta de cero a varios contactos, mientras que un contacto sólo pertenece una agenda. La navegabilidad tiene que indicar que la agenda tiene que poder acceder a la información de sus contactos.

El diagrama de clases de negocio es por tanto el siguiente:

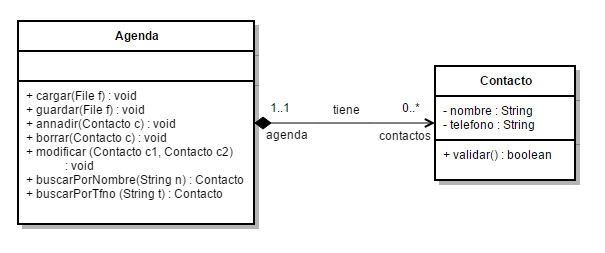


Figura 4: Diagrama de Clases de Negocio

## ACTIVIDAD ASI 8: DEFINICIÓN DE INTERFACES DE USUARIO

### Tarea ASI 8.1: Especificación de principios generales de la interfaz

En esta tarea vamos a tratar de analizar cómo debe ser la interfaz de usuario para cumplir con los requisitos.

Lo primero es definir cómo va a ser la ventana principal, la que se abre al iniciar el programa. Esta ventana va a contener un menú desde el cual acceder a las funcionalidades del sistema.

Los requisitos básicamente piden que el programa conste de más de una ventana, y que desde las ventanas secundarias sólo se pueda volver a la ventana original pulsando un botón, que no se puedan cerrar de otra forma las ventanas.

Las funcionalidades que generarán una ventana propia, y con esto se cumple el primer requisito, serán:

* Añadir, modificar, borrar y buscar contacto.
* Mostrar y cargar agenda.

La opción de cargar la agenda también abrirá una ventana nueva, siendo esta ventana es el cuadro que permite seleccionar un archivo.

Para cumplir el segundo requisito en cuanto a interfaz de usuario será necesario añadir un botón de “Volver” en cada una de las ventanas emergentes.

Otro principio general de la interfaz es que vamos a intentar comunicar al usuario toda peculiaridad surgida del uso del programa, desde comunicarle que un contacto ha sido añadido, a posibles errores como que el fichero que intenta cargar está corrupto. Esto lo haremos usando ventanas emergentes (clase JOptionPane de Swing).

Por último, cada pantalla va a incluir una serie de componentes. Estos componentes son cajas de texto, tablas, botones, menús desplegables, etc. En la fase de diseño se especificará cómo van a ser cada una de las ventanas que realicemos.

# DISEÑO DEL SISTEMA

## ACTIVIDAD DSI 4: DISEÑO DE CLASES

### Tarea DSI 4.1: Identificación de clases adicionales

### Tarea DSI 4.3: Identificación de atributos de las clases

### Tarea DSI 4.6: Descripción de métodos de las operaciones

+ DISEÑO DE INTERFAZ GRÁFICA (LEETE EL ANÁLISIS DE INTERFAZ, QUE TE DOY TRABAJO)

# CONSTRUCCIÓN DEL SISTEMA