**INSTITUTO POLITÉNICO NACIONAL**

**Escuela Superior de Cómputo**

**Unidad de Aprendizaje:**

Programación Orientada a Objetos

**Grupo:**

2CV8

**Profesor:**

Daniel Cruz García

**Práctica 6**

**Alumnos:**

Dávila García Rivas Emiliano

Luciano Espina Melisa

**Fecha de entrega:**

09/06/2018

# Introducción

En la vida real la aplicación de la programación cambia mucho a como se ve en la escuela o en algún curso, ya que hay muchos casos que pasan y que se deben tomar en cuenta.

Por ejemplo, en las empresas no piden pequeños programas donde solo muestren por consola unos datos ingresados por teclado, piden programas tales como los que vemos en algunas páginas web como formularios o alguno para resolver la organización en la estructura de la empresa.

En esta práctica se aplicarán los conocimientos básicos implementando conceptos nuevos tales como “clases abstractas” e “interfaces”

# Marco teórico

## Interfaces en Java

Si una interfaz define un tipo (al igual que una clase define un tipo) pero ese tipo no provee de ningún método podemos preguntarnos: ¿qué se gana con las interfaces en Java? La implementación (herencia) de una interfaz no podemos decir que evite la duplicidad de código o que favorezca la reutilización de código puesto que realmente no proveen código.

En cambio, sí podemos decir que reúne las otras dos ventajas de la herencia: favorecer el mantenimiento y la extensión de las aplicaciones. ¿Por qué? Porque al definir interfaces permitimos la existencia de variables polimórficas y la invocación polimórfica de métodos.

Un aspecto fundamental de las interfaces en Java es separar la especificación de una clase (qué hace) de la implementación (cómo lo hace). Esto se ha comprobado que da lugar a programas más robustos y con menos errores. Pensemos en el API de Java.

El hecho de declarar una variable de tipo lista, por ejemplo, List <String> miLista; nos dice que miLista va a ser una implementación de List, pero todavía no hemos definido cuál de las posibles implementaciones va a ser. De hecho, el código podría definir que se implementara de una u otra manera en función de las circunstancias usando condicionales. O a nivel de programación, mantendríamos la definición como List y nos permitiría comprobar el rendimiento de distintas configuraciones (hacer funcionar miLista bien como ArrayList bien como LinkedList viendo su rendimiento). La variable declarada se crea cuando escribimos miLista = new LinkedList <String> (); o también se puede usar la sintaxis: List <String> miLista = new LinkedList <String> ();

Los métodos de ArrayList en algunos casos definen los métodos abstractos de List, y en otros casos son específicos. Hay que recordar que en List todos los métodos son abstractos por ser una interfaz, aunque no se indique específicamente en la documentación del API de Java. Hay que recordar también que List, por ser una interfaz no tiene constructores y no es instanciable. Al ver la documentación del API nos puede parecer una clase, pero la ausencia de constructor (aparte del propio nombre en el encabezado) delata que no se trata de una clase. [1]

## Interfaces

**Para compensar la falta de herencia múltiple, Java soporta el concepto de *Interface***. Una interfaz se define como el conjunto de mensajes que una clase ofrece a sus clientes (public). En Java, este concepto es ofrecido como un tipo. Por lo tanto, una interfaz Java es un conjunto de declaraciones de métodos, sin implementación. En Java tienen las siguientes características:

* Las Interfaces pueden ser declaradas como public o no (package)
* Todos sus métodos son abstract (no es necesario poner la palabra abstract)
* Todos sus métodos son public (no es necesario poner public)
* Las interfaces no tienen constructores.
* Solo se permiten atributos con public static final.
* Una interface puede heredar de otras interfaces.
* Una clase puede heredar (implementar) **múltiples interfaces**.

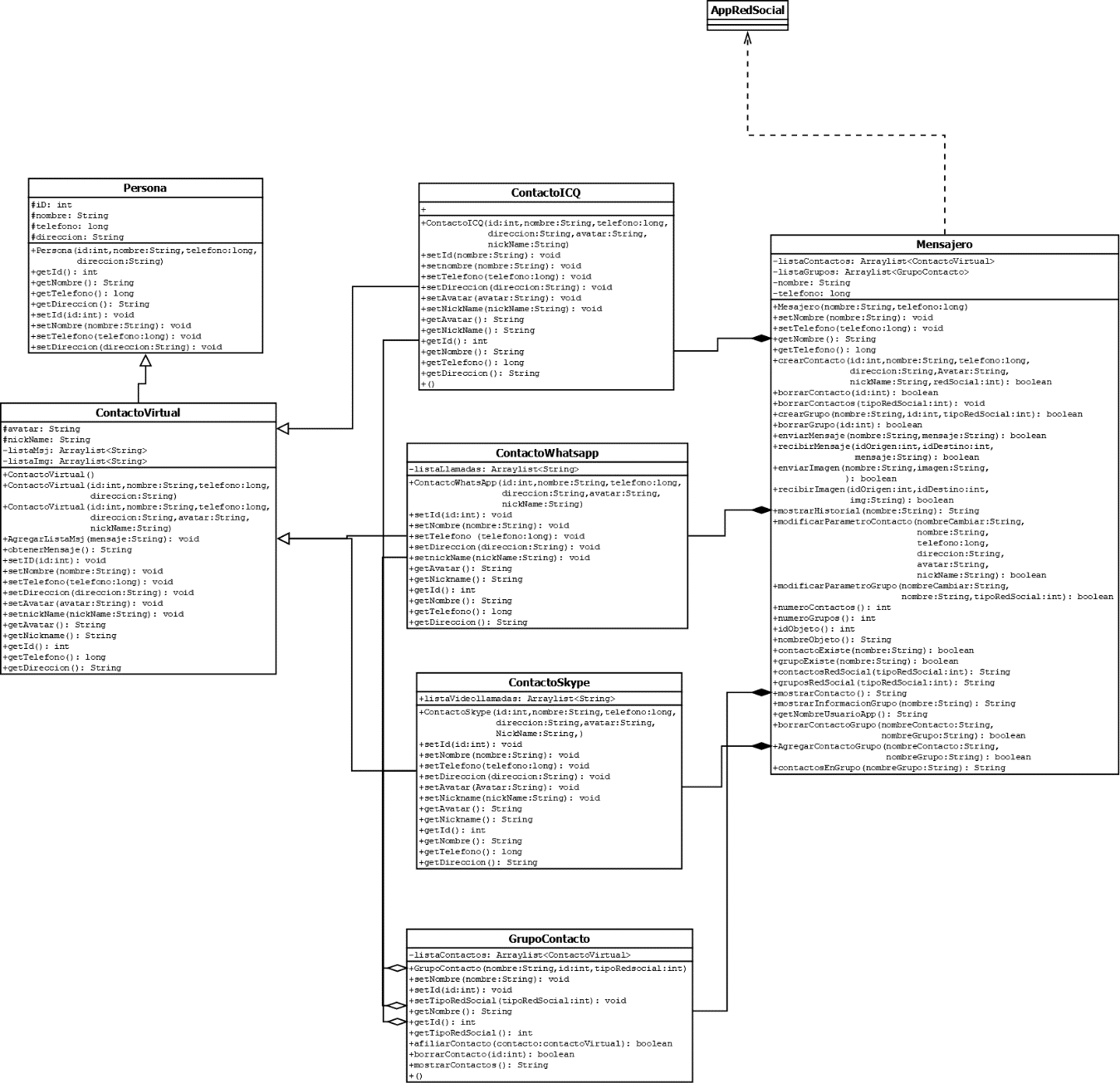
## Diferencia entre las dos

Existen varias diferencias entre una clase abstracta y una interfaz:

1. Una clase abstracta puede heredar o extender cualquier clase (independientemente de que esta sea abstracta o no), mientras que una interfaz solamente puede extender o implementar otras interfaces.
2. Una clase abstracta puede heredar de **una sola** clase (abstracta o no) mientras que una interfaz puede extender varias interfaces de una misma vez.
3. Una clase abstracta puede tener métodos que sean abstractos o que no lo sean, mientras que las interfaces sólo y exclusivamente pueden definir métodos abstractos.
4. En java concretamente (ya que has puesto la etiqueta Java), en las clases abstractas la palabra abstract es obligatoria para definir un método abstracto (así como la clase). Cuando defines una interfaz, esta palabra es opcional ya que se infiere en el concepto de interfaz.
5. En una clase abstracta, los métodos abstractos pueden ser *public* o *protected*. En una interfaz solamente puede haber métodos públicos.
6. En una clase abstracta pueden existir variables static, final o static final con cualquier modificador de acceso (*public*, *private*, *protected* o *default*). En una interfaz sólo puedes tener constantes (public static final).

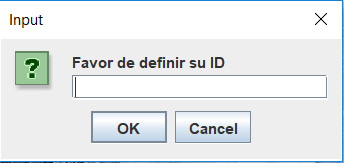
En definitiva, utilizar una u otra depende de las necesidades de cada uno. Las clases abstractas no se utilizan mucho. En cambio, las interfaces se utilizan mucho cuando trabajas con interfaces gráficas, por ejemplo. Uno de los usos más comunes es para crear aplicaciones que hagan uso del concepto de hebras para ejecutar procesos que consumen más tiempo de CPU. Esto se hace mediante la interfaz Runnable de Java, por poner uno de los innumerables ejemplos para los que se utilizan las interfaces. [3]

# Análisis del problema

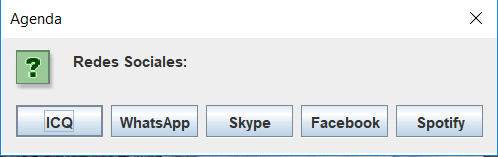


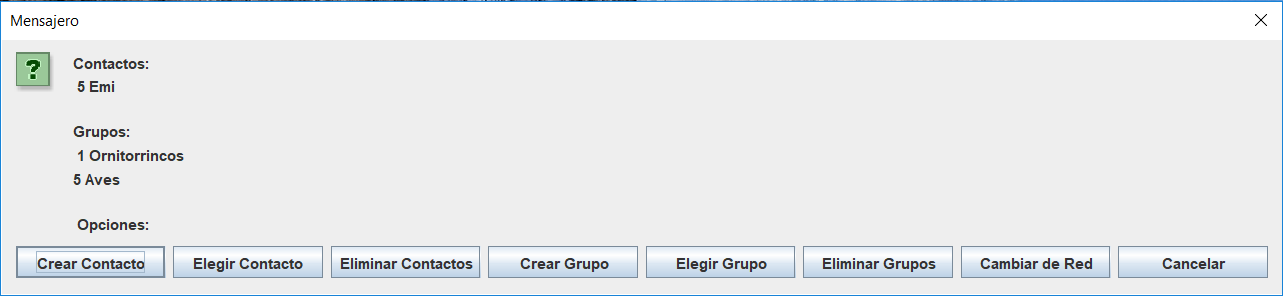
# Pruebas

1.- Aparece una ventana para la creación del perfil del usuario que usará la aplicación

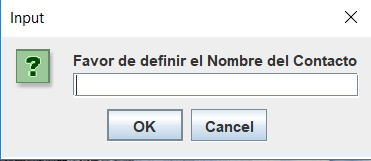


2.- Aparece la red social a la que quiere ser agregado el usuario

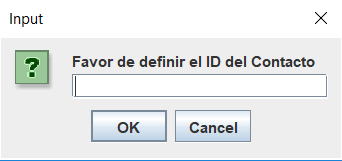


3.-Aparece el menú de los contactos que están creados por default en el directorio del usuario

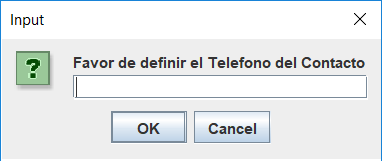
4.-Para un nuevo contacto aparece la ventana donde se deberá definir el nombre de este.



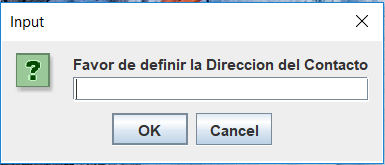
5.- Se deberá definir el ID del contacto nuevo



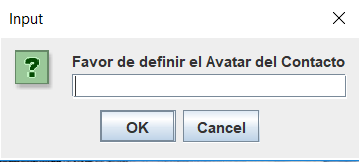
6.- Se le pide ingresar el teléfono del contacto nuevo



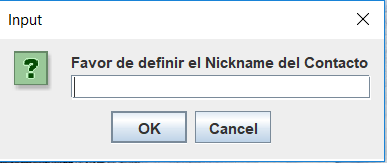
7.- También se le pide la dirección del contacto nuevo



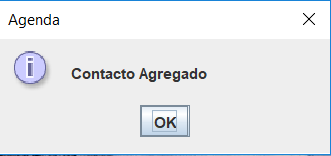
8.- Se define el avatar del contacto que deberá ser un String



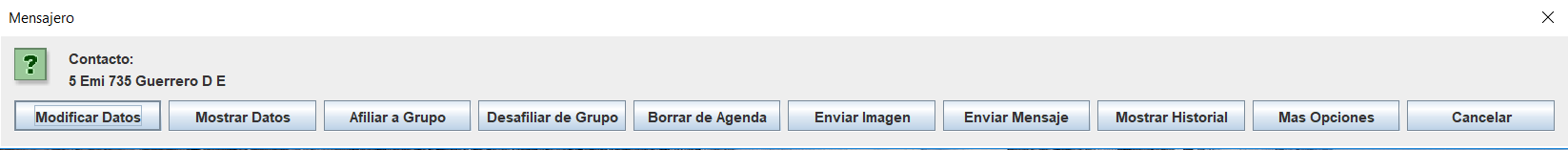
9.- Se define el Nickname del usuario



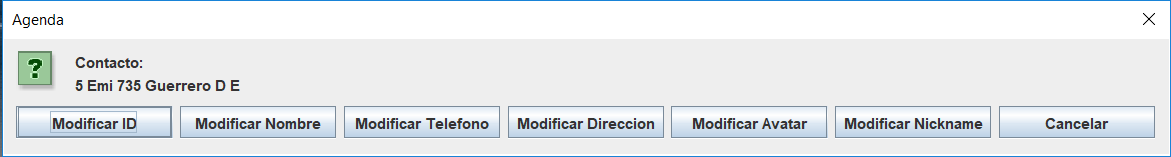
10.- y se muestra una ventana donde dice que el contacto ha sido agregado, por lo cual se guarda en la agenda del usuario



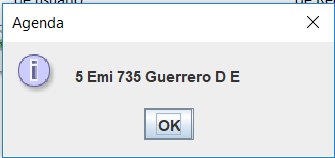
11.- Entonces en el mensajero se muestra el nuevo usuario y una serie de opciones que se puede hacer con ese contacto.



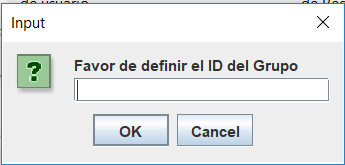
12.- Se puede modificar los datos del nuevo contacto



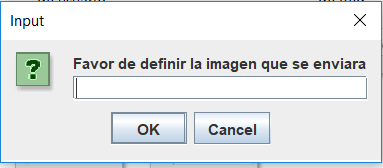
13.- Entonces si se modifica algo aparece la ventana del usuario con los datos cambiados



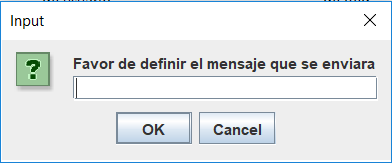
14.- Se puede afiliar a un grupo en específico cada contacto



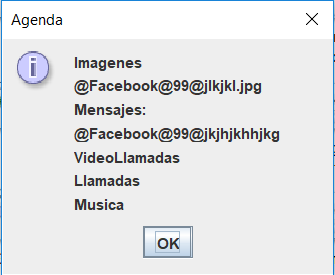
15.- Con el contacto se pueden hacer muchas acciones dependiendo de la red social al que está afiliado.

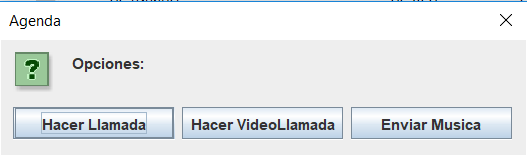


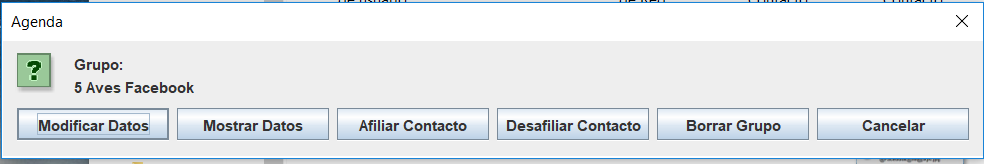
16.- Se pueden enviar los mensajes que se deseen al otro contacto

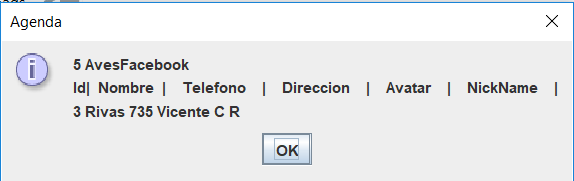


17.- Y se puede mostrar el historial de lo que se ha enviado al usuario



18.- Se pueden hacer otras opciones dependiendo de la red social que se haya elegido

19.- Con los oros grupos también se pueden realizar diversas acciones

20.- Y se muestra los datos de cada grupo que se pueda tener 

# Conclusiones

## Luciano Espina Melisa

Con esta práctica, como ya estaba desarrollada solo fue cosa de cambiar algunas cosas, se complico un poco cuando en la clase Mensajero había muchas funciones las cuales eran instancias de otras clases por lo cual no se podría hacer ya que había clases abstractas y en este tipo de clases no se puede instanciar. En la parte de la interfaz fue un poco más fácil de ver la aplicación ya que como en la investigación realizada se vio que es más útil en el aspecto de que al momento de hacer la configuración para algunos programas de forma gráfica sería más fácil de hacer.

La práctica estuvo larga, pero se logró el objetivo.

## Dávila Dávila García Rivas Emiliano

# Bibliografía

* [1] Alex Rodríguez, Para qué sirven las interfaces Java. Implementar una interfaz del API. Ventajas y ejemplos básicos. (CU00697B) |Aprenderaprogramar.com [Online] Available: <https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=671:para-que-sirven-las-interfaces-java-implementar-una-interfaz-del-api-ventajas-y-ejemplos-basicos-cu00697b&catid=68&Itemid=188>
* [2] Francisco Gala,   
  Programación orientada a objetos: clases abstractas e interfaces |   
  Rootear [Online] Available:   
  <https://rootear.com/desarrollo/poo-clases-abstractas>
* [3] &#191;Cu&#225;l java?,   
  ¿Cuál es la diferencia entre Interfaz y Clase Abstracta en java? |   
  Stack Overflow en español [Online] Available: <https://es.stackoverflow.com/questions/107742/cu%C3%A1l-es-la-diferencia-entre-interfaz-y-clase-abstracta-en-java>