Unidad 4. Diagrama de clases.

Entornos de desarrollo
1º Desarrollo de Aplicaciones Multiplataforma

Diagramas de clases

Representa los **elementos estáticos del sistema**, sus **atributos** y **comportamientos**, y cómo se **relacionan** entre ellos.

Contiene las clases del dominio del problema, y a partir de éste se obtendrán las **clases que formarán** después el programa informático que dará solución al problema.

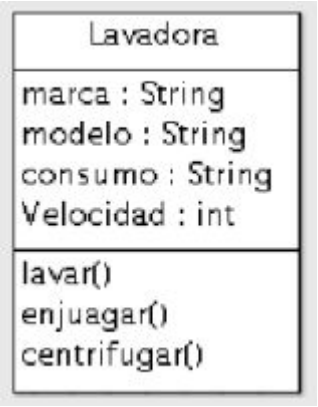
Clases y objetos

Clases: abstracciones del dominio del sistema que representan elementos del mismo mediante una serie de características, que llamaremos atributos, y su comportamiento, que serán métodos.

Los **atributos y métodos** tendrán una visibilidad que determinará quién puede acceder al atributo o método.

A partir de una clase se pueden crear **objetos**, también llamados instancias de la clase.

Clases y objetos



Los **objetos** son concreciones de una clase: todos los objetos de una clase comparten las **mismas operaciones**, pero sus **atributos** pueden tener **valores distintos**.

Las clases en UML se representan con un **rectángulo** dividido en 3 partes:

- la caja superior contiene el **nombre de la clase**,
- la del medio los **atributos**,
- y la inferior las **operaciones**

Atributos

Un **atributo** representa alguna propiedad de la clase de encuentra en todas las instancias de la clase.

La **caja central** de la clase contiene la lista de atributos, uno por cada línea.

La lista de atributos es opcional, así que se puede omitir, si se desea, dejando la caja en blanco.

Atributos

La forma de definición general es la siguiente:

[visibilidad] Nombre [: Tipo] [= valor_inicial]

visibilidad puede ser:

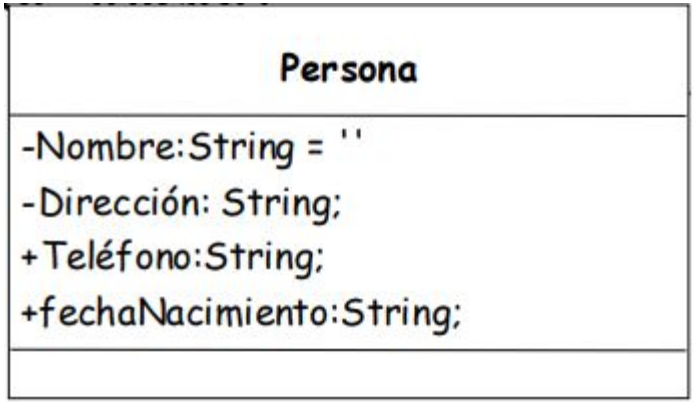
+ = Pública

 # = Protegida

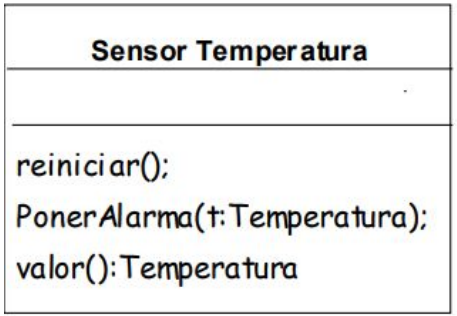
- = Privada (por defecto)

Los tipos permitidos para los atributos son tipos básicos (**integer, real, char, string, etc.**)

Atributos


Persona
-Nombre:String = ''
-Dirección: String;
+Teléfono:String;
+fechaNacimiento:String;

Operaciones



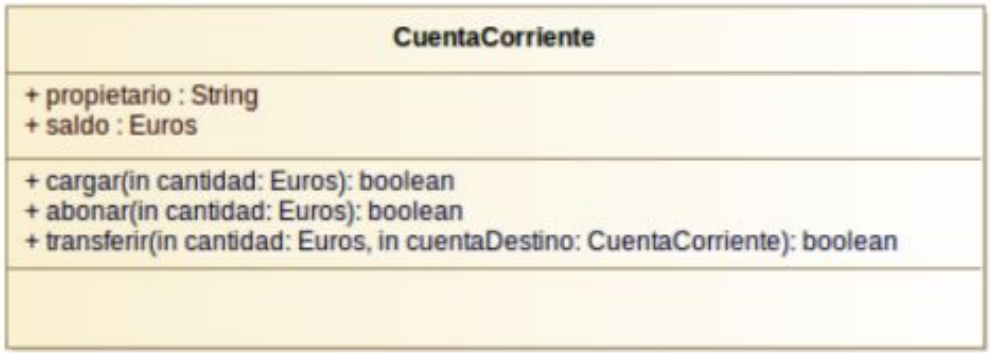
Una operación se corresponde con un servicio que puede ser requerido a cualquier objeto de la clase.

Una operación es una función o transformación que puede ser aplicada a los objetos.

Se definen de la siguiente forma:

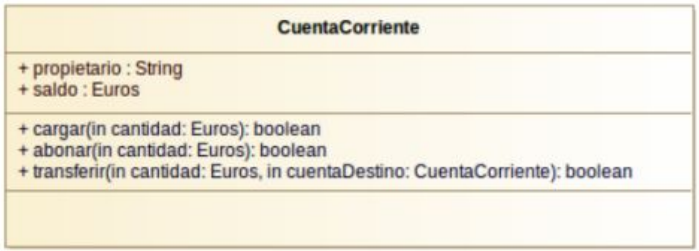
[visibilidad] Nombre([comalista_parámetros]) [: Tipo_Resultado]

Operaciones



Operaciones

La figura anterior muestra una clase de ejemplo: CuentaCorriente con 3 operaciones: cargar, abonar y transferir. Las 3 operaciones devuelven un valor de tipo boolean. Las operaciones cargar y abonar toman como único parámetro de entrada la cantidad de tipo Euros. La operación transferir toma 2 parámetros, ambos de entrada: cantidad de tipo Euros y cuentaDestino, de tipo CuentaCorriente.



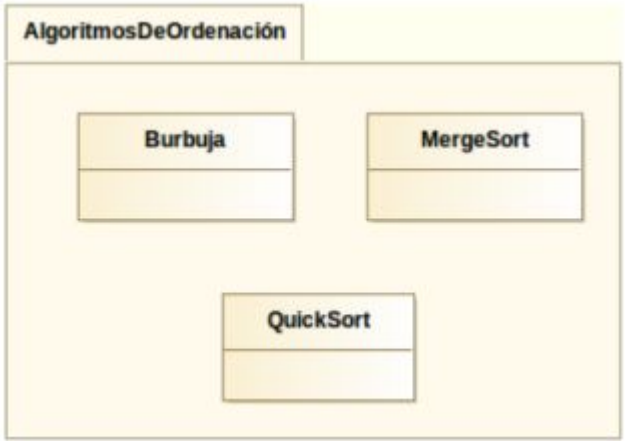
Paquetes

Si modelamos un **sistema complejo**, inevitablemente aparecerán muchas clases en el modelo.

Los paquetes (packages) son **contenedores que permiten agrupar clases con características o funcionalidad común**.

En UML, un diagrama se representa con un **recuadro con una solapa** (similar a una carpeta clasificadora). En la solapa se escribe el nombre del paquete y en el interior del recuadro se dibujan las clases pertenecientes al paquete

Paquetes




```
package AlgoritmosDeOrdenacion;

class Burbuja   { ... }
class MergeSort { ... }
class QuickSort { ... }
```

Paquetes

Si modelamos un **sistema complejo**, inevitablemente aparecerán muchas clases en el modelo.

Los paquetes (packages) son **contenedores que permiten agrupar clases con características o funcionalidad común**.

En UML, un diagrama se representa con un **recuadro con una solapa** (similar a una carpeta clasificadora). En la solapa se escribe el nombre del paquete y en el interior del recuadro se dibujan las clases pertenecientes al paquete

Visibilidad

El **acceso a los elementos** (operaciones o atributos) desde otras clases depende de la **visibilidad**.

- **Pública** (public): es visible desde cualquier otra clase.
- **Protegida** (protected): es visible sólo por elementos de su propia clase o las clases hijas.
- **Privada** (private): sólo es visible por elementos de su propia clase.
- **Paquete** (package): sólo es visible por otras clases que pertenezcan al mismo paquete.

Visibilidad

La visibilidad se **representa en el diagrama** con un símbolo antepuesto al nombre del atributo u operación:

- **+: pública**
- **#: protegida**
- **−: privada**
- **~: paquete**

Visibilidad

```
class Alumno {
  // atributos
  private String nombre;
  private String apellido;
  private int edad;

  // metodos de acceso
  public void setNombre(String nombre) {
    this.nombre = nombre;
  }
  protected void setApellido(String apellido) {
    this.apellido = apellido;
  }
  private void setEdad(int edad) {
    this.edad = edad;
  }
  String toString () {
    return(nombre + " tiene " + edad + " a~nos.");
  }
}
```

Relaciones

En el mundo real muchos **objetos están vinculados o relacionados entre sí**, los vínculos se corresponden con asociaciones entre los objetos.

En una aplicación mínimamente compleja habrá varias **clases cuyos métodos se llamarán unos a otros**.

Para que un método de la clase A pueda llamar a otro método de la clase B, la clase A debe **poseer una referencia a un objeto de la clase B**. Esta relación de posesión se representa con **líneas que unen las distintas clases en el diagrama**.

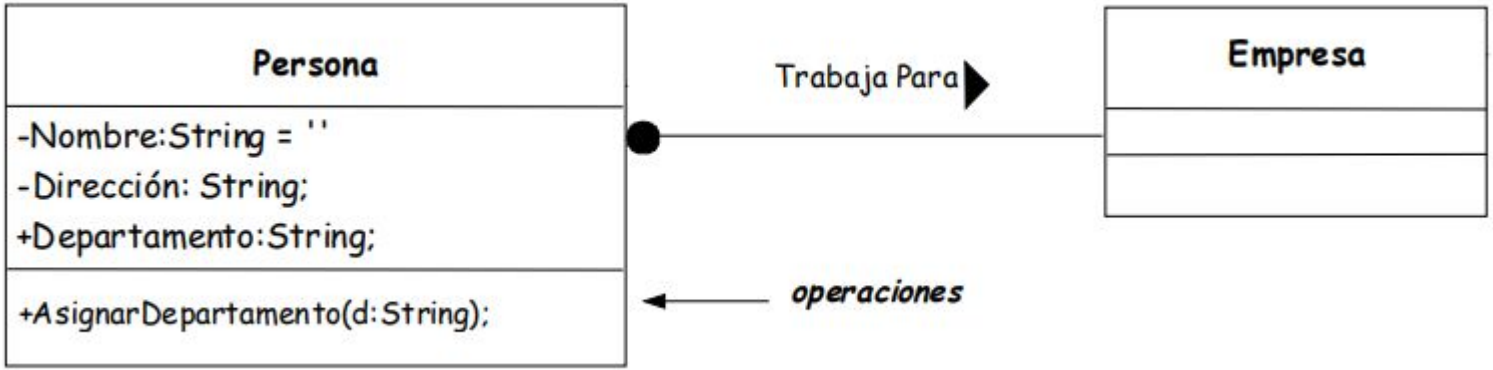
Relaciones: asociación

Es el tipo de relación **más frecuente** entre clases.

Se representa como una **línea continua**, que puede estar o no acabada en una flecha, según la navegabilidad.

Las asociaciones tienen un **nombre** y como ocurre con las clases, este es un reflejo de los elementos de la asociación.

Relaciones: asociación



Relaciones: asociación

Multiplicidad

Las asociaciones poseen una **cardinalidad** llamada multiplicidad. La multiplicidad **situada en un extremo de una asociación** indica a cuántas instancias de la clase situada en ese mismo extremo está vinculada una instancia de la clase situada en el extremo opuesto.

En uno de los extremos de la asociación, es posible especificar la **multiplicidad mínima y la máxima** con el fin de indicar el intervalo de valores al que deberá pertenecer siempre la multiplicidad.

Relaciones: asociación

Multiplicidad

0..1: Cero o una vez

1 : una y solo una vez

*: De cero a varias veces

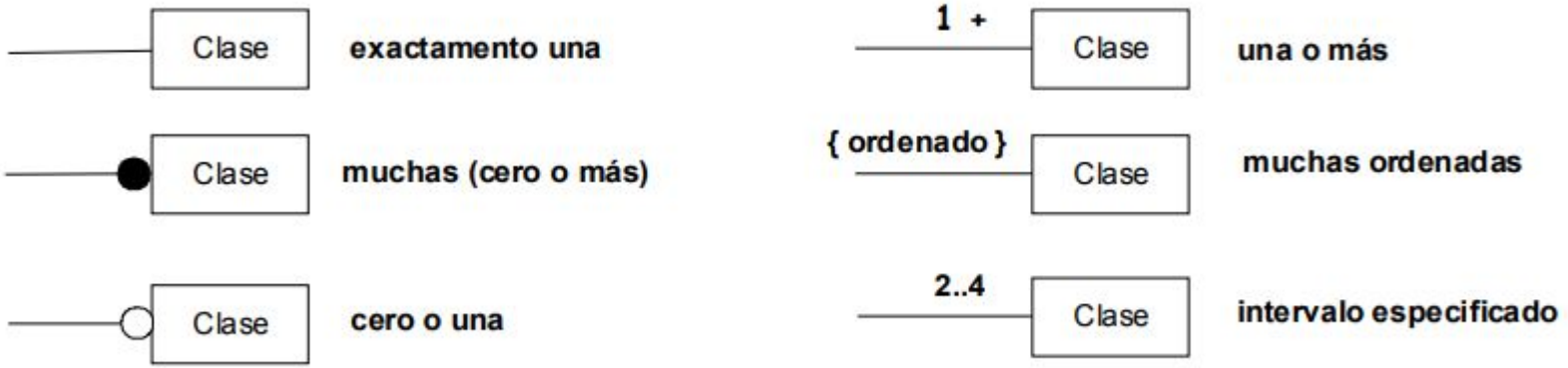
1..*: De una a varias veces

M..N : Entre M y N veces

N: N veces

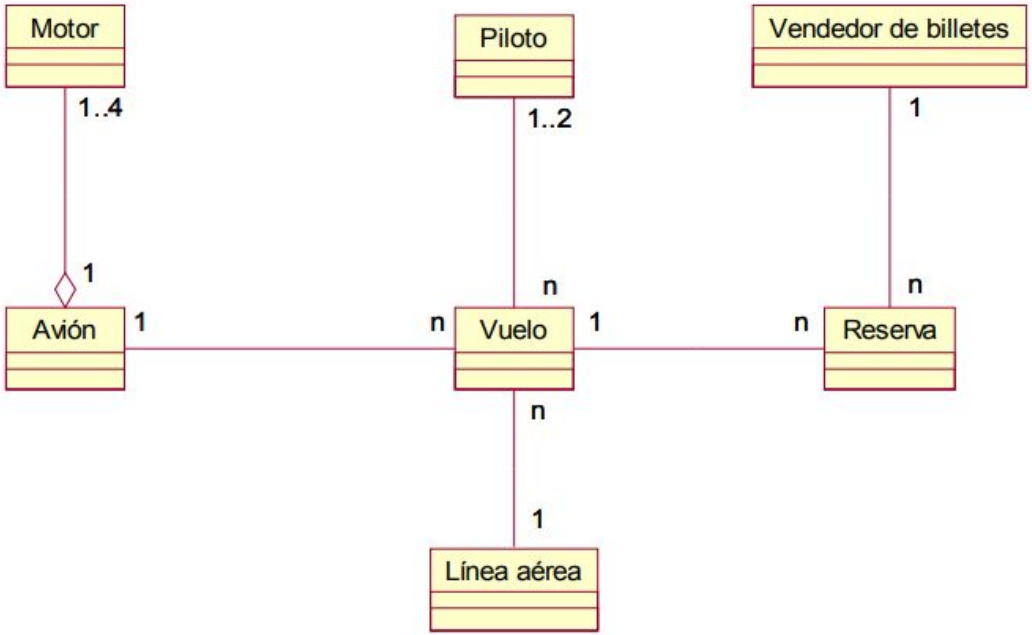
Relaciones: asociación

Multiplicidad



Relaciones: asociación

Multiplicidad



Relaciones: asociación

Navegabilidad

La navegabilidad entre clases nos muestra que es posible pasar desde un objeto de la clase origen a uno o más objetos de la clase destino dependiendo de la multiplicidad.

Si la flecha apunta de la ClaseA a la ClaseB, se lee como “ClaseA tiene una ClaseB” y se dice que la asociación o navegabilidad es **unidireccional**.

Si no dibujamos la flecha, se lee “ClaseA tiene una ClaseB y ClaseB tiene una ClaseA”, y se dice que la asociación o navegabilidad es **bidireccional**.

Relaciones: asociación

Ejemplo: asociación bidireccional

Supongamos una aplicación de gestión con las clases Cliente y Pedido.

En el siguiente diagrama, Cliente guarda información sobre los pedidos y Pedido tiene información sobre el cliente que lo realizó.

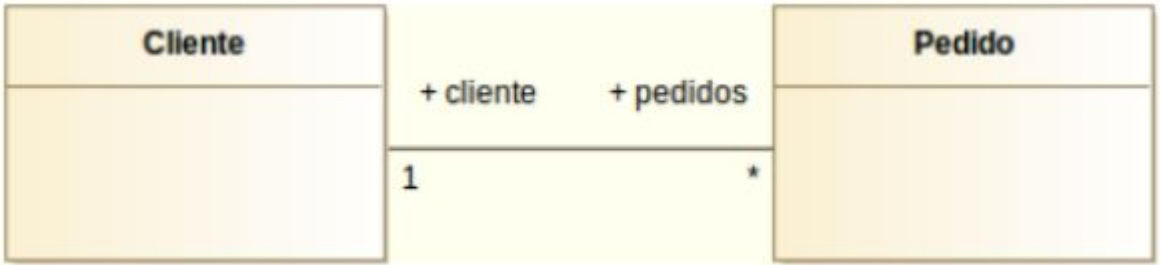
La navegabilidad es, por tanto, bidireccional.

Relaciones: asociación

Ejemplo: asociación bidireccional

```java
public class Cliente {
  public Vector<Pedido> pedidos;
}

public class Pedido {
  public Cliente cliente;
}
```



Relaciones: asociación

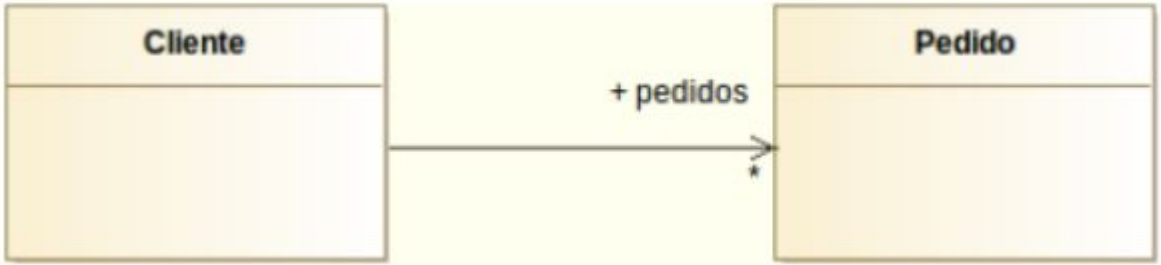
Ejemplo: asociación unidireccional

Puede ser que nuestra aplicación guarde la lista de pedidos en el objeto Cliente, pero no a la inversa.

En este caso la navegabilidad de la asociación sería unidireccional y su representación en UML sería:

Relaciones: asociación

Ejemplo: asociación unidireccional




```
public class Cliente {
  public Vector<Pedido> pedidos;
}

public class Pedido {
  // el pedido NO guarda una referencia al cliente
}
```

Relaciones: agregación

Representación **jerárquica** que indica a un **objeto y las partes que lo componen** pero aun así deben tener existencia en sí mismo.

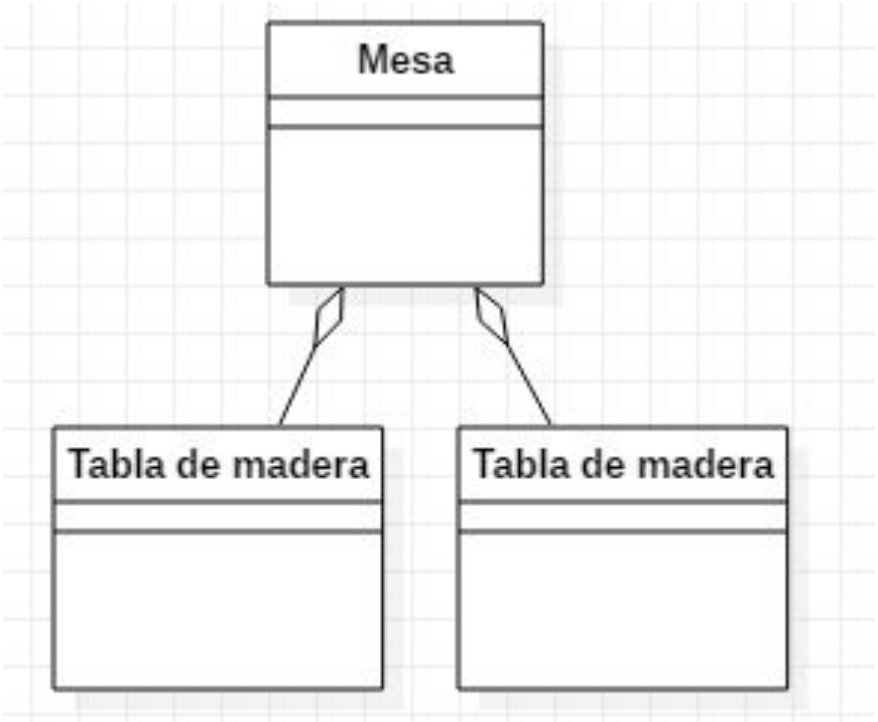
Se representa con una línea que tiene un **rombo** en la parte de la clase que es una agregación de la otra clase (es decir, en la clase que contiene las otras).

Relaciones: agregación

Un ejemplo de esta relación podría ser: “Las mesas están formadas por tablas de madera y tornillos o, dicho de otra manera, los tornillos y las tablas forman parte de una mesa”.

El tornillo podría formar parte de más objetos, por lo que interesa especialmente su abstracción en otra clase.

Relaciones: agregación



Relaciones: composición

Representa una **relación jerárquica** entre un **objeto y las partes que lo componen**, pero de una forma **más fuerte**.

En este caso, los **elementos que forman parte no tienen sentido de existencia cuando el primero no existe**.

Cuando el e**lemento que contiene los otros desaparece, deben desaparecer todos** ya que no tienen sentido por sí mismos sino que dependen del elemento que componen.

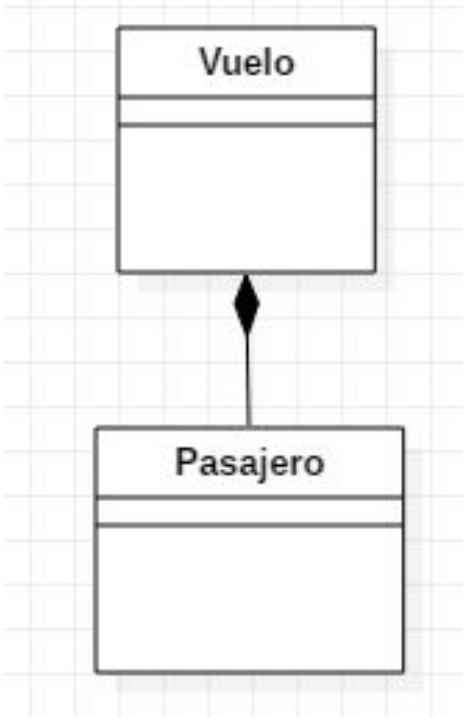
Relaciones: composición

Los **componentes no se comparten entre varios elementos**, esta es otra de las diferencias con la agregación.

Se representa con una **línea continua con un rombo relleno** en la clase que es compuesta.

Un ejemplo de esta relación sería: “Un vuelo de una compañía aérea está compuesto por pasajeros, que es lo mismo que decir que un pasajero está asignado a un vuelo”

Relaciones: composición

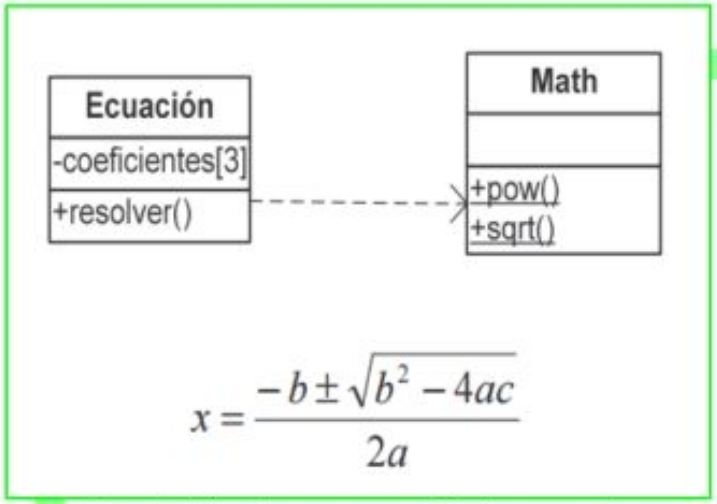


Relaciones: dependencia

Para representar que una clase requiere de otra para ofrecer sus funcionalidades.

Se representa con una flecha discontinua que va desde la clase que necesita la utilidad de la otra flecha hasta esta misma.

Relaciones: dependencia



Resolución de una ecuación de segundo grado.

Para resolver una ecuación de segundo grado hemos de recurrir a la función *sqrt* de la clase *Math* para calcular la raíz cuadrada.

Relaciones: herencia

Permite que una clase (**clase hija o subclase**) **reciba los atributos y métodos de otra clase** (**clase padre o superclase**).

Estos atributos y métodos recibidos se suman a los que la clase tiene por sí misma.

Se utiliza en relaciones “es un”.

Relaciones: herencia