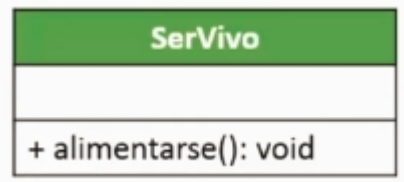
Clases y métodos abstractos

Clase abstracta:

- Solo se usan como Superclases, es decir clases padres.
- Como mínimo debe tener una subclase
- No se pueden instanciar (crear objetos) de una clase abstracta.
- Es un bosquejo, una plantilla de lo que tendrán sus clases hijas.

Veamos un Ejemplo!

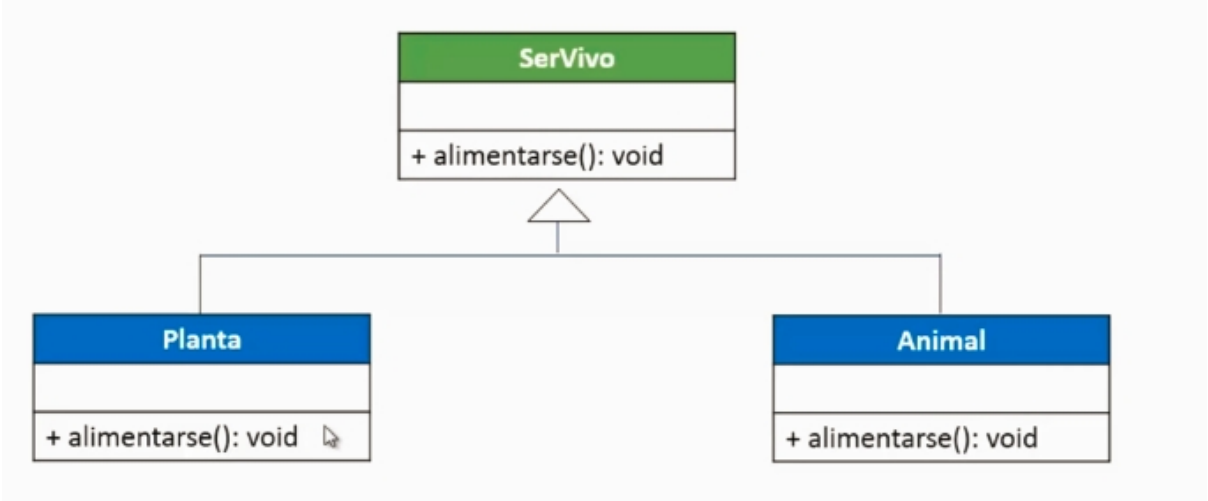
Supongamos que tenemos que implementar la siguiente Clase:



Esta clase tiene un método que es alimentarse().

Si tenemos que diseñar la clase **SerVivo** vemos que es una clase general, muy amplia, “Un ser vivo” puede ser una planta, un animal, un insecto, etc. Existen diferentes tipo de seres vivos, entonces cómo implementariamos el método **alimentarse()**, ya que este método depende de cada ser vivo, no es lo mismo cómo se alimenta una planta o cómo se alimentan las personas.

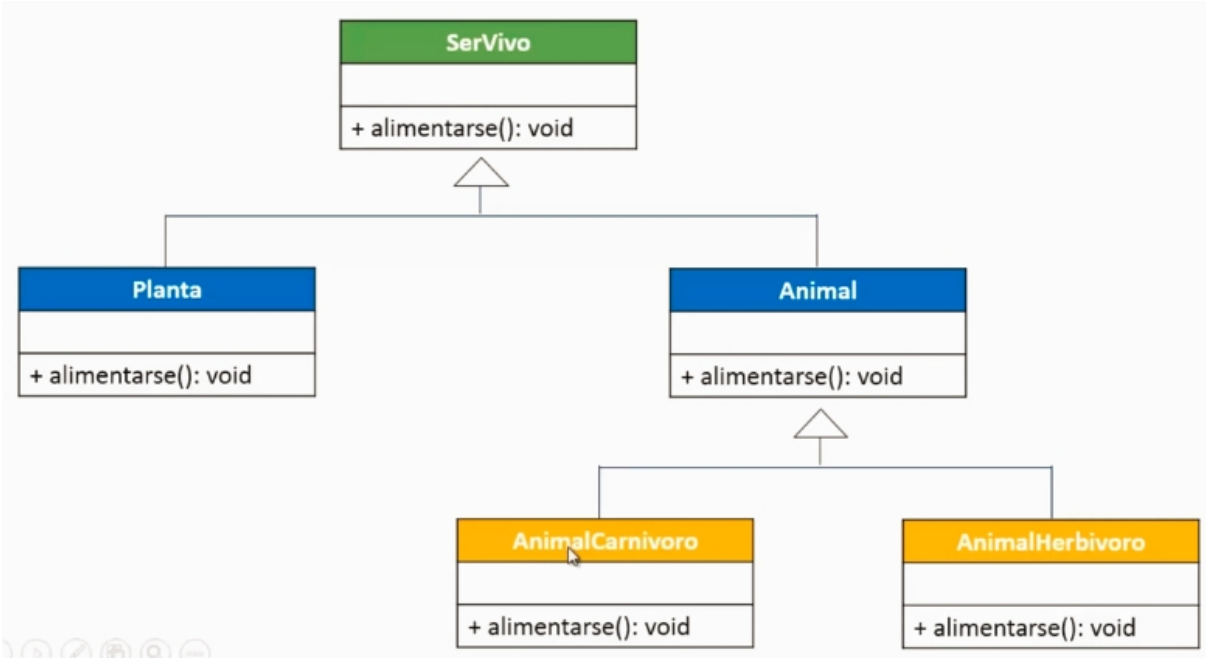
Supongamos que tenemos dos tipos de seres vivos: una planta y un Animal



Las clases **Planta** y **Animal** serían subclases de la clase **SerVivo**, ahora bien, si miramos la clase **Planta** vemos que como hereda de la clase **SerVivo**, está también tiene un método **alimentarse()**, el cual para el caso de una planta si sabemos cómo se debe implementar, ya que todos conocemos cómo se alimentan las plantas, por medio de la fotosíntesis, a diferencia de la clase padre que no sabíamos, con exactitud cómo cada ser vivo se podía alimentar.

En cambio en la subclase **Animal** tenemos otra vez el mismo problema, no sabemos con exactitud o es muy amplio como se alimenta cada animal, por lo cual no podemos implementar el método **alimentarse()**.

Entonces podemos diseñar dos clases más: **AnimalCarnivoro**, y **AnimalHerbivoro** que sean subclases de la clase **Animal**, por lo que nuestro diagrama de clases quedaría:



En donde ahora sí podríamos implementar cada uno de los métodos **alimentarse()** ya que contamos con la información correspondiente para hacerlo.

- ★ Hasta el momento sabemos que todo ser vivo debe alimentarse pero no sabemos con exactitud cómo debe hacerlo. Entonces, como no sabemos cómo lo debemos implementar lo definimos como abstracto.

- ★ Por definición cuando una clase tiene al menos un método abstracto dicha clase debe denominarse como abstracta también.

- ★ El método abstracto debe implementarse si o si en las subclases.

Método abstracto

- Son métodos declarados pero no implementados.
- Se declaran con la palabra reservada abstract
- Las clases subclases deberán implementar los métodos abstractos – O serán abstractas ellas también.

```
abstract void alimentarse();
```

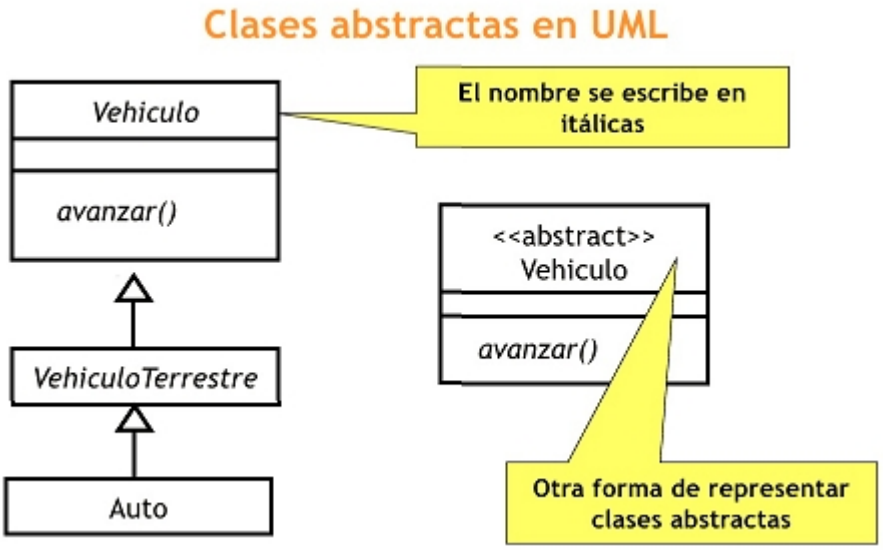
NOTA: **No hay llaves**!! No están implementados después de la declaración se pone solo un ;

Clase abstracta

- Las clases abstractas suelen usarse para representar clases con implementaciones parciales.
- Algunos métodos no están implementados pero sí declarados. (Como vimos en el ejemplo)
- No podemos crear objetos, es decir instanciar objetos de una clase abstracta.

En notación UML:

- Una clase abstracta se denota con el nombre de la clase y de los métodos con letra "itálica".



Importante:
Las clases y métodos no abstractos se denominan **concretos**.

Vamos a Codificar!:

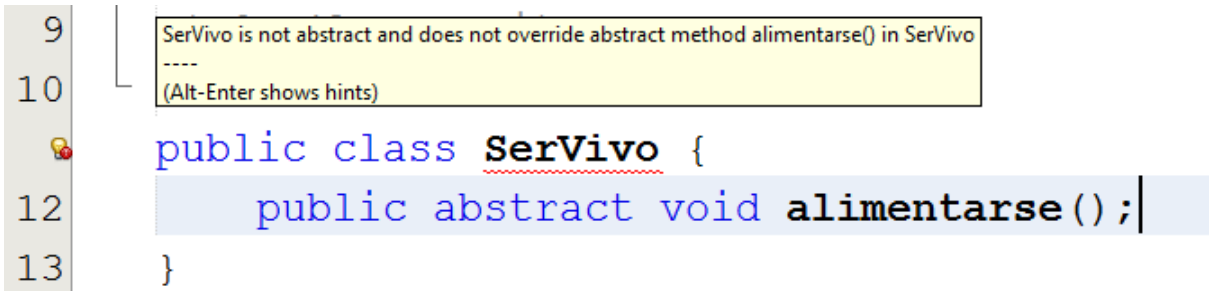
1- Creamos un nuevo proyecto: MiPrimerClaseAbstracta

2- Creamos una clase SerVivo y en ella colocamos el método:

```
public abstract void alimentarse();
```

(Estemos atentos a los errores que nos marca netbeans)

Vemos que Java nos detecta que hay un método abstract  entonces nos pide que declaremos a la clase SerVivo como abstract tambien.

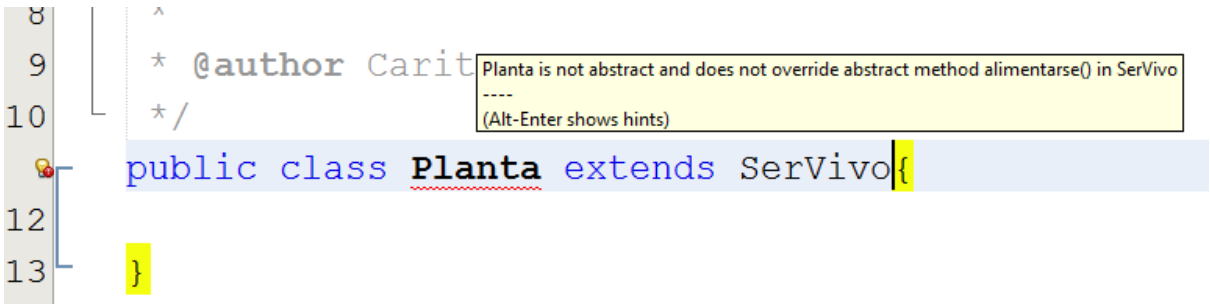


3- Definimos a la clase SerVivo como abstract.

```
public abstract class SerVivo {
    public abstract void alimentarse();
}
```

4- Creamos la subclase Planta la hacemos heredar, extender de la clase SerVivo.

(Estemos atentos a los errores que nos marca netbeans)



Java detecta que necesitamos implementar el método abstract heredado de SerVivo.

5- Entonces cómo lo solucionamos al error??

6- Te animas a seguir implementando el diagrama de clases del ejemplo?

7- Cuando termines de crear el resto de las clases, creamos una clase de nombre Principal en ella vamos a implementar nuestro método main.

8- Qué pasa si intentas crear un objeto de la clase SerVivo??

9- Creamos, instanciamos un objeto de la clase Planta y llamamos al método alimentarse()

10- Instanciamos objetos del resto de las clases e invocamos a sus métodos alimentarse()

Preguntas??????????

Interfaces

Una interfaz es una clase completamente abstracta (una clase sin implementación)

- En Java, las interfaces se declaran con la palabra reservada **`interface`** de manera similar a como se declaran las clases abstractas.
- En la declaración de una interfaz, lo único que puede aparecer son declaraciones de métodos (su nombre y signatura, sin su implementación) y definiciones de constantes simbólicas.
- Una interfaz no encapsula datos, sólo define cuáles son los métodos que han de implementar los objetos de aquellas clases que implementen la interfaz.
- En Java, para indicar que una clase implementa una interfaz se utiliza la palabra reservada **`implements`**.
- La clase debe entonces implementar **todos** los métodos definidos por la interfaz o declararse, a su vez, como una clase abstracta (lo que no suele ser especialmente útil):

```
abstract class SinArea implements Figura
{
}
```

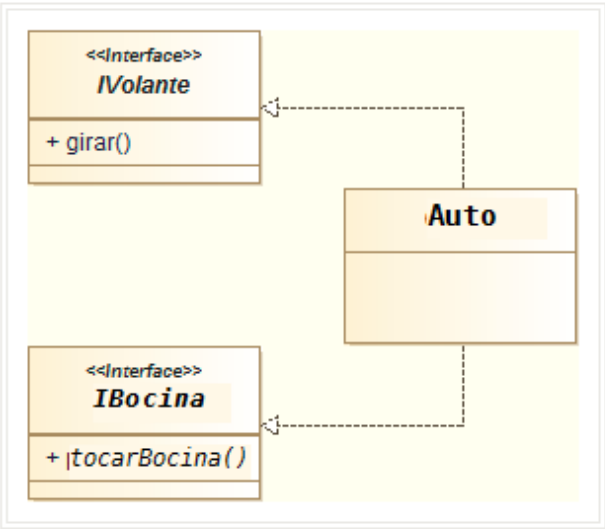
Interfaz – Representación

En los Diagramas de Clases UML los interfaces se pueden representar de varias formas.

Una clase puede implementar a la vez tantos interfaces como se desee, pero con un solo método de un solo interfaz que falte por implementar la clase será abstracta.

Interfaz - Utilización

Un Auto puede girar y puede tocar la bocina. Así, una clase Auto puede implementar simultáneamente la interfaz IVolante y la interfaz IBocina, cuya representación podría quedar como sigue:



Obsérvese que la linea que vincula una clase y sus interfaces es dirigida, **parte de la clase y acaba en el interfaz**. Además se representa con **línea discontinua** y con **punta de flecha cerrada**.

En resumen:

Se me ocurre el caso de un amigo que va a enseñarle a su vecino el coche ultimo modelo que se acaba de comprar.

> *— Entra, Entra. ¿Qué te parece?*

Lo primero con lo que se encuentra el vecino al entrar en la **plaza del conductor** es esa pieza grande y redonda al frente, que parece que esté hecha para cogerse a ella, que se llama **volante**. A pesar de que **es la primera vez** que el vecino entra en ese coche en concreto, da por sentado que si gira el volante el coche girará también. El vecino, probablemente, no sabe cómo se las ingenia el coche para girar con el volante, y ni falta que le hace para poder conducir.

Eso es un interfaz ...

El interfaz es la razón por la que cuando se gira el volante, el vehículo también gira. Así, una vez se sabe girar un vehículo se sabe girar cualquier otro, porque todos funcionan igual, porque todos realizan la misma funcionalidad impuesta por el hecho de que haya un volante, todos implementan el mismo interfaz.

> *— Bueno, y ... ¿Eso para que sirve en el diseño de sistemas?*

Pues la respuesta es:

> *"Si una clase implementa un interfaz, significa que los objetos instanciados de ella disponen de una determinada funcionalidad adicional en forma de métodos que podrán ser utilizados a conveniencia. Por tanto esa clase podrá ser utilizada para un determinado fin. En caso contrario no. Mala suerte".*

Interfaz – Codificación

El ejemplo anterior podría **codificarse** en **Java** de la siguiente **manera**:

```
1     public interface IVolante {
2        public void girar();
3     }
4     public interface IBocina {
5        public void tocarBocina();
6     }
7     class Coche implements IVolante, IBocina {
8        public void girar() {
9           System.out.println("¡Girando, girando!");
10       }
11       public void tocarBocina() {
12          System.out.println("¡Soy una bocina de precaución!");
13       }
14    }
```

Ejercicio:

La empresa Zapatos Fashion SRL necesita una aplicación informática para administrar los datos de su personal.

Del personal se conoce: número de DNI, nombre, apellidos y año de ingreso.

La empresa maneja dos tipos de personal: el personal con salario fijo y el personal a comisión. Los empleados con salario fijo tienen un sueldo básico y un porcentaje adicional en función del número de años que llevan: menos de dos años salario base, de 2 a 3 años: 5% más; de 4 a 7 años: 10% más; de 8 a 15 años: 15% más y más de 15 años: 20% más.

Los empleados a comisión tienen un salario mínimo que será constante para todos los empleados de este tipo e igual a $15000, y varía según el número de clientes atendidos y el monto de la venta por cada cliente captado. El salario se obtiene multiplicando los clientes captados por el monto por cliente, si el salario por los clientes captados no llega al salario mínimo, cobrará esta cantidad.Si, lo supera cobrará según lo siguiente: si es mayor a 20000 su sueldo es 20% más. Si está entre 20000 y 30000 su sueldo se aumentará un 30%. y si supera los 30000 cobrará un 40% más.

Se contará con una clase padre *Empleado* de la cual no se podrán crear objetos y

de la que heredan las clases *EAsalariado* y *EComision*. En todas las clases debe haber un constructor con parámetros para todos los atributos y otro vacío. En todos deben crearse los getters and setters correspondientes. *Empleado* contará con un método *imprimir()* y un método *obtenerSalario()*.

Se creará una clase gestora y en el método *main* se creará un arrayList con los siguientes objetos:

- Javier Gómez, DNI: 569587A, desde 2010, salario fijo base.
- Eva Nieto, DNI: 695235B, desde 2010, 10 clientes captados a $250 cada uno.
- José Ruiz, DNI: 741258C, desde 2012, 20 clientes captados a $250 cada uno.
- María Núñez, DNI: 896325D, desde 2013, salario fijo base.
- Ramon Romero, DNI: 341258C, desde 2016, 15 clientes captados a $350 cada uno.

Desde el método *main* se llamará a estos otros dos métodos:

- sueldoMayor(): Dado un arrayList de Empleado muestra el nombre, apellido y salario del que más cobra.

- mostrarTodos(): Dado un array de objetos Empleado lo recorre imprimiendo los datos de todos los empleados.