

Programación 2

**Tecnatura en Desarrollo de Aplicaciones
Informáticas**

Adicional Librería

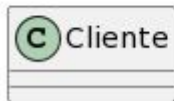
Una librería requiere un sistema para sus clientes.

La librería vende libros y revistas. Un libro se compone de un nombre, autor (solo uno), precio, cantidad de páginas, un resumen y géneros literarios que abarca (puede ser más de uno). Una revista posee nombre, autor, cantidad de páginas, precio, resumen y un listado de géneros. Del cliente se guarda su nombre y apellido, dni y dirección, una lista de autores favoritos y una lista de géneros que le gustan. También se guarda el listado de sus compras (sean libros o revistas).



Reconocer las clases

— — —



Agregar Atributos a las clases

— — —

Libro

```
private String nombre  
private String autor  
private double precio  
private int cantPaginas  
private String resumen  
private ArrayList<String> generos
```

Revista

```
private String nombre  
private String autor  
private double precio  
private int cantPaginas  
private String resumen  
private ArrayList<String> generos
```

Cliente

```
private String nombre;  
private String apellido;  
private int dni;  
private String direccion;  
private ArrayList<String> autores;  
private ArrayList<String> generos;  
private ArrayList<Libro> comprasL;  
private ArrayList<Revista> comprasR;
```

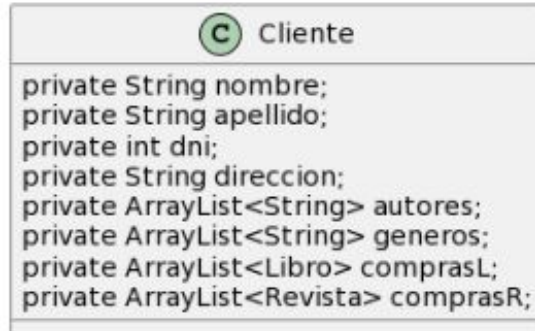
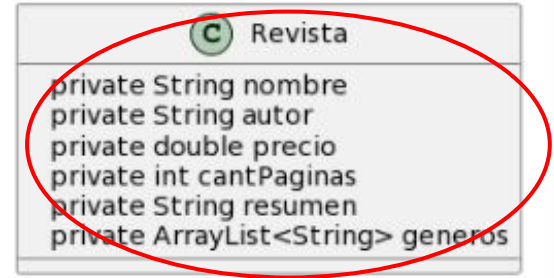
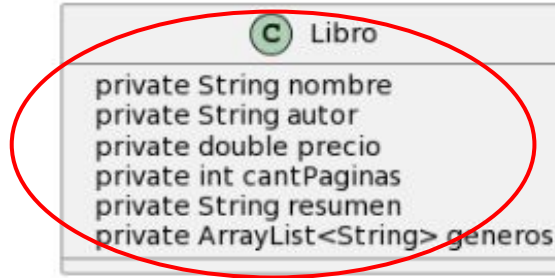
Libreria

```
private ArrayList<Libro> libros;  
private ArrayList<Revista> revista;  
private ArrayList<Cliente> clientes;
```

Agregar Atributos a las clases



Ambas clases
tienen los
mismos
atributos, hay
una relación
entre ellas

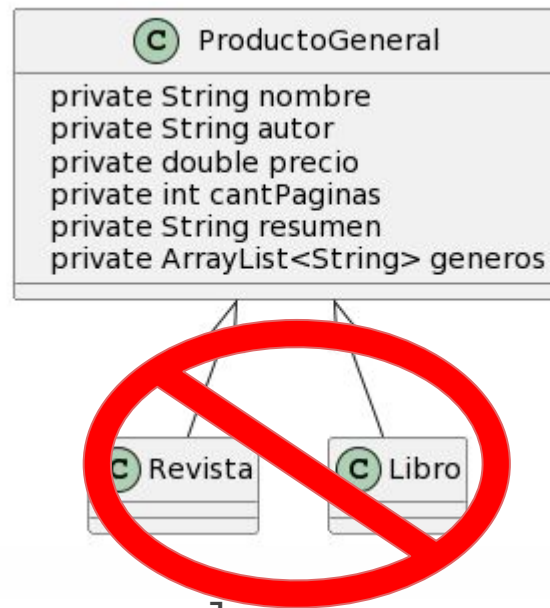


Primer aproximación

— — —
Revista y Libro quedan vacías
porque todos los atributos
comunes se subieron a una clase
“padre”



Revista y libro ¿Agregan algo o
cambian algún comportamiento?

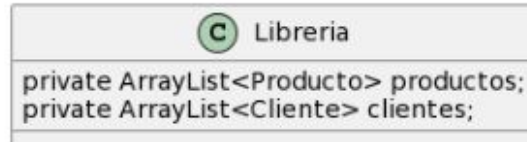
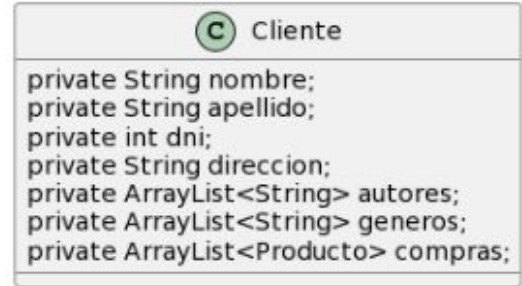
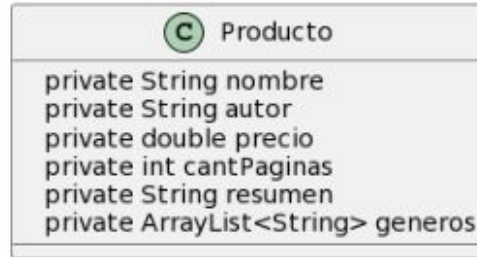


NO, entonces son sólo
instancias del producto y no
sub clases

Solución parcial

— — —


Agregar
comportamiento



Clase Producto

— — —

getters y setters
básicos

 Producto
<pre>private String nombre private String autor private double precio private int cantPaginas private String resumen private ArrayList<String> generos</pre>
<pre>public Producto(String nombre, String autor, double precio, int cantPaginas, String resumen) public String getNombre() public void setNombre(String nombre) public String getAutor() public void setAutor(String autor) public double getPrecio() public void setPrecio(double precio) public int getCantPaginas() public void setCantPaginas(int cantPaginas) public String getResumen() public void setResumen(String resumen)</pre>

Clase Producto - ArrayList géneros

Es necesario contemplar aparte los casos de manejo de ArrayList, no son get y set normales ya que peligra el encapsulamiento

Clase Producto - ArrayList generos

Asignar directamente:

```
public void setGeneros(ArrayList<String> generos){  
    this.generos = generos;  
}
```

1. No tenemos control de lo que viene. “hay repetidos?” “esta en minusculas?”
2. Se pierde lo que estaba. “se pisa el generos del producto”
3. Desde afuera se pueden guardar una referencia al arraylist de géneros y manipularlo sin pasar por el producto

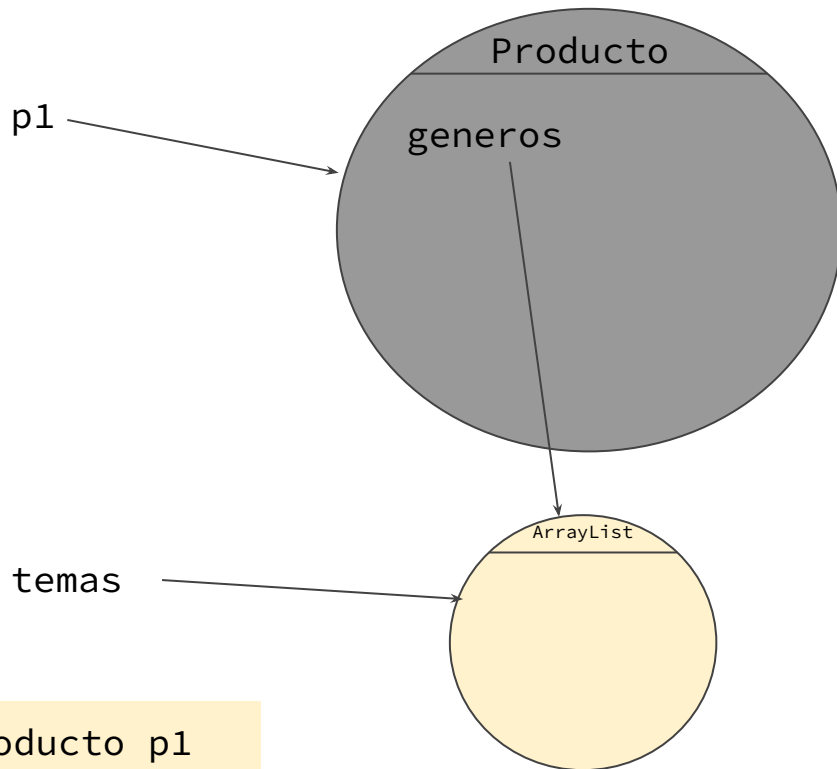
Referencia Exterior

Supongamos que tenemos un producto p1 y se le setea el géneros con una variable temas;

```
p1.setGeneros(temas);
```

temas y el atributo generos del producto p1 apuntan al mismo objeto

```
temas.clear(); // Borra para ambos  
temas.add("Hola"); //Agrega para ambos  
es el mismo objeto
```



Clase Producto - ArrayList generos

Mantener el control sobre el ArrayList

Se crea en el constructor, en lugar de que “venga” de afuera

```
generos = new ArrayList<String>();
```

de esta forma nadie de “Afuera” manipula el ArrayList

Clase Producto - ArrayList generos

Agregado individual

```
public void addGenero(String genero){  
    generos.add(genero);  
}
```

permite agregar controles, por ejemplo que no haya repetidos
o la forma que se agrega

Clase Producto - ArrayList generos

Agregado individual

```
public void addGenero(String genero){  
    if( ! generos.contains(genero) )  
        generos.add(genero);  
}
```

OJO

al usar el contains de ArrayList, se usa el equals del objeto parametro, en este caso es String por lo tanto no tengo que implementarlo, pero en otros casos sí puede ser necesario

Se puede incluso agregar el control eliminando el problema de mayusculas y minusculas

Clase Producto - ArrayList generos

Agregado individual

```
public void addGenero(String genero){  
    if( ! generos.contains(genero.toUpperCase()) )  
        generos.add(genero.toUpperCase());  
}
```

toUpperCase()

Es un método de String que retorna el String pero todo en Mayúsculas

Ya se que va a estar siempre en mayúsculas, por eso lo agrego en Mayuculas y pregunto si esta en Mayusculas

Clase Producto - ArrayList generos

Agregado Multiple

```
public void addGeneros(ArrayList<String> generos){  
    for(int i =0;i<generos.size();i++){  
        this.addGenero(generos.get(i));  
    }  
}
```

Se agregan los elementos, no se genera un acceso a mi estructura desde afuera

Reuso

se toma el arreglo que me pasan y se agrega uno a uno los nuevos géneros reusando el addGenero

Clase Producto - ArrayList generos

Devolver listado de generos

```
public ArrayList<String> getGeneros(){  
    return generos;  
}
```

1. Damos acceso a una variable que controlamos nosotros.
2. Desde afuera pueden agregar o sacar sin pasar por nosotros (perdemos el control)
3. Incluso se pueden agregar cosas que no sean sean String
4. Conceptualmente, puede ser que no se delegue apropiadamente la responsabilidad. En lugar de preguntarme si tengo un genero, me piden el listado y lo hacen desde afuera.

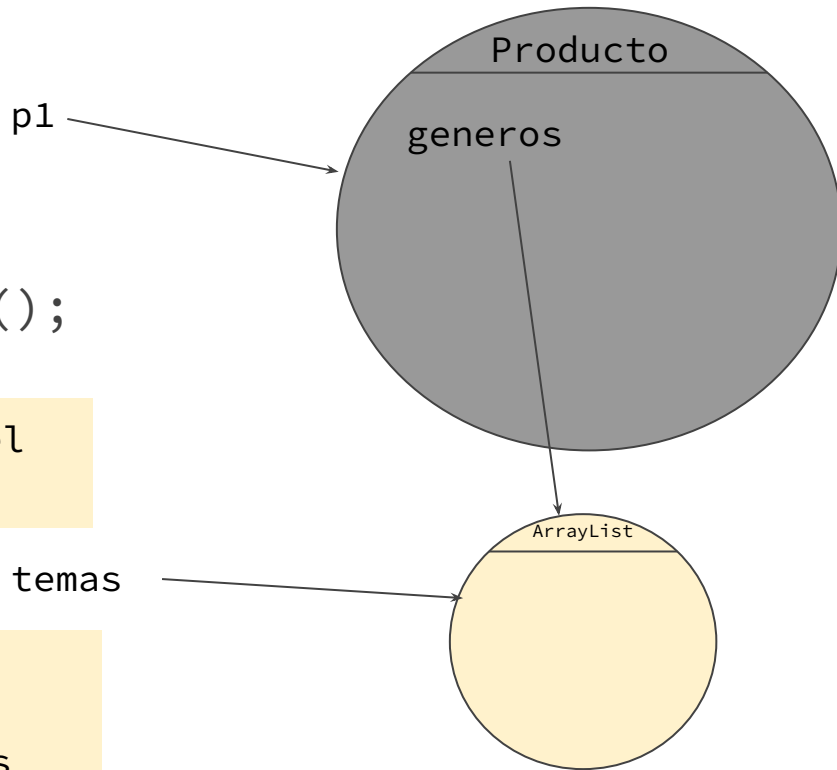
Referencia Exterior

```
ArrayList temas = p1.getGeneros();
```

Nuevamente, temas y el atributo generos del producto p1 apuntan al mismo objeto

```
temas.clear(); // Borra para ambos  
temas.add("Hola"); //Agrega para ambos no  
puedo garantizar más que no haya repetidos
```

```
temas.add(new Persona("juan")); // como temas  
no estaba tipado se puede agregar cualquier  
objeto
```



Clase Producto - ArrayList generos

Devolver listado de generos

```
public ArrayList<String> getGeneros(){  
    ArrayList aux = new ArrayList<String>();  
    aux.addAll(generos);  
    return aux;  
}
```

1. Se crea un arreglo auxiliar, luego se agregan todos los elementos de generos al auxiliar y se retorna el auxiliar
2. Desde afuera no apuntan a mi arrayList de generos, no pierdo el control
3. No soluciono el problema Conceptual, puede ser que no se delegue apropiadamente la responsabilidad. En lugar de preguntarme si tengo un genero, me piden el listado y lo hacen desde afuera.

Clase Producto - ArrayList generos

Consultar si tiene un genero

```
public boolean tieneGenero(String genero){  
    return generos.contains(genero.toUpperCase() );  
}
```

1. Mantiene el control, y además sabe que guardo todo en mayusculas!!
2. desde afueran preguntan y no saben como los guardamos a los generos


Manejo de ArrayList

El problema de los generos del producto se repiten cada vez que usamos ArrayList

Es necesario siempre plantearnos si es necesario retornar un ArrayList (siempre haciendo copia) ya que puede ser que desde afuera no deleguen bien la responsabilidad

Clase Producto

— — —

 Producto
<pre>private String nombre private String autor private double precio private int cantPaginas private String resumen private ArrayList<String> generos</pre>
<pre>public Producto(String nombre, String autor, double precio, int cantPaginas, String resumen) public String getNombre() public void setNombre(String nombre) public String getAutor() public void setAutor(String autor) public double getPrecio() public void setPrecio(double precio) public int getCantPaginas() public void setCantPaginas(int cantPaginas) public String getResumen() public void addGenero(String nGenero) public boolean tieneGenero(String unGenero) public void borrarGenero(String sacarGenero)</pre>

Clase Cliente

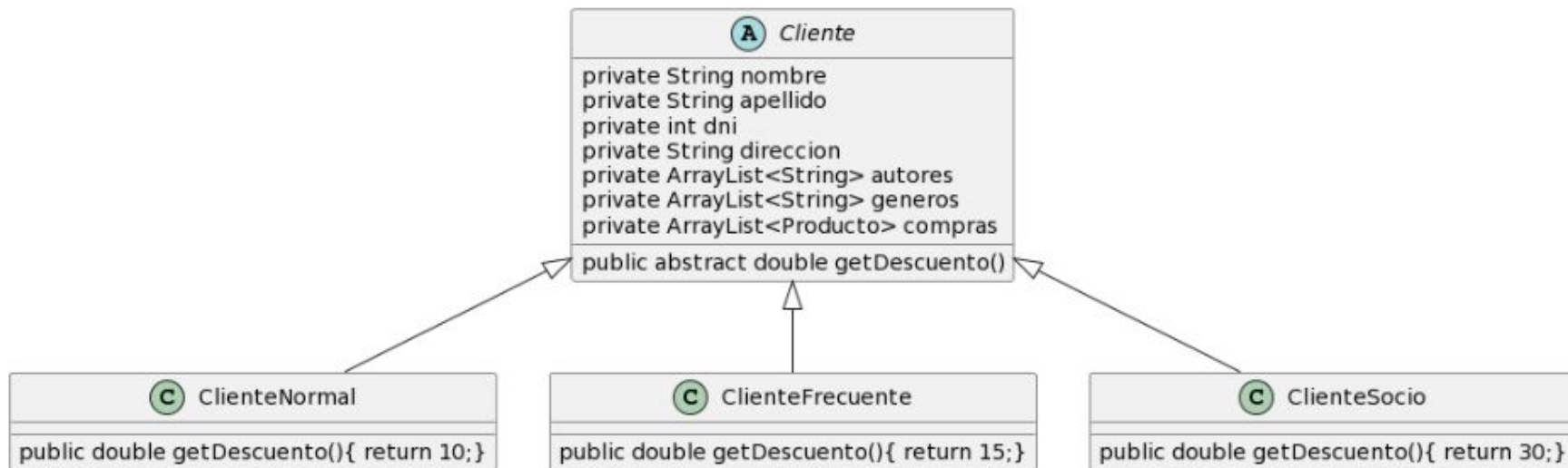
— — —

C Cliente	
<pre>private String nombre; private String apellido; private int dni; private String direccion; private ArrayList<String> autores; private ArrayList<String> generos; private ArrayList<Producto> compras;</pre>	
<pre>public Cliente(String nombre, String apellido, int dni, String direccion, double descuento) public String getNombre() public void setNombre(String nombre) public String getApellido() public void setApellido(String apellido) public int getDni() public void setDni(int dni) public String getDireccion() public void setDireccion(String direccion)</pre>	
<pre>public void addAutorFavorito(String autor) public boolean leGustaAutor(String autor) public void noMeGustaMasAutor(String autor) public void addGeneroFavorito(String gene) public boolean leGustaGenero(String gene) public void noMeGustaMasGenero(String gene) public void addCompra(Producto p) public boolean yaCompro(Producto p)</pre>	

Cientes

— — —

La librería les da descuentos a sus clientes: hay clientes **normales** que reciben un 10% de descuento, clientes **frecuentes** que reciben un 15% de descuento y clientes **socios** que reciben un 30% de descuento.

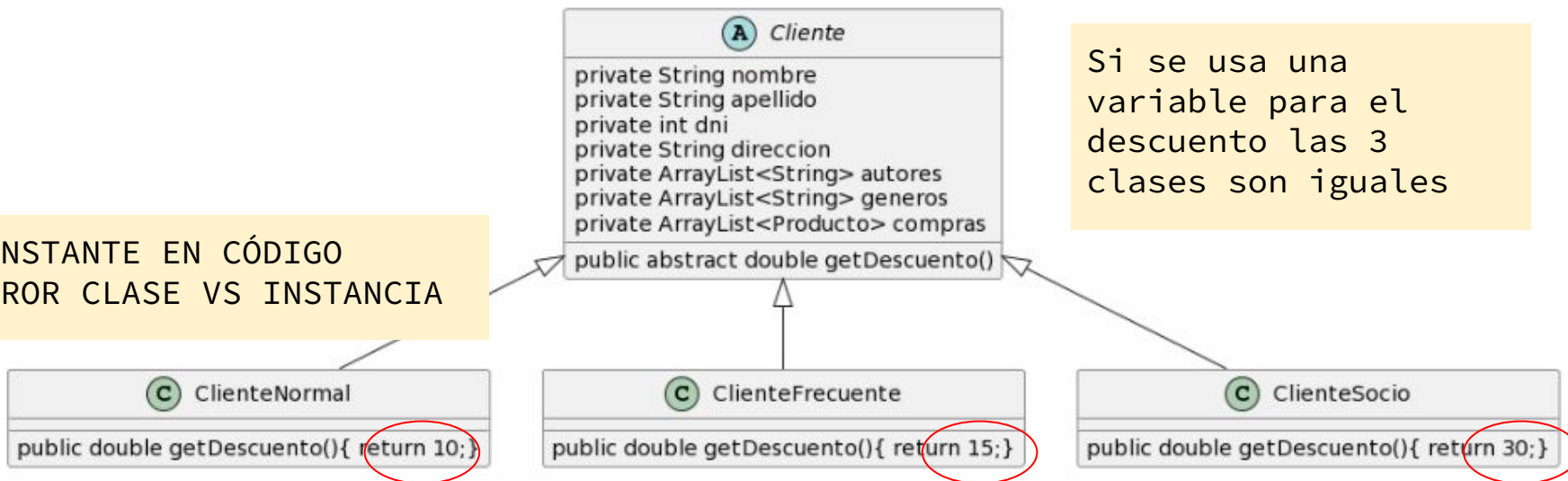


Clientes

— — —

La librería les da descuentos a sus clientes: hay clientes **normales** que reciben un 10% de descuento, clientes **frecuentes** que reciben un 15% de descuento y clientes **socios** que reciben un 30% de descuento.

CONSTANTE EN CÓDIGO
ERROR CLASE VS INSTANCIA



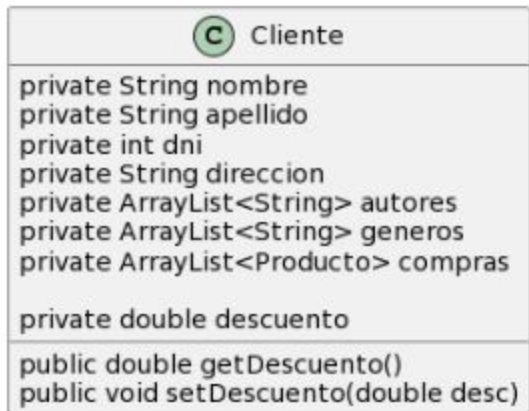
Si se usa una variable para el descuento las 3 clases son iguales

Clase Cliente

— — —

Se agrega un

atributo **descuento**




Servicios

Implementar los siguientes servicios:

1. Conocer el precio de un libro o revista para un cliente determinado (considerando el descuento)
2. Conocer si un cliente ya compró un libro o revista determinada.
3. Conocer si a un cliente le gusta un libro o revista determinada. A algunos clientes les gusta el libro solo si el autor del mismo está en su lista de autores favoritos. Pero hay clientes más exigentes que piden que el autor esté en su lista de favoritos y además que contenga al menos un género de los que le gusta.
4. Devolver un listado de los clientes que les gusta un libro o revista.

Servicio 1

— — —

 Libreria
private ArrayList<Producto> productos; private ArrayList<Cliente> clientes;
public double getPrecio(Producto p, Cliente c)

```
public double getPrecio(Producto pp, Cliente cc) {  
    return pp.getPrecio() - pp.getPrecio()*cc.getDescuento()/100;  
}
```

Servicio 2

— — —

```
public boolean yaCompro(Cliente cc, Producto pp) {  
    return cc.yaCompro(pp);  
} //Se delega al cliente la pregunta
```

C Libreria
private ArrayList<Producto> productos; private ArrayList<Cliente> clientes;
public double getPrecio(Producto p, Cliente c) public boolean yaCompro(Cliente cc, Producto pp)

Servicio 3

— — —

3. Conocer si a un cliente le gusta un libro o revista determinada. A algunos clientes les gusta el libro solo si el autor del mismo está en su lista de autores favoritos. Pero hay clientes más exigentes que piden que el autor esté en su lista de favoritos y además que contenga al menos un género de los que le gusta.

Acorde al tipo de cliente es lo que hay que implementar

Solución sin polimorfismo, hacer textual lo que pide

Servicio 3

Acorde al tipo

Solución sin p

if por tipo

3. Conocer si a un cliente le gusta un libro o revista determinada. A algunos clientes les gusta el libro solo si el autor del mismo está en su lista de autores favoritos. Pero hay clientes más exigentes que piden que el autor esté en su lista de favoritos y además que contenga al menos un género de los que le gusta.

```
public boolean leGusta(Cliente cc, Producto pp) {  
    boolean leGusta = cc.leGustaAutor(pp.getAutor());  
    if (!cc.esExigente() ) {  
        return leGusta;  
    } else {  
        if (leGusta) {  
            ArrayList<String> generos = cc.getGeneros();  
            for(int i =0; i<generos.size();i++) {  
                if (pp.tieneGenero(generos.get(i))) {  
                    return true;  
                }  
            }  
        }  
        return false;  
    }  
}
```

Mala Delegación de responsabilidades

Queda limitada la solución

Servicio 3

— — —

3. Conocer si a un cliente le gusta un libro o revista determinada. A algunos clientes les gusta el libro solo si el autor del mismo está en su lista de autores favoritos. Pero hay clientes más exigentes que piden que el autor esté en su lista de favoritos y además que contenga al menos un género de los que le gusta.

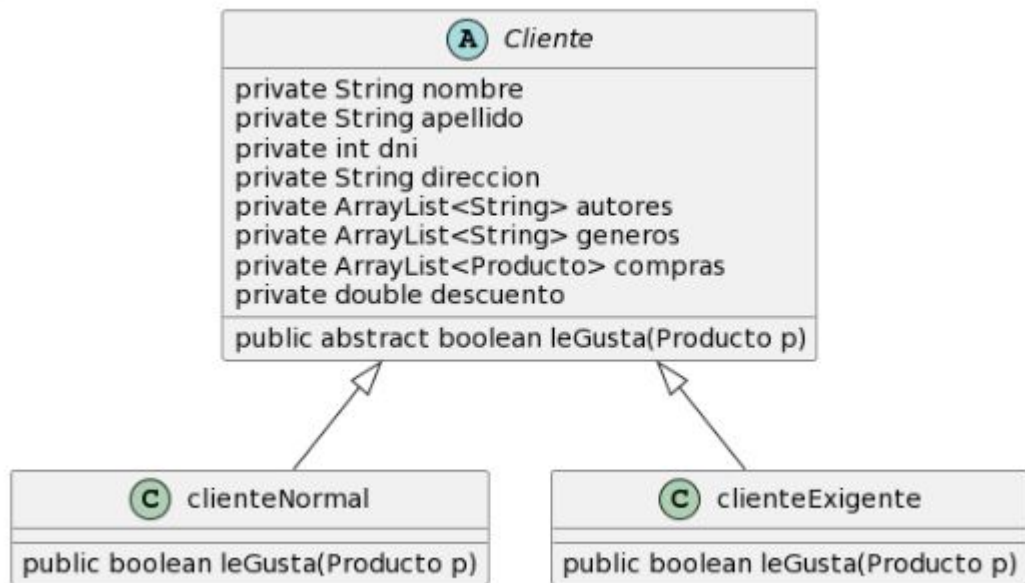
Acorde al tipo de cliente es lo que hay que implementar

Solución Orientada a Objetos, delegar la responsabilidad al Cliente, “Es él el que decide si le gusta un producto o no”

```
public boolean leGusta(Cliente cc, Producto pp) {  
    return cc.leGusta(pp);  
}
```


El cliente decide como le gusta un libro

— — —



Cliente Normal

Consulta en su lista de autores si esta el autor del producto

```
public boolean leGusta(Producto pp) {  
    return autores.contains(pp.getAutor());  
}
```

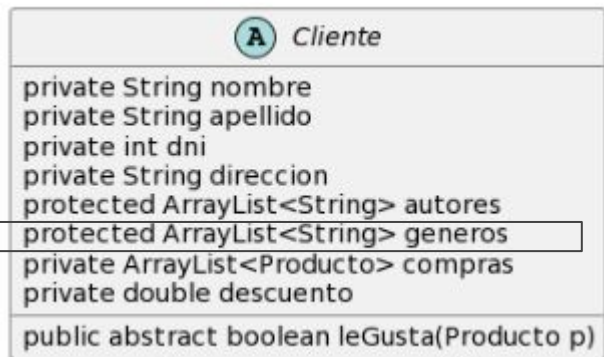
C clienteNormal
public boolean leGusta(Producto p)

A Cliente
private String nombre private String apellido private int dni private String direccion protected ArrayList<String> autores private ArrayList<String> generos private ArrayList<Producto> compras private double descuento
public abstract boolean leGusta(Producto p)

Cliente Exigente

Además de consultar si le gusta el autor, recorre su lista de géneros y se fija si el producto tiene alguno

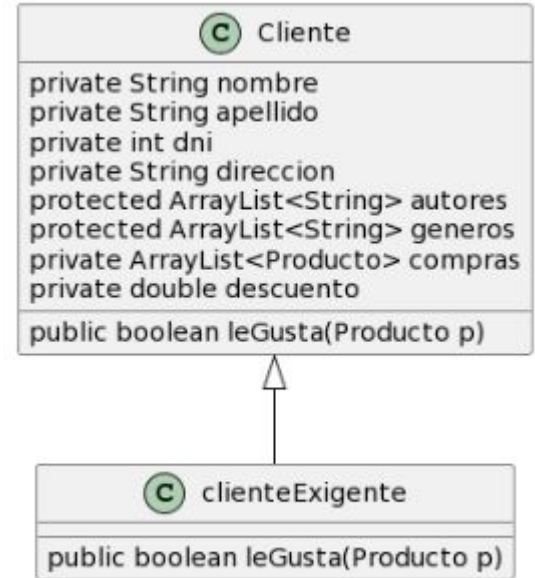
```
public boolean leGusta(Producto pp) {
    boolean seguir = autores.contains(pp.getAutor());
    if(seguir) {
        for(int i =0;i<generos.size();i++) {
            if (pp.tieneGenero(generos.get(i))) {
                return true;
            }
        }
    }
    return false;
}
```



Clientes

Ambos comparten la búsqueda del autor.

Se sube ese comportamiento a la clase Cliente, con lo cual la clase ClienteNormal queda sin funcionalidad (desaparece) y la clase Cliente ahora es concreta, tiene una conducta por defecto. ClienteExigente agrega comportamiento a la conducta del padre



Cliente

```
public boolean leGusta(Producto pp) {  
    return autores.contains(pp.getAutor());  
}
```

ClienteExigente

```
public boolean leGusta(Producto pp) {  
    boolean seguir = super.leGusta(pp);  
    if(seguir) {  
        for(int i =0;i<generos.size();i++) {  
            if (pp.tieneGenero(generos.get(i))) {  
                return true;  
            }  
        }  
    }  
    return false;  
}
```

NuevosClientes

Si aparecen nuevas conductas de cliente que definen como les gusta un libro, no es necesario tocar la clase librería, sino crear la nueva conducta como hijo de cliente y definir el leGusta de nuevo

Servicio 4

Devolver un listado de clientes que le gusta un libro o revista


```
public ArrayList<Cliente> getClientes(Producto pp){  
    ArrayList<Cliente> aux = new ArrayList<Cliente>();  
    for(int i =0; i< clientes.size();i++) {  
        Cliente cc = clientes.get(i);  
        if(cc.leGusta(pp)) {  
            aux.add(cc);  
        }  
    }  
    return aux;  
}
```

Se delega la consulta
al cliente

Librería

— — —

Además de los métodos básicos

 Libreria
<pre>private ArrayList<Producto> productos; private ArrayList<Cliente> clientes;</pre>
<pre>public double getPrecio(Producto p, Cliente c) public boolean yaCompro(Cliente cc, Producto pp) public boolean leGusta(Cliente cc, Producto pp) public ArrayList<Cliente> getCientes(Producto pp)</pre>

Solución

— — —

