***Relación de Agregación****:* se representa con una flecha y circulo vacio.

Se reconoce con “tiene” “conoce”. Ejemplo: la selva tiene animales (animales son parte de la selva).

Es unilateral, va en un solo sentido.

Las clases pueden existir individualmente.

***Relación de composición:*** se representa con una flecha y circulo lleno. Ejem. Los órganos del cuerpo humano. (ser humano compuesto por órganos)

Una clase está compuesta por otra, las clases no existen individualmente.

***Relación de Asociación:*** se dibuja con una flecha completa desde donde se asocian las clases (depende del contexto) El tipo de elemento que se contiene se representa list<clase>.

En el uml aparece la segunda clase como atributo de la primera. (Mascota:mascota)

***Relación de Uso:*** una clase usa a la otra. Es una colaboración temporal. (tocadiscos) Esta se muestra como parámetro en uno de sus métodos. Se representa con una flecha punteada.

Relación de Navegación:

Relación de Multiplicidad o cardinalidad: especifica el número de instancias de una clase que puede estar relacionadas con una única instancia de una clase asociada. La multiplicidad limita el número de objetos relacionados. (Se representa con un \* o N). Ejem. Persona – Trabajo.

Clases e instancias: clase es una abstracción, un molde que representa un concepto del objeto.

**Objeto** = instancia, la instancia viene a ser las diferentes variedades de una clase, las representaciones de los objetos concretos.

**Métodos** = responsabilidades.

Nombre del método o responsabilidad (parámetros -si es necesario-): tipo de dato (si va a devolver una responsabilidad). Puedes agregar un + o – para identificar si es público o privado.

**Generalización**: proceso mental de abstracción en que una clase hereda de un padre métodos en común con otras clases que heredan del mismo padre.

**Especialización**: proceso mental de abstracción en que una clase que hereda de un padre define otros métodos específicos y propios de la clase.

**Encapsulamiento**: ejemplo del CD, los atributos se mantienen en privado, se oculta la información para proteger la información y no se pueda modificar por cualquier clase.

**Protected** (#) modificador de visibilidad, permite que las clases hijas visualicen de forma intermedia los atributos o métodos del padre, más no las demás clases. No es buena práctica.

**Superclase**: como método hace referencia a su constructor.

Clases abstractas: no son algo concreto(no existen como tal en el mundo real) sirven para agrupar características especìficas comunes en clases que heredaran de esta.

Se pueden delcarar métodos pero que no serán implementados, es decir que no hacen nada en la clase abstracta. (No se puede instanciar objetos de una clase abstracta). Los métodos abstractos obligaràn a las subclases a sobreescribirlos para darles una implementación.

Que sería implementación?

Que es la sobreescritura?

Para implementar un método abstracto, se necesita sobreescribir(de hecho es lo que se hace, se sobreescribe el comportamiento abstracto). Se aplican las mismas reglas: respetar tipo, cantidad y orden de los parámetros. Si no lo hacemos no respetamos el contrato y habrá un error de compilación.

Cuando se implementan los métodos, estos deja de ser abstractos. Ya no se usa la palabra clave abstract.

Las clases abstractas pueden tener atributos y métodos concretos. Aun asi , solo los abstractos serán los que definan el contrato. Estos métodos serán suceptibles de ser reutilizados. Que no se puedan instanciar no significa que no pueda tener constructores, el objetivo es el mismo , podemos definir constructores para reutilizar código cuando heredamos de clase abstracta.

Redefinición de método, ¿cómo evitar equivocarnos?

Vimos que para redefinir un método(responsabilidad) de una superclase(clase padre), debemos sobrescribir el mismo, es decir, tenemos que escribir el mismo método con la misma firma en la subclase.

Sin embargo, ¿qué sucede, si cuando intentamos sobrescribir este método en la subclase tipeamos mal el nombre o nos olvidamos alguno de sus parámetros? Lo que pasaría es que la firma de este método sería diferente a la de la clase padre que intentábamos sobrescribir y este error no sería descubierto durante la compilación, excepto que sea un método abstracto.

Pero no nos preocupemos...

En Java existe una anotación llamada @override que podemos escribir en el método que estamos sobrescribiendo. Con esta anotación el compilador validará que ese método que estamos sobrescribiendo tenga una firma igual a algún método de la superclase y nos ayudará a no tener estos errores.

public class Doberman extends Perro{

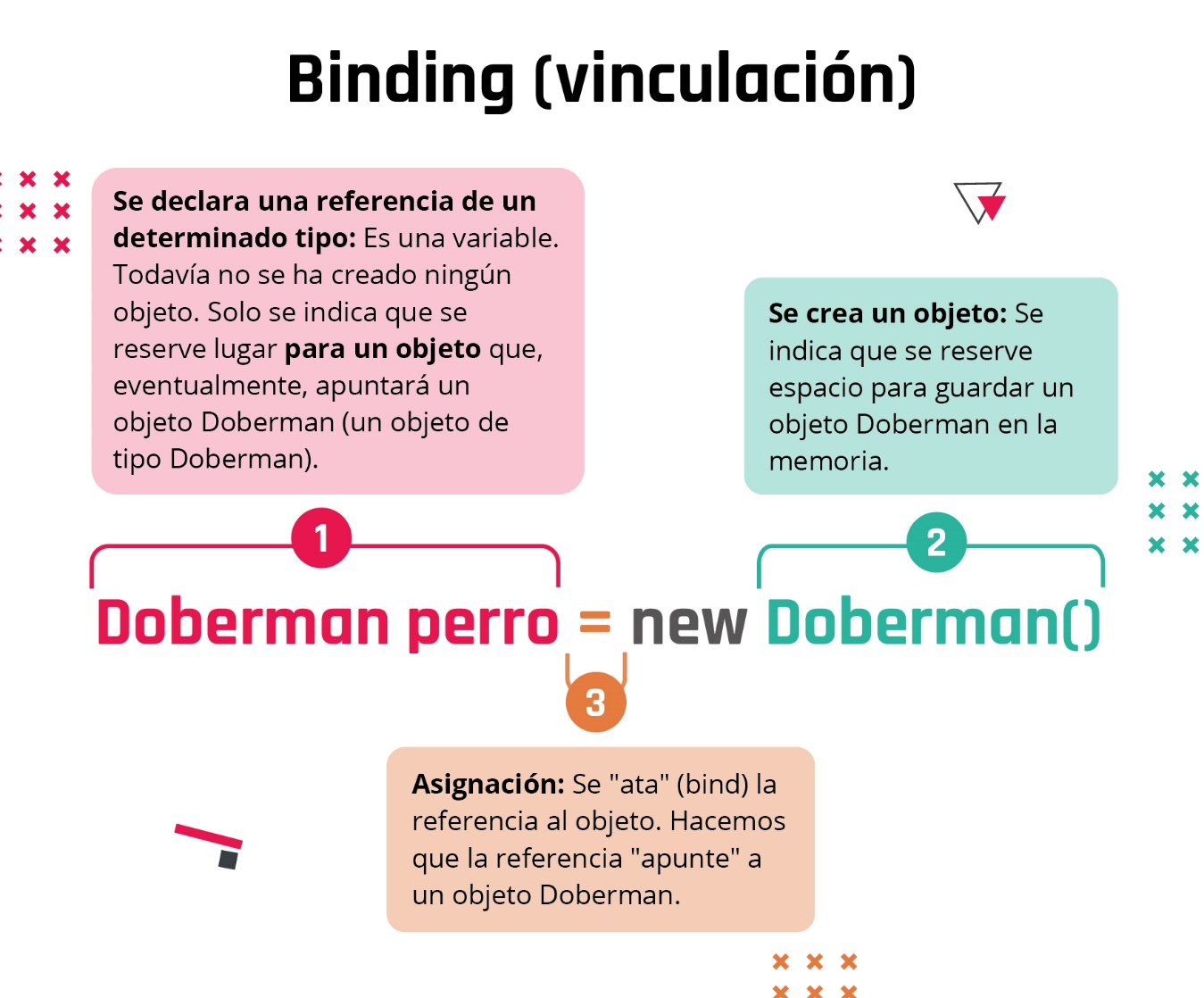
@Override

public String ladrar() {

return “GUAU GUAU!!!”;

}

}



Interfaces: son relaciones del tipo ”es un”, muy similar a la clase abstracta, se define con la palabra “interface” en lugar de “class”, todos sus métodos son abstractos por lo que no es necesario colocar la palabra.

Una interface establece un contrato, toda clase que implemente una interface , està obligada a implementar todos los métodos de esa interface.

Una clase puede solo heredar de una clase, pero puede implementar multiples interfaces.

La interface permite independizarse de las jerarquías y agregar comportamiento a una clase que no se obtenga de un nivel superior en la jerarquía, se “enchufa” lateralmente en la jerarquía.