Enunciado práctica 1 (2 sesiones).
PROGRAMACIÓN AVANZADA
PRÁCTICA 1: PROGRAMACIÓN ORIENTADA A OBJETOS

1. Objetivo

En esta primera práctica vas a utilizar los siguientes conceptos fundamentales en Programación Orientada a Objetos (POO): Sobrecarga, Polimorfismo y Herencia. Gran parte del trabajo de esta práctica consiste en definir un gran número de clases. Es por ello, que antes de empezar a programar, lo mejor es hacer un diagrama de las clases que vas a necesitar, de las relaciones entre ellas, así como de sus datos y métodos. Este pequeño esfuerzo inicial ten por seguro que te evitará tener que reescribir mucho código.

Para automatizar la introducción de datos en el programa, y que ésta no resulte una labor tediosa cada vez que se quiere realizar una nueva prueba, en tu programa harás uso de una biblioteca. Dicha biblioteca permite generar automáticamente algunos datos personales como: nombre y apellidos de personas, NIFs, provincias y poblaciones.

Para mejorar la calidad de tu código, te pedimos que programes una serie de test que permitan verificar el funcionamiento correcto de tu aplicación. Debes generar test unitarios para todas aquellas funciones del programa que no sean triviales. No tengas miedo a reescribir código. Si sientes la necesidad de reescribir código para mejorarlo significa que estás en el buen camino. Aunque seas un programador experto duda de que estás en el buen camino si no cambias con frecuencia tu código para mejorarlo.

Como ya sabes, todas las prácticas están orientadas a conseguir una única aplicación final, que es una aplicación de facturación. Dicha aplicación debe permitir realizar una gestión de los clientes de una compañía de telefonía móvil. Tu aplicación deberá mantener una relación de clientes. Para cada cliente, a su vez, existirá una colección de las llamadas que ha realizado y otra colección con las facturas que se le han emitido. La facturación se realizará teniendo en cuenta las llamadas y el tipo de tarifa que el cliente tenga contratada.

2. Enunciado

Como hemos dicho, vas a desarrollar una aplicación de facturación para una empresa de telefonía móvil, la cual tiene una cartera de clientes. Los clientes pueden ser de dos tipos, o bien empresas, o bien particulares. Cada cliente tiene asignada una tarifa, que inicialmente codificarás como un valor numérico representando el precio por segundo, aunque ésto es algo que evolucionará en posteriores prácticas. De cada uno de los clientes se conoce su nombre, NIF, dirección, correo electrónico, fecha de alta y tarifa (€/min). Además, si el cliente es un particular también se conocen sus apellidos. En las direcciones figura: código postal, provincia y población.

Dado que, como te hemos dicho, la forma de codificar las tarifas va a evolucionar en prácticas posteriores, será una buena idea definir una clase Tarifa. Inicialmente la clase sólo contendrá un dato numérico, pero en un futuro contendrá algo diferente. No obstante, si

codificas la tarifa en forma de clase, los cambios futuros que tendrás que hacer serán menores.

Los clientes efectúan llamadas, por lo que habrá que guardar una relación de todas las llamadas efectuadas. Para cada llamada guardaremos el número de teléfono al que se llamó, la fecha y hora de la llamada y la duración.

Cada cliente tiene un conjunto de facturas, cada factura tiene asignada la tarifa en el momento de emitir la factura, aunque un cliente puede cambiar de tarifa cuando lo desee. Por simplificar, el cambio de tarifa se verá reflejado en la siguiente factura. Cada factura tiene asociados: un código único que no puede poseer ninguna otra factura, la tarifa aplicada (€/min), la fecha de emisión de la factura, el periodo de facturación (de qué fecha a qué fecha) y el importe de la misma. El importe de la factura se calcula a partir de la suma de minutos de las llamadas que ha efectuado el cliente durante el periodo de facturación, y de la tarifa.

De cara a la sesión que dedicaremos a Genericidad, te pedimos que las clases Cliente, Factura y Llamada dispongan de un método llamado getFecha(). En el caso de los clientes, este método retornará la fecha de alta del cliente, mientras que en el caso de las facturas, devolverá la fecha de emisión de la factura, y en el caso de las llamadas, devolverá la fecha en la que ésta se efectuó.

3. Metodología

Esta práctica se desarrollará en dos sesiones. Intenta abstraer toda la información que tienes en la Sección 2. Intenta relacionar unas abstracciones con otras, determina la cardinalidad de las relaciones. ¿Puedes encontrar abstracciones que pueden relacionarse mediante herencia? ¿Puedes encontrar abstracciones que comparten un interfaz común? Programa también una clase que muestre los distintos menús por consola y que te permita elegir las distintas acciones que puede realizar la aplicación.

Debes desacoplar tanto la entrada de datos, que en principio será el teclado, como la salida de datos, que en principio será la consola, del resto del sistema. De este modo, cuando añadamos una interfaz gráfica de usuario a la aplicación, los cambios que tendrás que realizar serán mínimos. Ya sabes que una buena práctica es buscar código repetido e intentar eliminar la repetición utilizando polimorfismo.

Con respecto a las acciones que se pueden llevar a cabo en la aplicación y de las que deberá existir alguna opción de menú, para los clientes se deberá poder:

Dar de alta un nuevo cliente.
Borrar un cliente.
Cambiar la tarifa de un cliente.
Recuperar los datos de un cliente a partir de su NIF.
Recuperar el listado de todos los clientes.
Con respecto a las llamadas:

Dar de alta una llamada.
Listar todas las llamadas de un cliente.
Con respecto a las facturas:

Emitir una factura para un cliente, calculando el importe de la misma en función de las llamadas.
Recuperar los datos de una factura a partir de su código.
Recuperar todas las facturas de un cliente.
Para generar datos de manera automática, tales como nombres, apellidos, etcétera, relativos a los clientes vas a utilizar una biblioteca java que encontrarás en el aulavirtual. Tal y como hemos comentado, esta biblioteca genera datos personales. La idea es que en lugar de tener que introducir de manera manual en cada prueba de tu código dichos datos, tomes los datos que te proporciona dicha biblioteca. Esto hará que el proceso de pruebas sea más ágil.

4. **Uso de la biblioteca para la generación de datos aleatorios**

```
<repositories>
	<repository>
		<id>jitpack.io</id>
		<url>https://jitpack.io</url>
	</repository>
</repositories>
<dependencies>
	<!--Biblioteca para generar datos aleatorios-->
	<dependency>
		<groupId>com.github.TallerIngenieriaDelSoftware</groupId>
		<artifactId>GeneradorDatosINE</artifactId>
		<version>v1.0.0</version>
	</dependency>
<dependencies>
```

5. **Fuentes de información**

Big Java Capítulos 2, 8 y 9.
Desarrollo de proyectos informáticos con tecnología Java Capítulo 3.
JUnit Página web de JUnit. Allí encontrarás los javadocs de este framework.