Actividad 1. Proyectos de programación. Juan José Sorlí Castelló.

Se ha definido la clase Usuario la cual es abstracta. De ella heredarán diferentes clases para implementar diferentes tipos de usuarios con distintos roles. Está la clase Administrador, que ya se ha visto en el enunciado que va a tener permiso para realizar más gestiones con la base de datos que otras clases que implementen otros tipos de usuario.

```java
/**
 * Clase abstracta que no implementa ningun metodo ya que, de ella
 * heredaran distintos tipos de usuario que implementaran de diferente
 * manera los metodos aqui listados aprovechando el polimorfismo
 *
 * @author sorli
 */
public abstract class Usuario {
    protected String DNI;
    protected String password;
    protected String tfno;

    //Constructor de la clase Usuario
    protected Usuario (String dni, String pwd, String tfn){
        this.DNI = dni;
        this.password = pwd;
        this.tfno = tfn;
    }

    //Para favorecer encapsulado habilito metodos para acceder a atributos privados y setearlos
    protected abstract String getTfno ();

    //Sobrecargo el metodo con diferentes implementaciones en funcion del parametro que le pase
    protected abstract String getTfno (String dni);

    protected abstract void setTfno (String tfn);
```

Ilustración 1: Clase Usuario

A efectos prácticos, esa diferencia en la gestión de las peticiones a la base de datos, o a las funciones a las que cada perfil esté autorizado a ejecutar en general, en función del rol se llevarán a cabo mediante el polimorfismo visto en clase. Así un objeto de la clase Administrador podrá implementar ciertas *queries* por ejemplo sobre la base de datos y otro de la clase Cliente, (también subclase de la clase abstracta Usuario), mediante el polimorfismo lanzará una excepción, (también visto en clase), en el mismo método si el rol no está autorizado a hacer esas consultas.

```java
protected void creaOperacion(Operacion operacion) {
    throw new UnsupportedOperationException("Esta operación no está soportada por este Usuario. "
            + "Según el enunciado de la actividad de proyectos de Programación se requiere ser administrador.");
}
```

Ilustración 2: Ejemplo de lanzamiento de excepción en el código

Otro ejemplo de entidades que se están trabajando dentro del paradigma de la POO es la clase Operación la cual, como se comenta en el enunciado es la base de todo el flujo de información. Esta clase sí que se ha decidido implementarla frente a hacerla abstracta o una interfaz porque se ha considerado que una implementación para esta clase tenía sentido.

```java
/**
 * Operacion es la base de todo. Algunas clases como Facturacion heredaran
 * de ella y añadiran atributos y metodos de la transaccion economica vinculada
 * al trafico de mercancias de cada Operacion
 * No la voy a hacer abstracta ni interfaz ya que creo que tiene mas sentido
 * con implementacion en sí misma
 * @author sorli
 */
public class Operacion {

    //Una operacion puede tener muchas rutas asociadas
    //Ruta es una clase abstracta de la que heredan las rutas segun
    //sean terrestres, aereas o maritimas
    List <Ruta> listaRuta = new ArrayList <>();

    //Constructor
    public Operacion(){

    }

    protected void anadeRuta(Ruta ruta){
        listaRuta.add(e: ruta);
    }

}
```

Ilustración 3: Clase Operacion la cual puede constar de varias rutas terrestres, marítimas y/o aéreas.

Sin embargo, respecto de esta clase a la que se añaden las distintas rutas y la información en general del flujo de mercancías vinculado a esta operación, se ha hecho que otras clases, como Facturación, hereden de ella, ésta última por ejemplo para añadir la funcionalidad concreta de las transacciones monetarias vinculadas a la operación en cuestión.

Finalmente, el otro gran grupo de entidades que se ha trabajado en esta actividad han sido las que contienen información de las rutas. Se ha creado la clase Ruta como una clase abstracta ya que se considera que una Ruta no tiene sentido en sí misma. Es sólo el resultado de aglutinar características comunes de las rutas marítimas, terrestres o aéreas pero nunca va a ser necesario que exista en el código un objeto de la clase código como tal.

```java
package proyectologistica;

/**
 * La clase Ruta la hago abstracta porque no tiene
 * sentido que cree objetos de ella ni que defina su implementacion
 * Simplemente la he hecho porque agrupa cosas comunes de las distintas
 * rutas, (terrestres, aéreas y marítimas), que son las que si que pueden ocurrir
 * en la realidad
 * @author sorli
 */
public abstract class Ruta {
    protected String ciudadOrigen;
    protected String ciudadDestino;
    protected String paisOrigen;
    protected String paisDestino;

    protected Ruta(){

    }

    //Sobrecargo el constructor
    protected Ruta (String ciudadOrg, String ciudadDest, String paisOrg, String paisDest){
        this.ciudadOrigen = ciudadOrg;
```

Ilustración 4: Clase abstracta Ruta de la que heredaran otras clases para añadir el detalle y la concreción de los transportes marítimo, aéreo y/o terrestre.

Las clases RutaAerea, RutaTerrestre y RutaMaritima heredan, por tanto, todas ellas de la clase abstracta Ruta y, por un lado añaden parámetros característicos de cada una de las rutas en forma de atributos, (la ruta marítima tiene un booleano para indicar si el contenedor es open top o no mientras que la ruta aérea tiene otro para indicar si el bulto viaja de pie), y por el otro utilizan el polimorfismo para que cada método de la clase abstracta Ruta se implemente de forma diferente según si el trayecto es aéreo, marítimo o terrestre.

```java
package proyectologistica;

/**
 * Esta clase hereda de la clase abstracta Ruta
 * @author sorli
 */
public class RutaMaritima extends Ruta {


    private String nombreBarco;
    private boolean canalPanama;
    private boolean canalSuez;
    private int cantidadContenedores;
    private boolean openTop;

    protected RutaMaritima(){
        super();
    }

    protected RutaMaritima(String ciudadOrg, String ciudadDest, String paisOrg, String paisDest){
        super(ciudadOrg, ciudadDest, paisOrg, paisDest);
        this.canalPanama = false;
        this.canalSuez = false;
        this.openTop = false;
    }
```

Ilustración 5: RutaMaritima añade detalles propios del transporte marítimo a su superclase, la clase abstracta Ruta