****

**Diagrama estático**

**Manual de uso**

**por Fernando Dodino**

**Versión 1.0**

**Septiembre 2017**

Distribuido bajo licencia [Creative Commons Share-a-like](https://creativecommons.org/licenses/by-sa/4.0/legalcode)

**Índice**

[1 Introducción](#_tasotndz29m)

[2 Activación](#_j148dsnmt0uh)

[3 Generando nuestro primer diagrama](#_gxsaspplefso)

[3.1 Integración con el editor](#_bcuyeygde98c)

[3.2 El diagrama se actualiza automáticamente](#_1um2hfty6ky3)

[3.3 Por qué es un diagrama estático y no “de clases”](#_tvtl15njfexf)

[3.4 Exportar el diagrama](#_q73d08gh5hjp)

[3.5 Asociación entre wlk y diagrama](#_ux8lpx7lcwzb)

[4 Configuraciones básicas](#_wuj7pz9bo74z)

[4.1 Mostrar variables](#_q35r69397eet)

[4.2 Recordar posiciones y tamaños de los elementos](#_rfmy6ef0aimj)

[4.3 Eliminar configuración de tamaños y posiciones de elementos](#_e7ossguwn135)

[5 Nuestro valor agregado](#_s7qot3c36w7b)

[5.1 Ocultando métodos](#_5nxzj92qeq5x)

[5.2 Ocultar una clase](#_yozpd61meduc)

[5.3 Ocultar una variable](#_a20q9mantclr)

[5.4 Volver a mostrar un elemento](#_kl7jo3y94yyk)

[5.5 Volver a mostrar atributos y métodos ocultos](#_h44vpntbfd5a)

[5.6 Volver a mostrar atributos y métodos de un elemento particular](#_g0ki7253zqu9)

[5.7 Panel de Zoom](#_myllpab8yws2)

[6 Relaciones](#_2ptbtl15stsh)

[6.1 Relaciones de asociación](#_9b1org64y683)

[6.2 Relaciones de asociación autorreferenciales](#_a88s7n2obrnf)

[6.3 La relación de asociación queda activada](#_hoy4xns8ma8s)

[6.4 Eliminar una relación de asociación](#_zdn703v9ugry)

[6.5 Eliminar todas las relaciones de asociación y dependencia](#_iawtuhorocc)

[6.6 Relaciones de dependencia](#_nv8m18vyzgdr)

[7 Componentes externos](#_ccbspcvfyb3k)

[7.1 Agregar componentes externos](#_lweicpkrhn6t)

[7.2 Eliminar elementos externos](#_46nw0klc2l79)

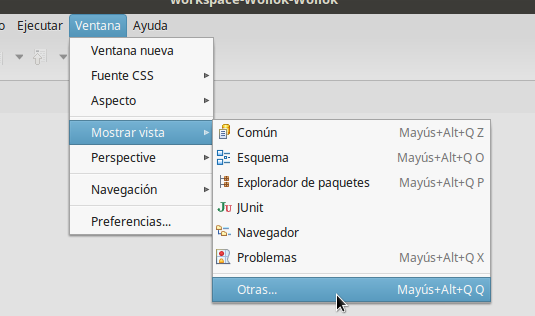
[8 Resumen](#_kqlnoh411f0b)

# 1 Introducción

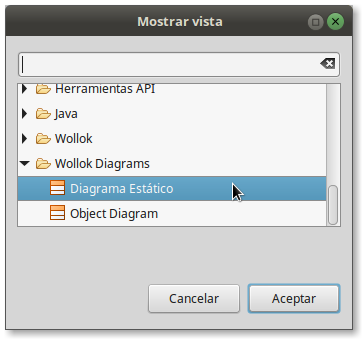
En el presente capítulo estudiaremos el diagrama estático que trae la herramienta Wollok para documentar el diseño de un trabajo general. Asumiremos que el idioma del entorno es español, por lo que algunos nombres y mensajes pueden diferir basado en el idioma que tenga instalado.

# 2 Activación

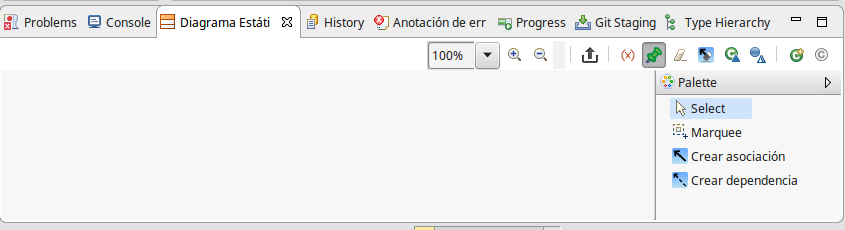
En el caso que el diagrama se encuentre oculto, se puede activar desde el menú Ventana: Ventana | Mostrar vista | Otras...



Y luego dentro del conjunto “Wollok diagrams” está el Diagrama Estático



Al dar Aceptar se visualizará el Diagrama dentro del entorno de trabajo:



Inicialmente no muestra nada, ya que trabaja siempre asociado a un archivo.

# 3 Generando nuestro primer diagrama

El diagrama estático

* está pensado para mostrar los elementos estáticos de un archivo wlk
* no está pensado para tests o programas

Podemos abrir un archivo wlk, y esto muestra todas las definiciones de ese archivo (objetos, clases y mixins). Supongamos que tenemos este archivo:

**class** Ave {

**var** energia = 0

**method** volar() { energia -= 15 }

**method** comer(gramos) { energia += gramos }

**method** energia() = energia

**method** energia(\_energia) { energia = \_energia }

}

**class** Gorrion **inherits** Ave {

**var** color

**var** carreteos = 0

**method** color() = color

**method** color(\_color) { color = \_color }

**override** **method** volar() {

**super**()

color = "gris"

}

**method** carretear() { carreteos = carreteos + 1 }

}

**class** Torcaza **inherits** Ave {

**override** **method** comer(gramos) { energia += 50 }

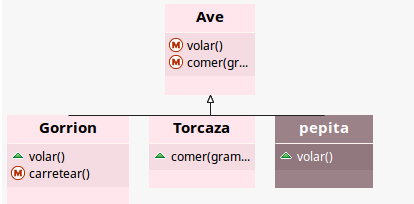
}

**object** pepita **inherits** Ave {

**override** **method** volar() { energia -= 10 }

}

Esto produce por defecto el siguiente diagrama estático:

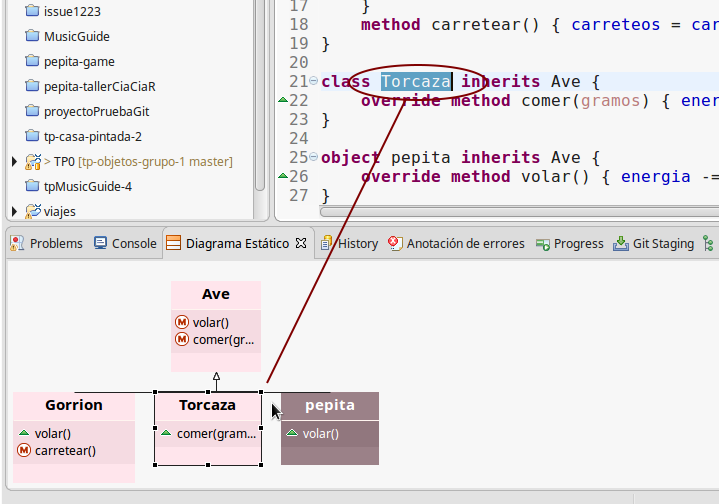


Algunas observaciones:

* las subclases se ordenan alfabéticamente
* no aparecen las variables
* los accessors (getters y setters) se ocultan, siguiendo la convención de que deben tener el mismo nombre que la referencia
* los métodos redefinidos aparecen con una flecha verde
* el diagrama se ordena de arriba hacia abajo y de izquierda a derecha

## 3.1 Integración con el editor

El diagrama se integra con el editor, de manera que al hacer click sobre un método, va directamente a la definición de dicho método (lo mismo con la clase, o las variables cuando necesitemos verlas).



## 3.2 El diagrama se actualiza automáticamente

Una vez asociado a un archivo, el diagrama se actualiza con cualquier modificación de código: si cambiamos el nombre de la clase, de un método o una referencia, el diagrama de clases se acomoda para que se muestren los elementos correctamente: si escribimos

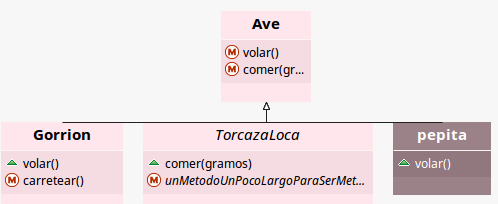
**class** TorcazaLoca **inherits** Ave {

**override** **method** comer(gramos) { energia += 50 }

**method** unMetodoUnPocoLargoParaSerMetodo

}

Eso se ve reflejado de la siguiente manera en el diagrama:



Como vemos

* TorcazaLoca no define aún “unMetodoUnPocoLargoParaSerMetodo”, por lo tanto se considera clase abstracta y se marca en cursiva.
* También marca “unMetodoUnPocoLargoParaSerMetodo” como método abstracto, ya que no tiene implementación

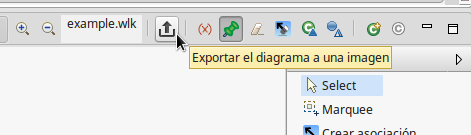
## 3.3 Por qué es un diagrama estático y no “de clases”

Como vimos anteriormente, pepita es un wko y aparece en el diagrama estático, que ya no es exclusivamente de clases. También los mixins[[1]](#footnote-0) escapan de la definición inicial de clase.

Sí es estático ya que su intención es mostrar la estructura general de la solución, sin detenerse a pensar en casos concretos para todos los elementos que participan del diagrama.

## 3.4 Exportar el diagrama

En cualquier momento que necesites podés exportar el diagrama en formato png, mediante el botón “Exportar el diagrama a una imagen” que está en la barra de herramientas del diagrama:



Esto permite guardar el png en un directorio.

## 3.5 Asociación entre wlk y diagrama

Una de las preguntas iniciales que uno se hace es: ¿puedo pasar el diagrama a un compañero? Sí, podés, ya que hay un paralelismo entre el archivo wlk y su diagrama, y se almacena dentro del proyecto.

+ proyectoTP

+ src

- example.wlk

+ .diagrams

- example.wsdi

Cuando subas los cambios, no ignores la carpeta .diagrams para que tus compañeros al bajar los cambios del repositorio puedan visualizar el diagrama exactamente igual que vos.

**Para esto ambos deben usar exactamente la misma versión de Wollok**. O de lo contrario será necesario que cada uno mantenga su propio diagrama

* ignorando el directorio .diagrams en el control del código que hace el repositorio
* y borrando los archivos .wsdi que estén en dicha carpeta

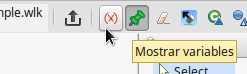
ya que de lo contrario te va a aparecer un mensaje de error.

# 4 Configuraciones básicas

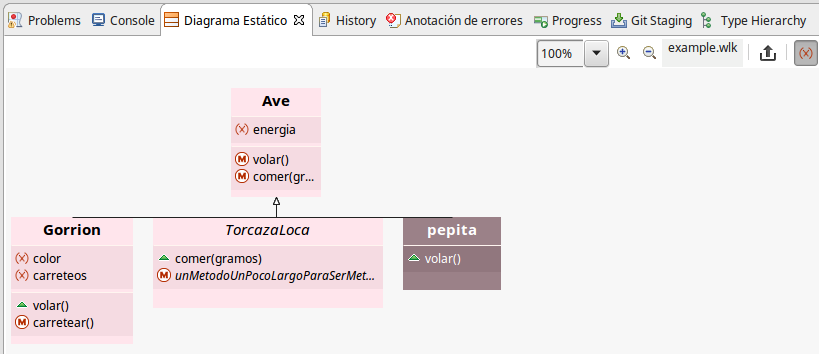
La barra de herramientas del diagrama estático tiene muchas variantes, vamos a explicar a continuación cada una de ellas.

## 4.1 Mostrar variables

Por defecto, las variables no se muestran en el diagrama. Para activar la visualización de dichas referencias, se debe dar un click en el botón “Mostrar variables” que tiene un ícono rojo con una equis entre paréntesis:



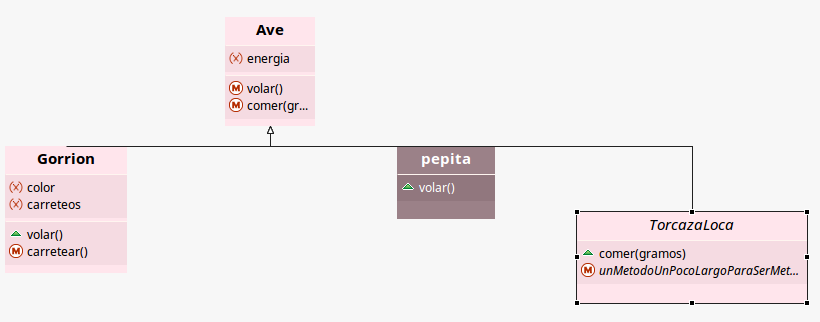
El botón es *toggle*, esto significa que deja activa la visualización de variables. El diagrama ahora muestra:



Y como vemos el botón “Mostrar variables” queda presionado. Para desactivar esta opción volvemos a presionar el mismo botón.

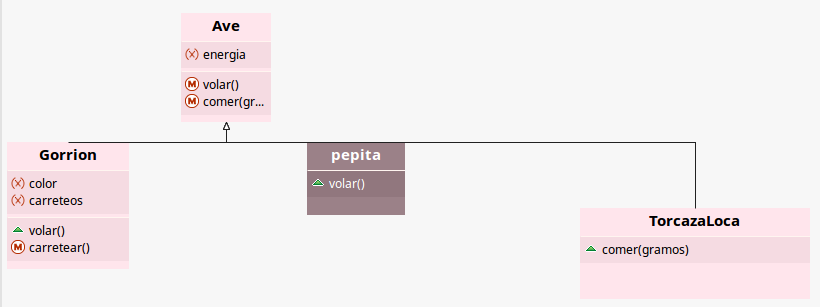
## 4.2 Recordar posiciones y tamaños de los elementos

Por defecto está activada esta opción. Esto implica que al mover un elemento de lugar, o agrandarlo/achicarlo, tanto la posición como el tamaño se almacenan dentro de la configuración. Entonces cuando el código cambia, todos los elementos se redibujan a excepción del componente que previamente movimos. Por ejemplo, llevemos *TorcazaLoca* a la derecha de *pepita*:

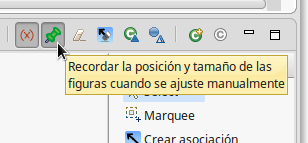


Esto se hace al seleccionar TorcazaLoca mediante un click izquierdo del mouse, luego mantenemos apretado y arrastramos hacia la derecha el mouse, soltando en el lugar donde nosotros queremos ubicar a dicha clase.

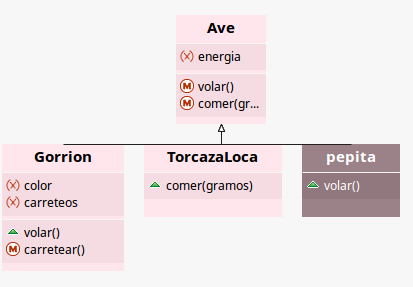
Si eliminamos el método con extensión larga, vemos que TorcazaLoca queda con un ancho grande:



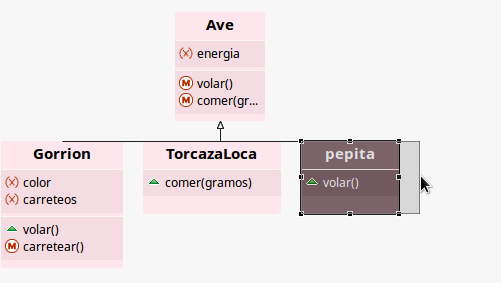
Esto es debido a que el toggle button que recuerda tamaños y posiciones de los elementos está activo:



Si desactivamos ese botón haciendo click, el diagrama se vuelve a dibujar desde cero cada vez:



Entonces no importa si movemos un elemento hacia la izquierda, o derecha, o lo agrandamos (posicionándonos sobre uno de los extremos, haciendo click y arrastrando hasta el tamaño deseado):

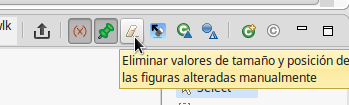


Al escribir nuevamente un espacio en el editor, el diagrama volverá a dibujarse desde cero, perdiendo la configuración. Por este motivo es importante dejar activo el botón que recuerda tamaños y posiciones, sabiendo que

* **si se agregan nuevos métodos, o los métodos son más largos, debemos manualmente ajustar el diagrama**. Por este motivo es conveniente hacer ajustes previo a una entrega de trabajo práctico, o cuando sabemos que las definiciones de dicho archivo son estables.
* **si modificamos el nombre de la clase, la configuración que tenía se pierde**, ya que la referencia es por el nombre completo de la clase: paquete + clase, lo que se conoce como FQN o *full qualified name*. Por ejemplo si tenemos un paquete model y el nombre de la clase es Ave, el FQN es model.Ave

## 4.3 Eliminar configuración de tamaños y posiciones de elementos

Supongamos que modificamos muchas posiciones o tamaños de los elementos, ¿cómo podemos eliminar esa configuración sin tener que presionar dos veces el botón “Recordar posición y tamaño de los elementos”? Mediante el botón que está al lado, que es una goma de borrar:



De esa manera el diagrama vuelve a redibujarse pero seguimos teniendo activada la opción de recordar los cambios manuales en las posiciones o tamaños de los componentes.

# 5 Nuestro valor agregado

Sabemos que las herramientas “automágicas” necesitan cierta inteligencia adicional, para que el diagrama estático muestre información importante y no sea un simple vuelco de la misma información que está en el código.

Entonces el valor que le damos nosotros al diagrama es que podemos decirle

* qué elementos son importantes comunicar
* cuáles son las responsabilidades más importantes que cumplen los componentes (los métodos que queremos comunicar)
* y eventualmente si queremos mostrar las referencias, cuáles son las variables que resaltan en nuestra solución

Esto lo podemos hacer ocultando

* clases, objetos o mixins
* referencias
* métodos

## 5.1 Ocultando métodos

Por ejemplo, definimos dos métodos adicionales en TorcazaLoca:

**class** TorcazaLoca **inherits** Ave {

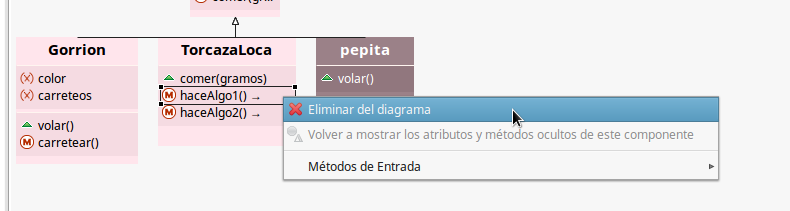
**override** **method** comer(gramos) { energia += 50 }

**method** haceAlgo1() = **return** energia + 1

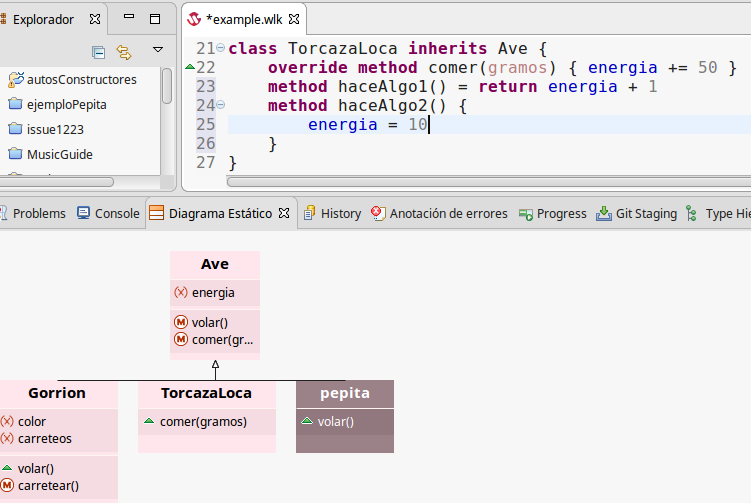
**method** haceAlgo2() = **return** energia + 2

}

Pero son métodos internos, o bien que podrían formar parte de la implementación de otro requerimiento más general. Entonces vamos a ocultar ambos métodos. Seleccionamos el método haceAlgo1(), luego botón derecho y Eliminar del diagrama:



Hacemos lo mismo con el método haceAlgo2() y vemos que a partir de aquí no se visualiza más esos métodos:

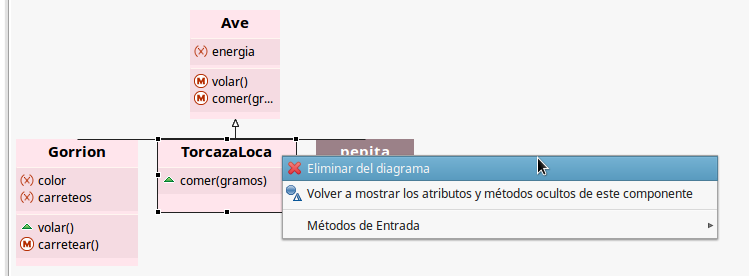


**Importante:** dado que la configuración almacena los nombres de los métodos a ocultar

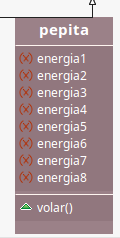
* si renombran algún método oculto, éste volverá a verse en el diagrama
* si además algún método nuevo tiene el nombre de algún método oculto que ha sido renombrado, éste no se verá en el diagrama. Por ejemplo: si ocultamos hacerAlgo1(), lo renombramos a hacer() y luego escribimos un método hacerAlgo1(), en el diagrama se visualizará hacer() y no hacerAlgo1()

## 5.2 Ocultar una clase

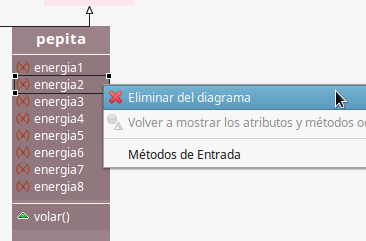
Con un botón derecho sobre la clase, se la puede eliminar del diagrama



## 5.3 Ocultar una variable

Si estamos mostrando variables de un elemento, es un problema cuando un componente tiene muchas referencias.

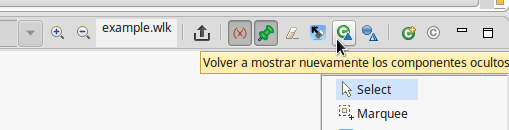
En ese caso podemos hacer click sobre las variables que no queremos mostrar y eliminarlas del diagrama.



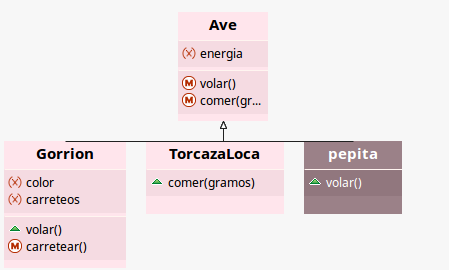
Las variables a ocultar se guardan en la configuración en base al nombre, entonces hay que tener en cuenta en caso de renombrar que volverán a aparecer y desaparecerá cualquier variable nueva que tenga el nombre de alguna variable oculta.

## 5.4 Volver a mostrar un elemento

Para visualizar nuevamente un elemento oculto, tenemos una opción en la barra de herramientas, representado por el ícono de clase (una C enmarcada en un círculo verde) con un triángulo azul en la diagonal derecha de abajo:

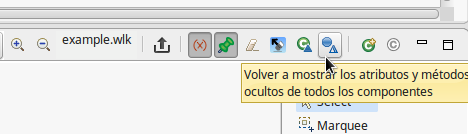


Esto vuelve a mostrar TorcazaLoca en el diagrama:

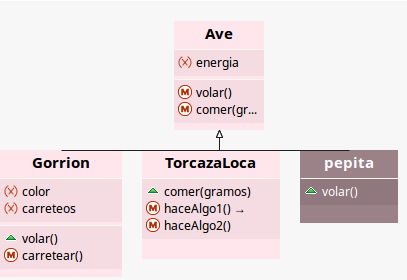


## 5.5 Volver a mostrar atributos y métodos ocultos

En TorcazaLoca todavía no aparecen dos métodos ocultos hacerAlgo1() y hacerAlgo2(). Para volverlos a visualizar en el diagrama, tenemos la opción en la barra de herramientas, representado por un círculo azul con un triángulo azul en la diagonal derecha de abajo:

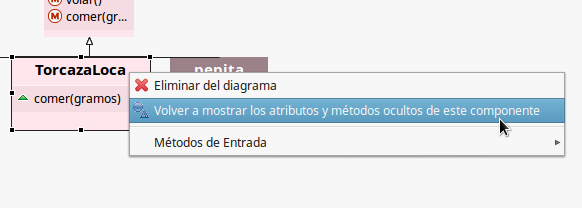


Ahora sí, los dos métodos de TorcazaLoca que estaban ocultos se visualizan nuevamente:

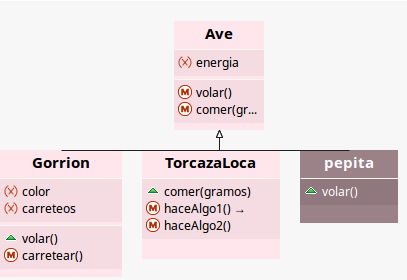


## 5.6 Volver a mostrar atributos y métodos de un elemento particular

La barra de herramientas hace aparecer todos los atributos y métodos ocultos de todos los componentes. Esto puede ser demasiado abarcativo cuando nos equivocamos y ocultamos involuntariamente un atributo o método de un componente. Entonces podemos seleccionar la clase, dar un click derecho y seleccionar la opción “Volver a mostrar los atributos y métodos ocultos de este componente”:



De esa manera, solo los atributos y métodos ocultos de TorcazaLoca se volverán a visualizar, dejando los atributos y métodos de los otros elementos ocultos:

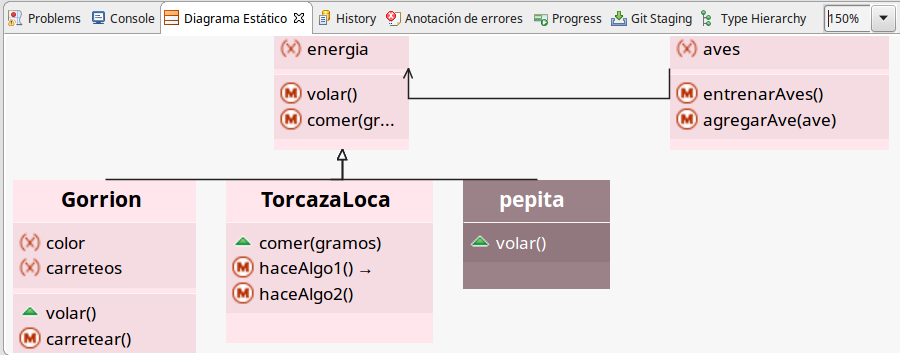


## 5.7 Panel de Zoom

El diagrama estático cuenta con un panel de zoom, para aumentar o disminuir el tamaño de las clases visualizadas. Esto se puede lograr mediante el cuadro de selección (que permite ingresar un número) o bien con los botones + / - de la barra de herramientas:



Al presionar el botón + podremos ver el efecto que tiene:



# 6 Relaciones

Las relaciones de herencia son inmediatamente detectadas por el diagrama estático. Por el momento hay otras dos relaciones que debemos incorporar manualmente:

* relaciones de asociación
* relaciones de dependencia

En caso de dudas puede verse el apunte que explica [cómo comunicar las decisiones de diseño](https://drive.google.com/open?id=1eXLlNppAX-7E2M8Xxs0MCckdn4XVEYmeQNaS_E1RqTc).

## 6.1 Relaciones de asociación

Una relación de asociación implica que un objeto A tiene una referencia a un objeto B. En nuestro ejemplo podemos incorporar un entrenador, que conoce a una lista de aves:

**class** Entrenador {

**const** aves = []

**method** entrenarAves() {

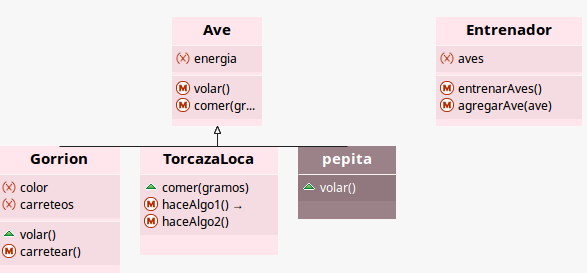
aves.forEach { ave => ave.volar() }

}

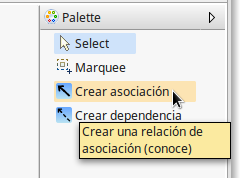
**method** agregarAve(ave) { aves.add(ave) }

}

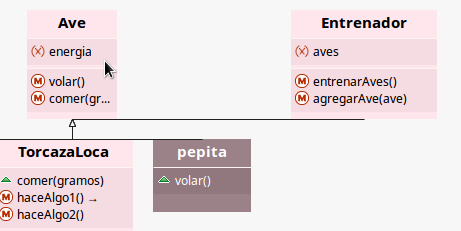
Al agregar esta clase, el diagrama se ve de la siguiente manera:



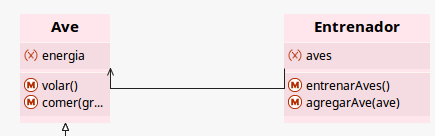
Aquí nos está faltando marcar la relación que tiene Entrenador con Ave, asumiendo que aves será una lista de especializaciones de Ave. Para ello, vamos a la paleta (*Palette*) que está ubicada a la derecha del diagrama y seleccionamos la herramienta “Crear asociación”



A continuación debemos hacer un click en el primer elemento, y otro click en el segundo: esto marcará el origen y destino de la asociación

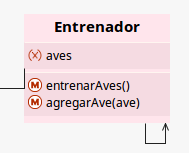


Con el segundo click se genera la relación de asociación en nuestro diagrama:



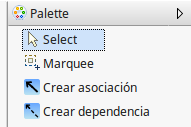
## 6.2 Relaciones de asociación autorreferenciales

Es posible generar una relación de asociación que vaya de un elemento a sí mismo, basta con seleccionar la herramienta “Crear asociación”, elegir un componente y luego hacer dos clicks:



## 6.3 La relación de asociación queda activada

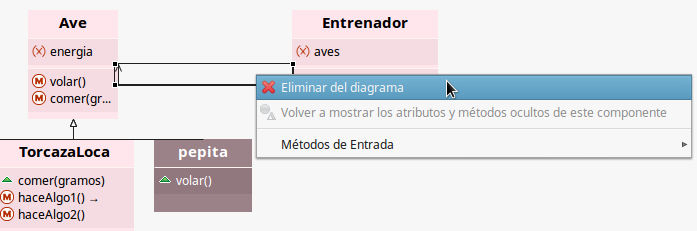
Esto implica que al seleccionar la herramienta “Crear asociación” para volver a mover elementos o seleccionarlos, debemos volver a presionar “Select”:



El sentido que tiene esto es que permite configurar todas las relaciones de asociación juntas.

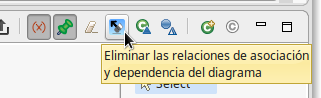
## 6.4 Eliminar una relación de asociación

Para eliminar una relación de asociación, primero hay que elegir la opción “Select” en la paleta (“*Palette*”) y luego hacer click sobre la relación



## 6.5 Eliminar todas las relaciones de asociación y dependencia

Para no tener que eliminar las relaciones de asociación y dependencia una por una, puede presionar el botón que está en la barra de herramientas, representado por una goma de borrar sobre una flecha que enmarca una relación:



Esto permite rápidamente recomenzar la tarea de definir las relaciones de asociación en el diagrama.

## 6.6 Relaciones de dependencia

Aunque menos frecuente, una relación de dependencia marca un conocimiento más débil de un objeto hacia otro, que se da

* porque un objeto A tiene una variable local de B
* o bien porque A instancia un objeto B

sin almacenarlo en una variable de instancia.

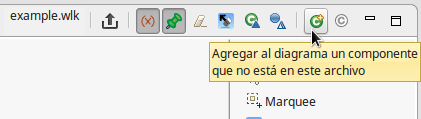
La operatoria es similar a la creación y eliminación de relaciones de asociación, agregando estas restricciones:

* si definieron que Entrenador conoce a Ave, no tiene sentido que Entrenador use Ave, es decir que una relación de asociación hace innecesario que se defina una relación de dependencia adicional. En el diagrama estático eso se valida, por lo que no van a poder hacerlo.
* no es posible marcar una relación de asociación de un objeto hacia sí mismo.

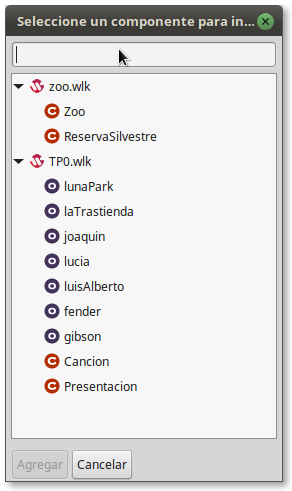
# 7 Componentes externos

## 7.1 Agregar componentes externos

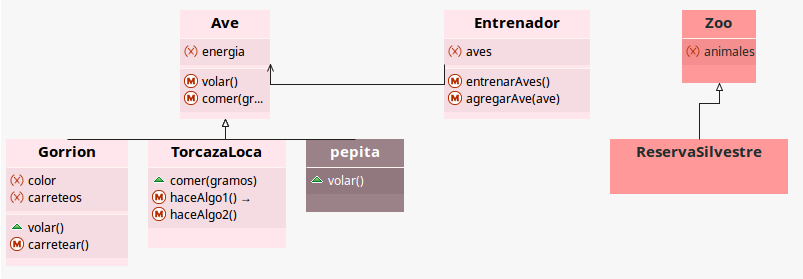
Lo común es que hagamos definiciones en diferentes archivos wlk. Para tener un diagrama en donde podamos dar un vistazo general a nuestra solución, el diagrama estático permite agregar un elemento externo:



Esto habilita un menú contextual, donde podemos elegir uno o varios elementos a incorporar:

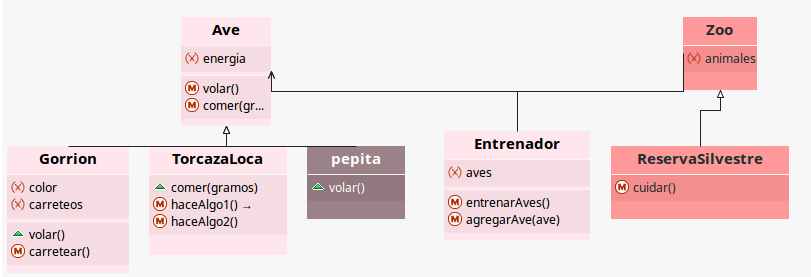


Aquí se muestran los elementos de cada archivo wlk que esté en el proyecto actual. Podemos seleccionar ReservaSilvestre, por ejemplo y el efecto que tiene es traer tanto ReservaSilvestre como Zoo, que es su superclase:



Cada vez que actualicemos el código en el archivo zoo.wlk, y volvamos luego al archivo example.wlk, estos cambios se visualizarán en el diagrama. Podemos

* ocultar atributos, métodos o componentes de archivos externos
* también definir relaciones de asociación o dependencia

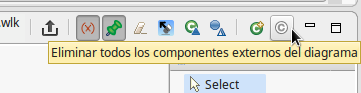


Aquí vemos que tanto Entrenador como Zoo conocen a Ave.

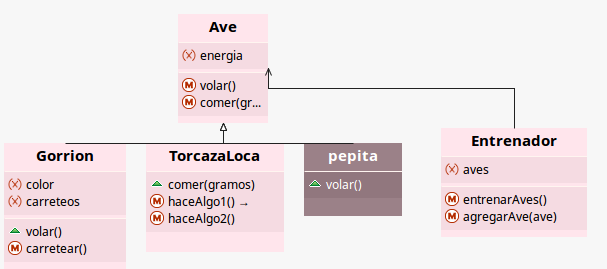
Para diferenciar los elementos que están en el archivo de los que son externos, aparecen en el diagrama estático con otro color.

## 7.2 Eliminar elementos externos

El diagrama presenta una opción en la barra de herramientas, para eliminar componentes situados en archivos externos:



El diagrama estático vuelve a tener los elementos del archivo original:



# 8 Resumen

En el presente apunte hemos repasado al diagrama estático como una herramienta que ayuda a la comunicación del diseño, recordando que la parte más importante es el valor agregado irreemplazable que tiene la persona que modela una solución.

1. no te preocupes si no viste este concepto, es una idea avanzada para reutilizar comportamiento [↑](#footnote-ref-0)