

**13.** En un sistema Java que gestiona los usuarios de un servicio telemático se utiliza una clase Usuario:

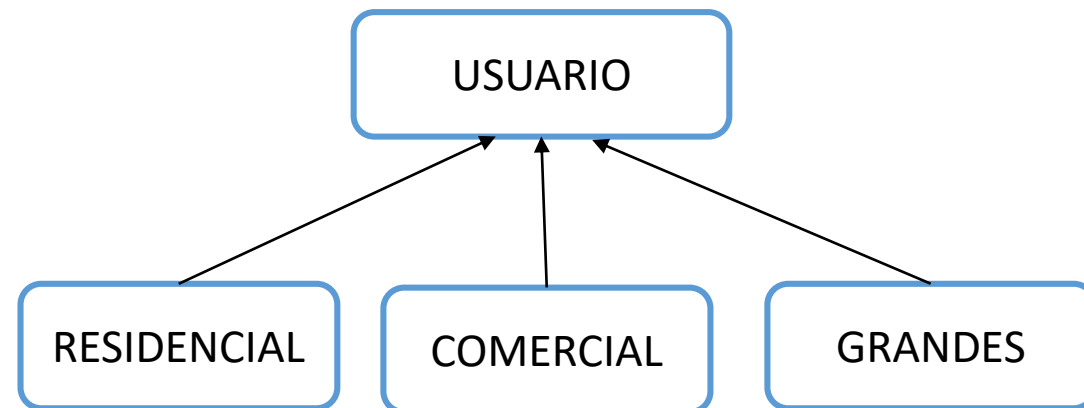
```
class Usuario {  
    private String nombre, dni;  
    private int cuenta;  
    Usuario (String d, String n)  
        { nombre = n;  
          dni = d;  
          cuenta = 0;}  
    void conexion (int s) // Contabiliza "s" segundos en la cuenta  
        { cuenta = cuenta + s;}  
    double calculaFacturación() //Calcula el importe facturable  
        { return cuenta * 0,32; }  
    void reset() // Pone a cero la cuenta  
        { cuenta = 0;}  
}
```

El departamento de marketing ha diseñado un conjunto de ofertas no acumulables para los diferentes tipos de usuario con las siguientes condiciones:

- Oferta1 (para usuarios residenciales): No se contabilizan los tres primeros minutos de cada conexión del cliente.
- Oferta2 (para usuarios comerciales): Se aplica un tanto por ciento de descuento sobre el importe facturable total. Este tanto por ciento se negocia por separado con cada cliente, pero una vez fijado no es modificable.
- Oferta3 (para grandes usuarios): No se contabiliza la conexión más larga de cada periodo de facturación.


Elaborar las clases necesarias para resolver las nuevas necesidades del sistema considerando que no podemos modificar la clase USUARIO.

- Usuario residencial **es un** usuario
- Usuario comercial **es un** usuario
- Gran usuario **es un** usuario








Oferta1 (para usuarios residenciales): No se contabilizan los tres primeros minutos de cada conexión del cliente

```
class Residencial extends Usuario{  
    Residencial (String d, String n)  
        { super(d,n); }  
    double calculaFacturación() //Calcula el importe facturable  
        { return (cuenta - 160) * 0,32; }  
}
```








**CORREGIR!!!!**

- Oferta2 (para usuarios comerciales): Se aplica un tanto por ciento de descuento sobre el importe facturable total. Este tanto por ciento se negocia por separado con cada cliente, pero una vez fijado no es modificable.

```
class Comercial extends Usuario{  
    private int descuento;   
    Comercial (String d, String n, int desc)  
    { super(d,n);   
      descuento = desc; }   
    double calculaFacturación() //Calcula el importe facturable  
    { double desc = (cuenta * 0,32) * descuento / 100;  
      return (cuenta * 0,32) - desc; }   
    } 
```

Oferta3 (para grandes usuarios): No se contabiliza la conexión más larga de cada periodo de facturación

```
class GranUsuario extends Usuario{  
    private int maxcon;   
    GranUsuario (String d, String n)  
    { super(d,n);  
      maxcon = 0;}   
    void conexion (int s) // Contabiliza "s" segundos en la cuenta  
    { cuenta = cuenta + s;  
      if (s > maxcon)   
        maxcon = s;}   
    double calculaFacturación() //Calcula el importe facturable  
    { return (cuenta - maxcon)* 0,32; }  
}
```



Todos los boletos de avión de cabotaje incluyen origen, destino, nombre y dni el pasajero, fecha de viaje, importe y la aerolínea (LAN, Aerolíneas, etc.). Si el boleto es de primera clase se permiten cambios de fechas o destino. Si el boleto es Turista también, pero tiene un recargo. Los boletos de la clase económica no admiten ningún tipo de cambio.

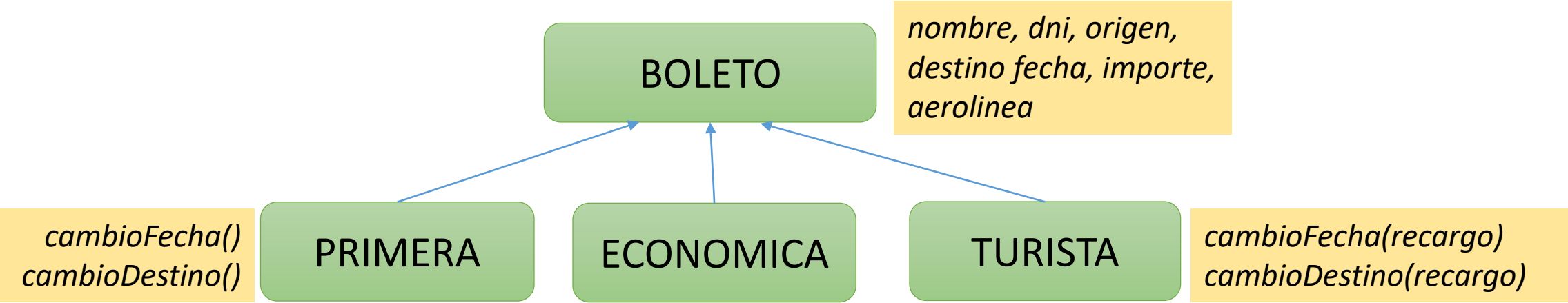
Una agencia de viajes tramita boletos de avión de cabotaje de diversas aerolíneas. Por eso necesita los siguientes servicios:


- Total de boletos vendidos de una determinada aerolínea
- Total recaudado por cada tipo de boleto.
- Buscar el boleto de un determinado pasajero por dni.
- Listar los boletos de clase económica de una determinada aerolínea
- Listar los boletos de clase turista a un terminado destino.

Se pide.

1. Modelar el problema
2. Implementar el modelo
3. Implementar un menu que permita interactuar con la agencia.

Todos los boletos de avión de cabotaje incluyen origen, destino, nombre y dni del pasajero, fecha de viaje, importe y la aerolínea (LAN, Aerolíneas, etc.). Si el boleto es de primera clase se permiten cambios de fechas o destino. Si el boleto es Turista también, pero tiene un recargo. Los boletos de la clase económica no admiten ningún tipo de cambio.





```
class Boleto {  
    String origen, destino, nombre, fecha, aerolínea;  
    float precio;  
    int dni;  
    Boleto (.....)  
        { .....}
```

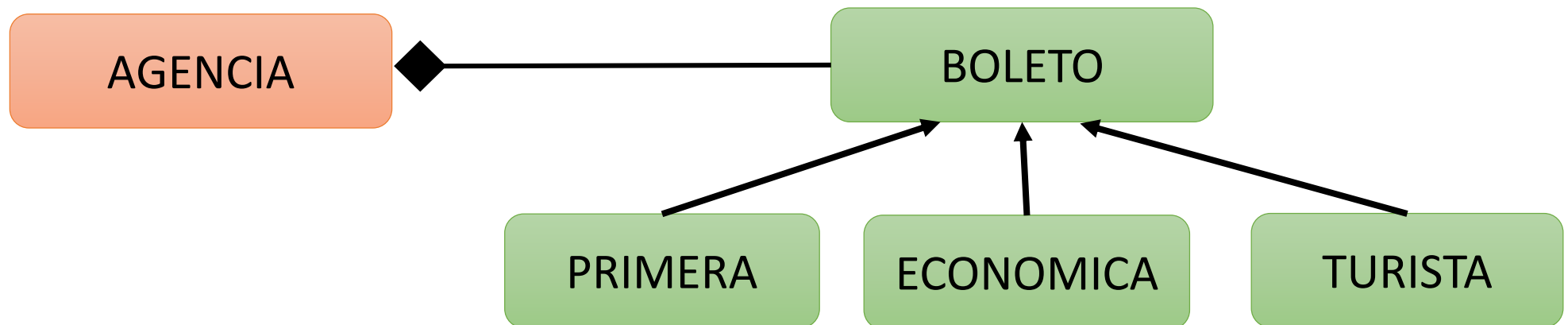
**Set/get**

```
class Turista extends Boleto {  
    Turista (.....)  
        { super...}  
  
    void cambiarFecha(String cambio, doublé recargo)  
    { fecha=cambio;  
      importe = importe+ recargo;}  
    void cambiarDestino(String cambio, doublé recargo)  
    { destino=cambio;  
      importe = importe+ recargo;}
```



Una agencia de viajes tramita boletos de avión de cabotaje de diversas aerolíneas. Por eso necesita los siguientes servicios:

- Total de boletos vendidos de una determinada aerolínea
- Total recaudado por cada tipo de boleto.
- Buscar el boleto de un determinado pasajero por dni.
- Listar los boletos de clase económica de una determinada aerolínea
- Listar los boletos de clase turista a un terminado destino.



# Qué comportamiento (métodos) tiene el gestor Agencia?

```
class Agencia {  
    Vector pasajes;  
    Agencia(.....)  
        { pasajes = new Vector();}  
    void agregarPasaje(Boleto b)  
        {pasajes.addElement(b);}
```

**Set/get**

**Total de boletos vendidos de una determinada aerolínea**

```
int totalVendidos(String aero)  
{    int total = 0;  
    Boleto b;  
    for (int i=0; i<pasajes.size(); i++)  
    { b= (Boleto) pasajes.elementAt(i);  
      if (aero.equals(b.getAerolinea()))  
          total++;  
    }  
    return total;  
}
```

# Qué comportamiento (métodos) tiene el gestor Agencia?

```
class Agencia {  
    Vector pasajes;  
    Agencia(.....)  
        { pasajes = new Vector();}  
    void agregarPasaje(Boleto  
        {pasajes.addElement(b);}
```

**Set/get**

Total recaudado por cada tipo de boleto

```
void totalRecaudado()  
{double pc=0,tur=0, ec=0;  
  for (int i=0; i<pasajes.size(); i++)  
  { if (pasajes.elementAt(i) instanceof Primera)  
      pc++;  
    else  
      if (pasajes.elementAt(i) instanceof Turista)  
          tur++;  
      else ec++;  
    S.O.P.  
  }  
}
```

```
class Agencia {  
    Vector pasajes;  
    Agencia(.....)  
        { pasajes = new Vector();}  
    void agregarPasaje(Boleto b)  
        {pasajes.addElement(b);}
```

**Set/get**

**Buscar el boleto de un determinado pasajero por dni**

```
Boleto buscarBoleto(int dni)  
{    boolean enc=false  
    int i=0; Boleto b =null;  
    while (i<pasajes.size() && !enc)  
    { b= (Boleto) pasajes.elementAt(i);  
      if (b.getDNI()==dni)  
          enc= true;  
      else  
          i++;  
    }  
    return b;  
}
```

```
class Agencia {  
    Vector pasajes;  
    Agencia(.....)  
        { pasajes = new Vector();}  
    void agregarPasaje(Boleto b)  
        {pasajes.addElement(b);}
```

**Set/get**

Listar los boletos de clase económica de una determinada aerolínea

```
void listarEconomicaAerolinea (String aero)  
{  
    Economica e;  
    for (int i=0; i<pasajes.size(); i++)  
        if (pasajes.elementAt(i) instanceof Economica)  
        {  
            e = (Economica) pasajes.elementAt(i);  
            if (e.getAerolinea().equals(aero))  
                S.O.P.  
        }  
}
```

```
class Agencia {  
    Vector pasajes;  
    Agencia(.....)  
        { pasajes = new Vector();}  
    void agregarPasaje(Boleto b)  
        {pasajes.addElement(b);}
```

**Set/get**

Listar los boletos de clase turista a un determinado destino

```
void listarTuristaDestino (String dest)  
{  
    Turista t;  
    for (int i=0; i<pasajes.size(); i++)  
        if (pasajes.elementAt(i) instanceof Turista)  
        {   t = (Turista) pasajes.elementAt(i);  
            if (t.getDestino().equals(dest))  
                S.O.P.  
        }  
}
```

# Recomendaciones

- En relaciones de herencia, verificar la correcta aplicación de **“es un”**
- En clases POJOS, **no leer datos** (console..etc), los datos vienen por parámetros
- En clases POJOS, **no desplegar mensajes** (SOP), los datos se devuelven por métodos “get”.
- En gestores **no leer datos** (console..etc), los datos vienen por parámetros
- En gestores, los métodos que calculan valores, **deben devolver valores**. NO usar void + SOP.
- Solo desplegar mensajes en métodos **listadores**, pero use opciones alternativas (vectores o cadena)
- Usar identificadores adecuados para clases/atributos/métodos
- Para diferenciar distintos tipos de objetos en un jerarquía de herencia esta **prohibido** usar atributos “tipo”. **NO HAY POLIMOFISMO**