## POO

## Herencia en java

La herencia en java la expresamos con un extends, sabemos que hay una super clase y que tiene una subclase ( hija).

Entonces en el código de abajo lo que miramos es que en la subclase EMPLEADO aplicamos EXTENDS PERSONA (**Persona es la clase mayor o super clase**), le creamos un constructos a la subclase que seria empleado PERO LE AGREGAMOS SUPER CON LOS DOS ATRIBUTOS DEL CONSTRUCTOR DE LA SUPER CLASE.

Text

Description automatically generated

Para mostrar mi objeto en el ejercicio de persona, tengo que llamar al objeto y aplicarle el metodo que muestra mis datos, en este caso persona1.mostrarDatos();

# Sobre carga de métodos

Creamos dos constructores en nuestra clase, seleccionando diferentes atributos. Se pueden sobrecargar, pero los parámetros tienen que ser diferentes o una cantidad diferentes de parámetros o que sean de otro tipo. Se pueden llamar igual

Ah eso le llamamos LA FIRMA, que tiene que ser diferente en cada método.

Tenemos dos constructores

# Modificadores de Acceso

Antes de, en los proyectos grandes de empresas ya laborando, están desarrollados con muchas clases y muchos paquetes.

* Public: para poner atributos públicos y utilizarlos en otras clases que están en otros paquetes.
* Private: hace una restricción de ese atributo, solo lo podemos usar en la misma clase.

# Encapsulamiento y Métodos Accesorios

El encapsulamiento vamos a coger los atributos de mis clases, encapsulamiento ocultar los datos para que solo puedan ser modificados por métodos de la misma clase.

Tambien es para proteger los datos y nadie los pueda editar.

Private: lo puedo usar con métodos.

## Los métodos accesores o accesorios

Son los getters y setters

* Set: Establecer
* Get : Mostrar en java

# Constantes

Son cosas que no podemos cambiar que serán iguales y al ponerle final en el atributo la vamos a volver constante.



# Miembros de clase

Miembros estáticos de una clase, hay varios objetos y puedo personalizar los atributos que este mismo tiene. Cuando este static, cuando ya está en un atributo estático le pertenece ya a la clase, o sea que el ultimo cambio que haga en ese objeto, va a cambiar para todos los objetos.

# Ejercicio Explicación 5

## Diseñar un programa para trabajar con triángulos isósceles, para ello defina los atributos necesarios que requieren , proporcione métodos de consulta, un método constructor e implemente métodos para calcular el perímetro y el área de un triángulo, además implementar un método que a partir de un arreglo de triángulos devuelva el área más grande de los triangulo.

1. Identificamos nuestras variables como privadas.
   1. Double base
   2. Double lado
2. Hacemos el método constructor.
3. Metodo para encontrar el perímetro del triangulo
4. Me voy para la clase Main y tengo que llamar el método escaner, ya que tengo que pedir los datos de cada triangulo.

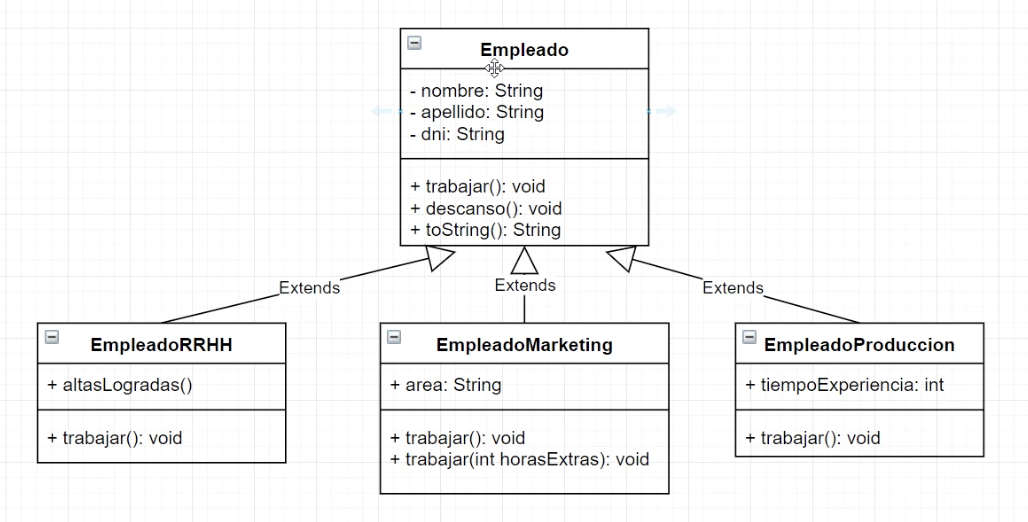
**Entendiendo el ejercicio, tenemos que ingresar una cantidad de triángulos en el programa, con las características y el programa nos debe de mostrar el triángulo con el área más grande entre la cantidad de triángulos.**

**identifico**

En POO



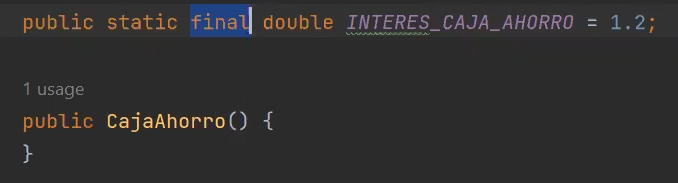
public boolean ejercerTitulo() {  
 boolean ejercerTituloValor = true;  
 if (isSelladoPorInstituto() != true && isSelladoPorMinistro() != true) {  
 ejercerTituloValor = false;  
 }  
 return ejercerTituloValor;



# Clase de 10 de noviembre

Explicación del código de que fue generado. Recordando que los primitivos son los que están en minúscula por así decirlo.

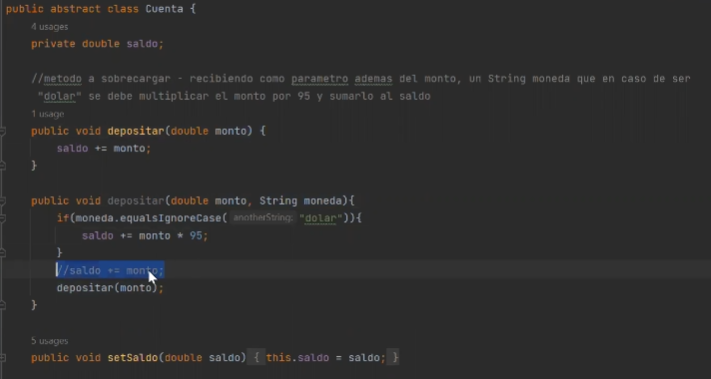
La idea de tener un método abstracto, es que mis clases hijas, hagan ese método diferente cada una de las clases hijas.



Final es para constantes, por eso mi constructor queda vacío.

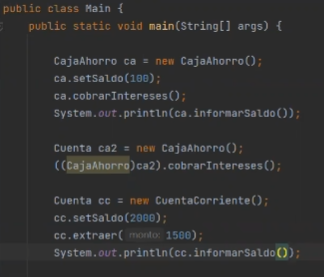
Static solo lo podemos poner en la clase, todo lo que no es static lo referencio cmo un objeto.

* Equals()🡪 es para comparar objetos. También podemos poner el equalsIgnoreCase, que es para poner Mayúsculas o minúsculas.



La idea es siempre usar lo que ya tenemos o reutilizar el código que ya tenemos, métodos y entre otras cosas.

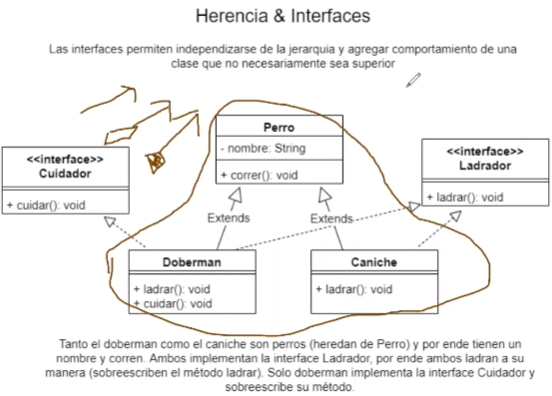
En el código de abajo vimos que podemos crear objetos de esa forma y mirar ambos casos cuando tenemos que instanciar para invocar el método necesario.



# INTERFACE

permiten declarar constantes que van a estar disponibles para todas las clases que queramos (implementando esa interfaz) Obligar a que ciertas clases utilicen los mismos métodos (nombres y parámetros).

* No tiene una jeraquia de clases. Pero si se puede realizar la relación.
* Si podemos implementar varis interfaces, ayudan a subsanar la herencia multiple.
* No tienen constructor directamente.
* Es un tipo de clase de java.
* Los atributos que va a tener la interface son tipo  **public, static y final.**
* **<<interface>>** seria la nomenclatura en el UML
* **Toda clase que implemente una INTERFACE esta obligada a implementar todos los métodos de esa interface.**
* Todos sus métodos son abstractos, pero no hace falta de poner que sean abstractos.
* No se tiene que tener varios métodos.
* Recordar siempre, que la firma de los métodos se TIENE QUE RESPETAR.
* Mediante una interface indicamos que vamos hacer pero no EL FAMOSO COMO, o sea, el procedimiento.



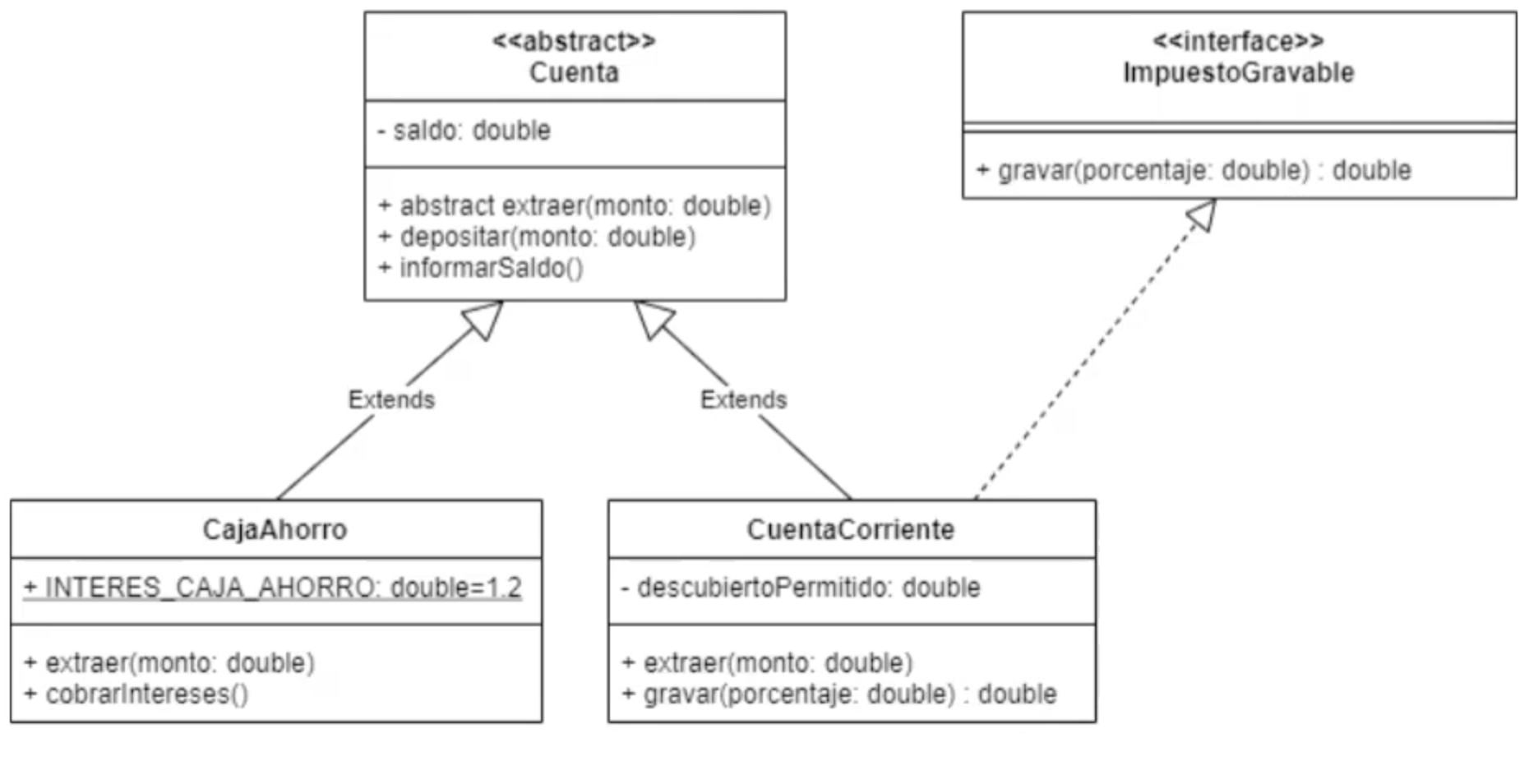
# Casos en que elegimos una clase abstracta

* Hay atributos no constantes, es decir, que todas las clases tienen atributos que no sean final o constantes, que en cada objeto que instanciamos sean **Distintos**.
* Pueden tener métodos que las hijas no hagan pero si otras.
* Si necesitamos que todas las clases hijas hagan algo, pero cada una a su manera, ese mtodo es abstract y se deberá sobrescribir en las clases hijas.
* Solo podemos hacer extend en una clase.

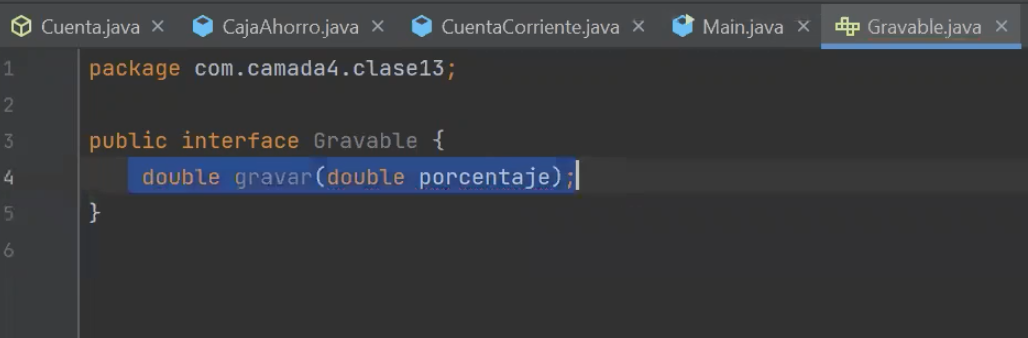
# Casos en que elegimos Interface

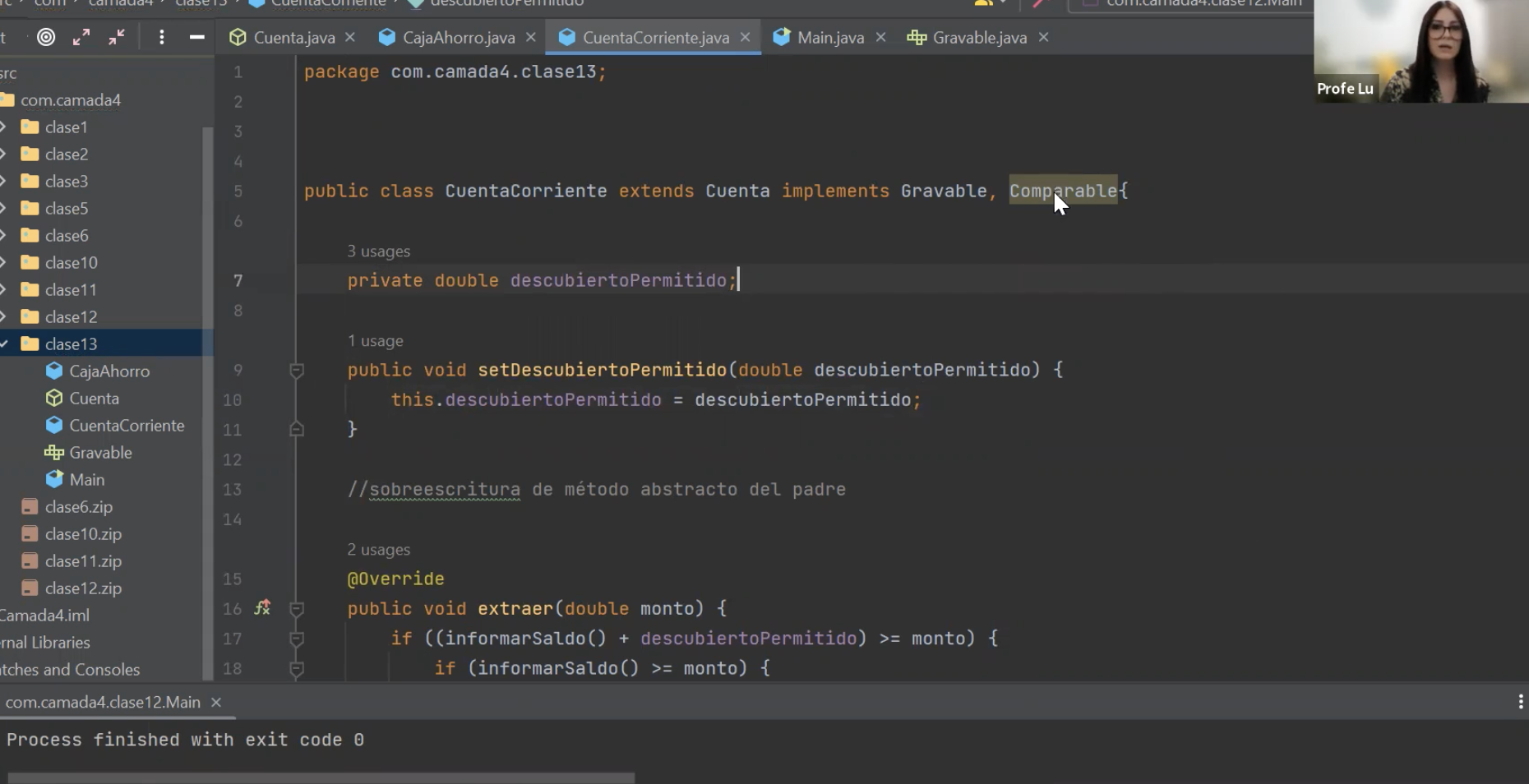
* Si nuestros atributos pueden ser constantes, static o final, podemos definir nuestra clase como INTERFACE.

Ejercicio con UML



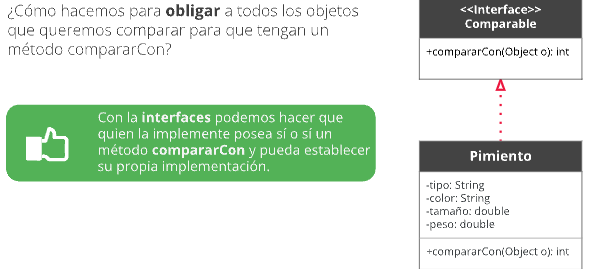
En este caso asi creamos los métodos de nuestra interface.



ACA VEMOS, como en la clase hija agregamos las interfaces necesarias, con el implements.

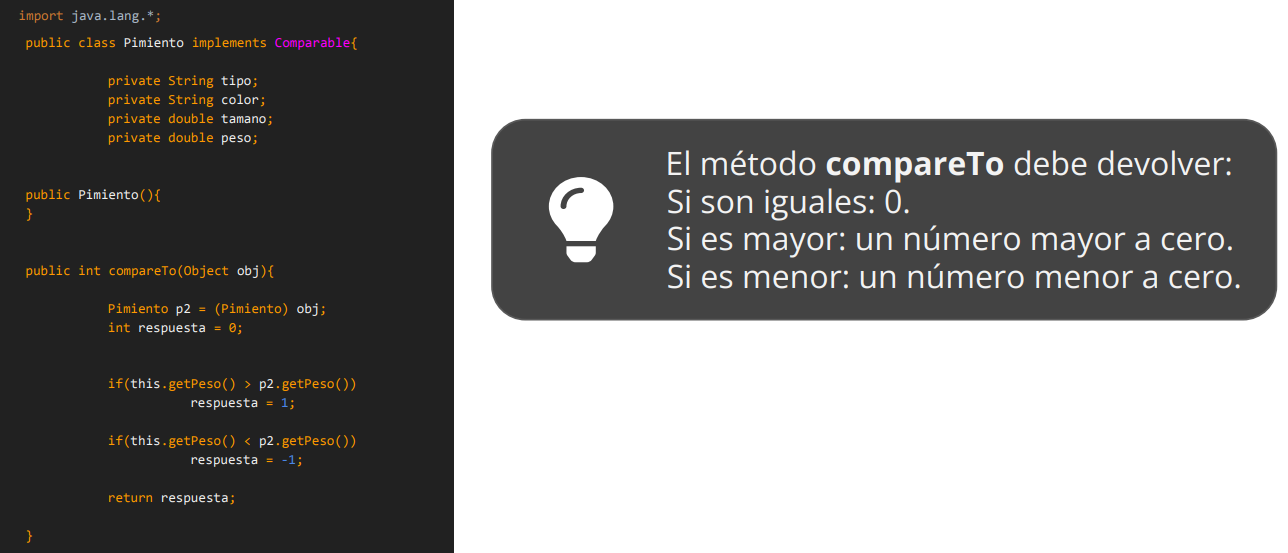
Mirar la actividad de **15 de noviembre para hacerla.**

# Interface Comparable

* Poder comparar dos objetos, recibiendo como parámetro al otro objeto con el que se desea hacer la comparación, teniendo los siguientes resultados.
  + Cero si son iguales
  + 1 si el objeto que invoca el método es mayor al recibido.
  + -1 si el objeto que invoca el método es menor al recibido.
* 
* Graphical user interface, text, application

  Description automatically generated

Acá casteamos el objeto.

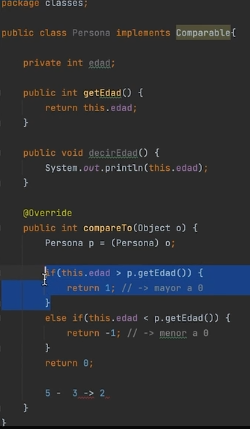


Text

Description automatically generated

Siendo finalmente el pimiento amarillo mayor que al rojo.

El siguiente ejemplo en código, lo que esta haciendo es hacer el comparable, donde se comparara la edad de ambas personas ( objetos) y mostrara el resultado. Pero hace el casteo del objeto.



Segunda forma

Text

Description automatically generated

# Ejercicio 16 de noviembre

El ejercicio trata de una universidad, pero lo que importa es que vemos el ejemplo de que hay una clase abstracta, pero que no tiene necesidad de que tiene métodos abstractos.

* Una clase abstract no tiene como obligación tenes métodos abstractos.

# Colecciones

Grupo de objetos, coger toda esa cantidad de objetos, por ejemplo, varias ruedas de un carro. Usamos la famosa COLLECTION, es una interface.

* Collection tiene métodos comunes que nos ayuda a añadir, eliminar y etc.
* También de la interface collection, salen otras interfaces como las set
* Set, no permite elementos duplicados (equals y hashcode).
* List, si permite duplicados
* Map no es hered