Bloque práctico 2: *Java*

1: Programación modular, herencia simple y polimorfismo.

En esta práctica utilizaremos las capacidades de programación orientada a objetos de *Java*. El objetivo de la práctica es programar una aplicación informática en *Java* para una empresa de telefonía móvil que desea informatizar la facturación de sus clientes.

Todas las clases asociadas al problema serán programadas siguiendo una filosofía modular, realizando un paquete con todas las clases a utilizar (Fecha, Cliente, Grupo,...) y otro paquete con las clases de prueba de las clases.

0) Recordatorio: la programación modular en *Java*

Para programar una clase dentro de un paquete, deberemos incluir en el archivo donde se crea esta clase la línea *package nombredepaquete.* Para esta práctica, el nombre del paquete de librería debe ser *libClases* y el de las clases de prueba *libPruebas*

En las clases de *libPruebas* se deberá utilizar *import* para referenciar a las clases del paquete *libClases*.

Ejemplo de clases en un paquete y compilada en un módulo

```
package libClases;

public class Cliente {
  private String nombre;
  private final Fecha fechaContrato;
  private static int cuenta=0;
  public String getNombre( ){
    return nombre;
  }
  static {
   //este bloque de codigo solo se
   //ejecuta una vez cuando se cree
   //el primer objeto Cliente o se
   //invoque el 1er método estatico
   System.out.println("Primero!!");
  }
```

```
package libPruebas;
import libClases.*;

public class Prueba {
public static void main (String [ ] args) {
   Cliente c = new Cliente ( );
   …
   System.out.println("Nombre: " +
                              c.getNombre( ));
 }
}
```

El paquete *libClases* deberá estar en un directorio */libClases* para ser accesible desde *libPruebas.*

Planteamiento del Problema

Una empresa de telefonía móvil desea informatizar la facturación de sus clientes. La compañía tiene dos tipos de clientes

- **clientes tarifa plana:** Estos clientes pagan una cantidad fija al mes, independientemente de la cantidad de llamadas realizadas y el número de minutos hablados hasta un cierto límite. Si el cliente supera el límite de minutos, el exceso de minutos se tarificará a un precio fijo de 0.15 céntimos/minuto.

  factura = precio tarifa plana + (exceso limite minutos x 0.15)

  Por cuestiones de estudio de mercado, a la compañía le interesa saber la nacionalidad de este tipo de clientes (cree un atributo de tipo String en este clase concreta para guardar la nacionalidad del cliente).

- **clientes móvil:** Estos clientes pagan en función del número de minutos que hablan al mes. **Estos clientes tienen Permanencia (atributo adicional de tipo Fecha)**

  factura = precio minuto x minutos hablados

Actualmente la oferta que tiene la compañía para los clientes de tarifa plana es de 300 minutos por 20 euros (programar la clase de forma que cuando la compañía actualice la oferta de tarifa plana los cambios se reflejen en todos los clientes actuales y futuros de la compañía). En cuanto a los clientes de tarifa móvil el precio por minuto es específico y particular para cada cliente.

Para cada cliente, independientemente del tipo que sea, la empresa desea guardar la siguiente información:

dni: dni del cliente (incluida la letra) que lo identifica de forma única.

codCliente: número **entero único** que identifica al cliente que genera automáticamente la aplicación (cada cliente tiene un número distinto **que no puede cambiar**)

nombre: nombre y apellidos del cliente.

fechaNac: fecha de nacimiento (dd/mm/aa) del cliente. **Este campo es un objeto de tipo *Fecha*** (clase que permite trabajar con Fechas).

fechaAlta: fecha (dd/mm/aa) en la que el cliente se da de alta en la compañía. **Este campo es un objeto de tipo *Fecha*** (clase que permite trabajar con Fechas).

Programar una clase base llamada ***Cliente*** para representar a un cliente genérico. A partir de dicha clase y mediante herencia programar las clases derivadas ***ClienteTarifaPlana*** y ***ClienteMovil***.

1) Programar las clases *Fecha*, *Cliente y sus clases hijas*

Programe una clase *Fecha*, que no pueda tener clases derivadas de forma que sea una clase mutable (posee métodos públicos *set* que permiten modificar el valor de sus atributos). **Al ser mutable, los métodos *getFecha( )* de la clase *Cliente* NO DEBEN devolver directamente la referencia del objeto *Fecha* que contienen sino una copia**. La clase *Fecha* debe estar implementada de forma que permita ejecutar el siguiente *main( )*:

```
public static void main(String[] args) {
  Fecha f1 = new Fecha(29,2,2001), f2 = new Fecha(f1), f3 = new Fecha(29,2,2004);
  final Fecha f4=new Fecha(05,12,2023);  //es constante la referencia f4
  System.out.println("Fechas: " + f1.toString() + ", "+f2+ ", " +f3+ ", " +f4+ "\n");
  f1=new Fecha(31,12,2016);  //31/12/2016
  f4.setFecha(28, 2, 2008);              //pero no es constante el objeto al que apunta
  System.out.println(f1 +" "+ f2.toString() +" " + f3 +" "+ f4+" "+ f1);
  f1=new Fecha(f4.getDia()-10, f4.getMes(), f4.getAnio()-7);      //f1=18/02/2001
  f3=Fecha.pedirFecha(); //pide una fecha por teclado
  if (f3.bisiesto() && Fecha.mayor(f2,f1))
    System.out.print("El " + f3.getAnio() + " fue bisiesto, " + f1 + ", " + f3);
}
```

Salida:

```
Fechas: 28/02/2001, 28/02/2001, 29/02/2004, 05/12/2023
31/12/2016 28/02/2001 29/02/2004 28/02/2008 31/12/2016
Introduce Fecha (dd/mm/aaaa): 29/02/2001
Fecha no valida
Introduce Fecha (dd/mm/aaaa): 01/01/2000
El 2000 fue bisiesto, 18/02/2001, 01/01/2000
```

Nota: El constructor fecha debe admitir cualquier año como válido, si el día y/o el mes no es válido se asigna por defecto el día y/o mes más cercano al indicado.
Ej: 29/2/2001 → **28**/2/2001, 33/0/2002 → **31**/**1**/2002, 0/14/2004 → **1**/**12**/2004, 31/09/2007 → **30**/09/2007, 29/2/2100 → **28**/2/2100 (el año 2100 no va a ser bisiesto)

A continuación programe una clase *Cliente*, en la que los campos ***nif***, ***codCliente*** y ***fechaNac*** deben ser constantes (sus valores no pueden cambiar una vez asignados) y en la que la propia clase asigna un código único de forma automática a cada cliente creado:
final declara constante la referencia no el objeto → si la clase es inmutable no se puede cambiar el contenido, si es mutable podemos evitar que se pueda cambiar el contenido no proporcionando métodos set publicos

Cliente.java

```
package libClases;

public class Cliente {
  private final String nif;       //dni del cliente (letra incluida) (NO puede cambiar)
  private final int codCliente;   //codigo único (y fijo) generado por la aplicación
  private String nombre;          //nombre completo del cliente (SI se puede modificar)
  private final Fecha fechaNac;   //fecha nacimiento del cliente (NO se puede cambiar)
  private final Fecha fechaAlta;  //fecha de alta del cliente (SI se puede modificar)

  public Cliente (String NIF, String nom, Fecha fNac, Fecha fAlta); //constructor
  public Cliente (String NIF, String nom, Fecha fNac);              //constructor
  public String toString(); //devuelve una cadena con la información del cliente
}
```

Añade a *Cliente* métodos públicos *getNombreAtributo( )* y *setNombreAtributo( )* que permitan consultar y modificar (si es posible) sus atributos privados.

En el caso del constructor en el que no se indica la fecha de alta, la clase *Cliente* debe establecer una fecha de alta que por defecto debe ser el 01/01/2018. Añade a la clase *Cliente* los **atributos** y **métodos necesarios** para que se permita **consultar** y **modificar** la fecha de alta que por defecto se asigna cuando ésta no se indica.

Finalmente, programe en Java las clases *ClienteTarifaPlana* y *ClienteMovil*, de forma que hereden de la clase *Cliente*.

Consideraciones a tener en cuenta:

Tanto la clase base como las clases derivadas deben tener los atributos declarados como privados y por tanto deben proporcionar en sus respectivas interfaces métodos *get* para poder acceder a los atributos privados y métodos *set* para poder modificarlos.

Todas las clases deben sobrecargar el método público *toString( )* que por defecto heredan de `Object` **(véase apuntes Java página 24 a 26)** para que devuelvan en una cadena de caracteres (*String*) la información de todos los campos (atributos) de la clase.

Haz que la clase `Fecha, Cliente` y sus clases derivadas implementen la interfaz ***Cloneable*** (**véase apuntes Java página 27 a 32**) y la interfaz ***Proceso*** (**véase apuntes Java página 33 a 34**) definida a continuación:

Proceso.java

```
package libClases;

public interface Proceso {
  public abstract boolean equals(Object obj);  //true sin son iguales
  void ver();                                  //muestra en pantalla el objeto
}
```

Nota:

- **Cada clase debe tener los métodos *get* y *set* exclusivos de sus atributos** (ej: la clase Cliente no debe tener *getMinutos() o getPrecio()* ya que eso pertenece a las clases hijas).

- **Los nombres de la clase y métodos deben ser los indicados en negrita** en el enunciado.

- Consideramos que **2 clientes son iguales si tienen el mismo *nif* y son del mismo tipo** y **2 fechas son iguales si son exactamente idénticas**.