

Bloque I:
Introducción a la Programación
1.2 Introducción a la POO

1r DAW Semipresencial
Programación

Maria José Lozano Pérez
IES Pere Maria Orts i Bosch

Tema 1.2: Introducción a la POO

- ❏ POO vs programación procedimental
- ❏ Introducción a la herencia
- ❏ Introducción a los métodos
- ❏ Una clase: métodos y variables de instancia
- ❏ Creando el primer objeto
- ❏ Utilizando main()
- ❏ UML: Diagrama de clases
- ❏ Ejemplos y prácticas

Programación Orientada a Objetos



Entramos en el mundo de los objetos. Dejamos atrás la programación clásica para mejorar. ¡Nos vemos!

Introducción a la POO

- Empezaremos a crear objetos propios
- Veremos las diferencias entre **clase** y **objeto**
- Veremos cómo los objetos nos ayudan a mejorar nuestra vida (como programadores)
- Un detalle importante: una vez que entres al mundo de la **POO**, **¡no podrás volver atrás!**

Cómo los objetos toman vida

🖥️ Imagina un problema:

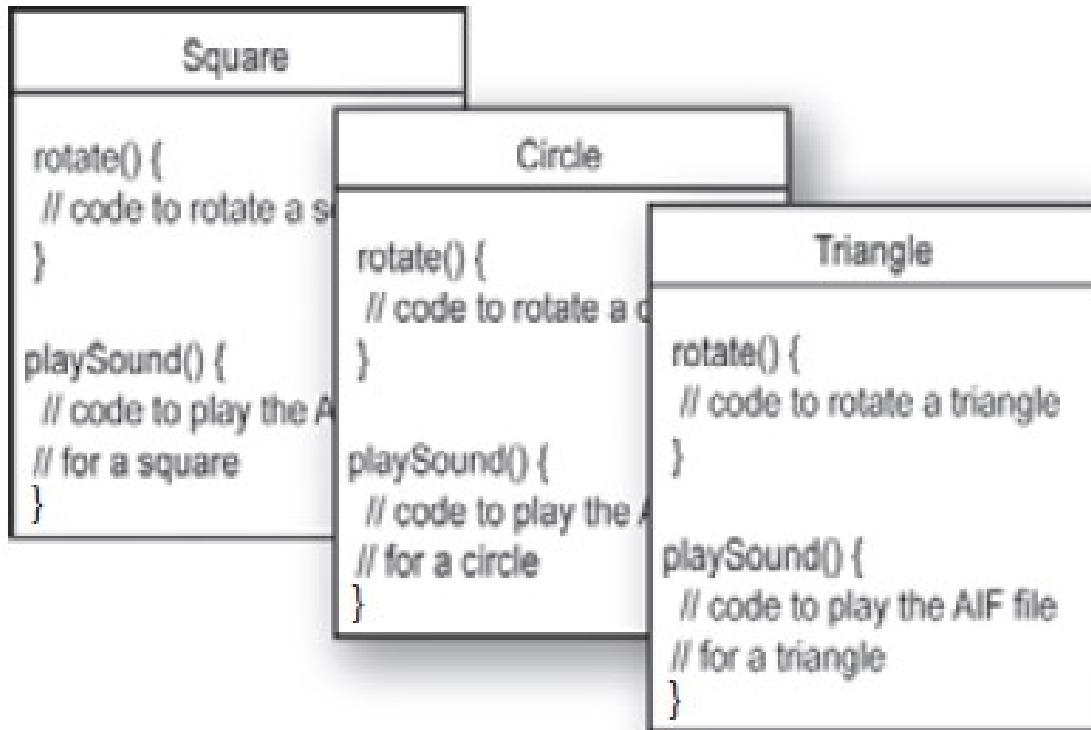
La especificación del problema



Tendremos formas: un cuadrado, una esfera y un triángulo. Cuando el usuario pulse en una forma, la forma debe rotar 360° en sentido horario y debe sonar un sonido característico de esa forma en concreto.



Crearemos una clase para cada *phieta*

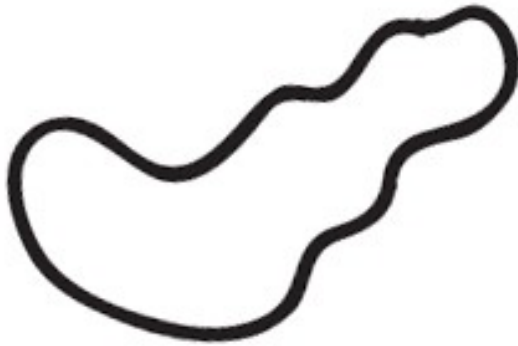


- ❏ Cada clase tiene dos **métodos**:
 - ❏ Uno para hacer rotar a la forma: **rotate()**
 - ❏ Otro para hacer sonar la melodía de cada forma: **playSound()**

Añadir un nuevo objeto no es problema

← Añadimos a la especificación

Tendremos una forma nueva: una ameba.
Cuando el usuario pulse en la ameba, la
forma rotará como las otras formas y hará
sonar un sonido específico para la ameba.



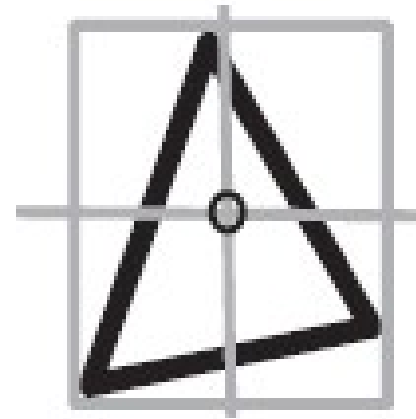
Amoeba

```
rotate() {  
    // code to rotate an amoeba  
}  
  
playSound() {  
    // code to play the new  
    // .hif file for an amoeba  
}
```

Cómo calcular la forma de rotar

🖥️ Pasos para **rotar** una forma:

- 📏 Determinar el rectángulo que rodea la forma
- 📏 Calcular el centro del rectángulo y rotar la figura sobre dicho punto



Pero la ameba no rota igual...



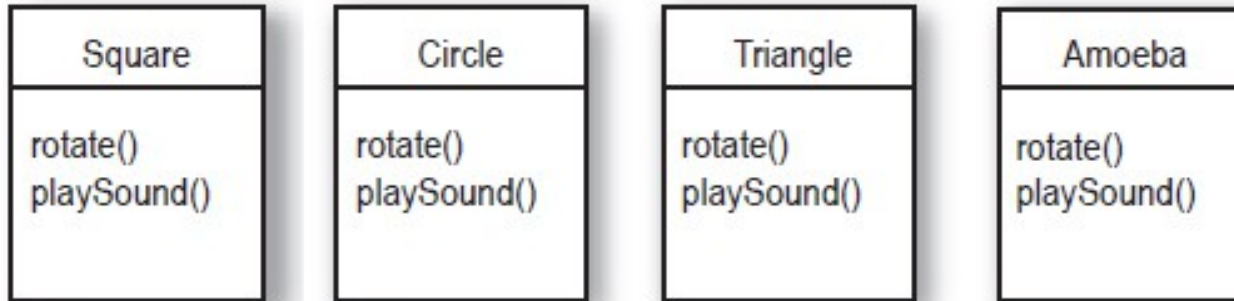
← Lo que la especificación olvidó mencionar

Amoeba
<pre>int xPoint int yPoint rotate() { // code to rotate an amoeba // using amoeba's x and y } playSound() { // code to play the new // .hif file for an amoeba }</pre>

❏ No servirá el mismo método para rotar para la ameba, habrá que especificar las coordenadas

Clases y características comunes

1



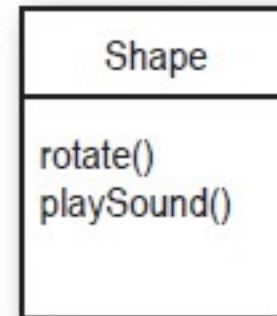
Tenemos 4 clases.

¿Qué tienen en común entre ellas?



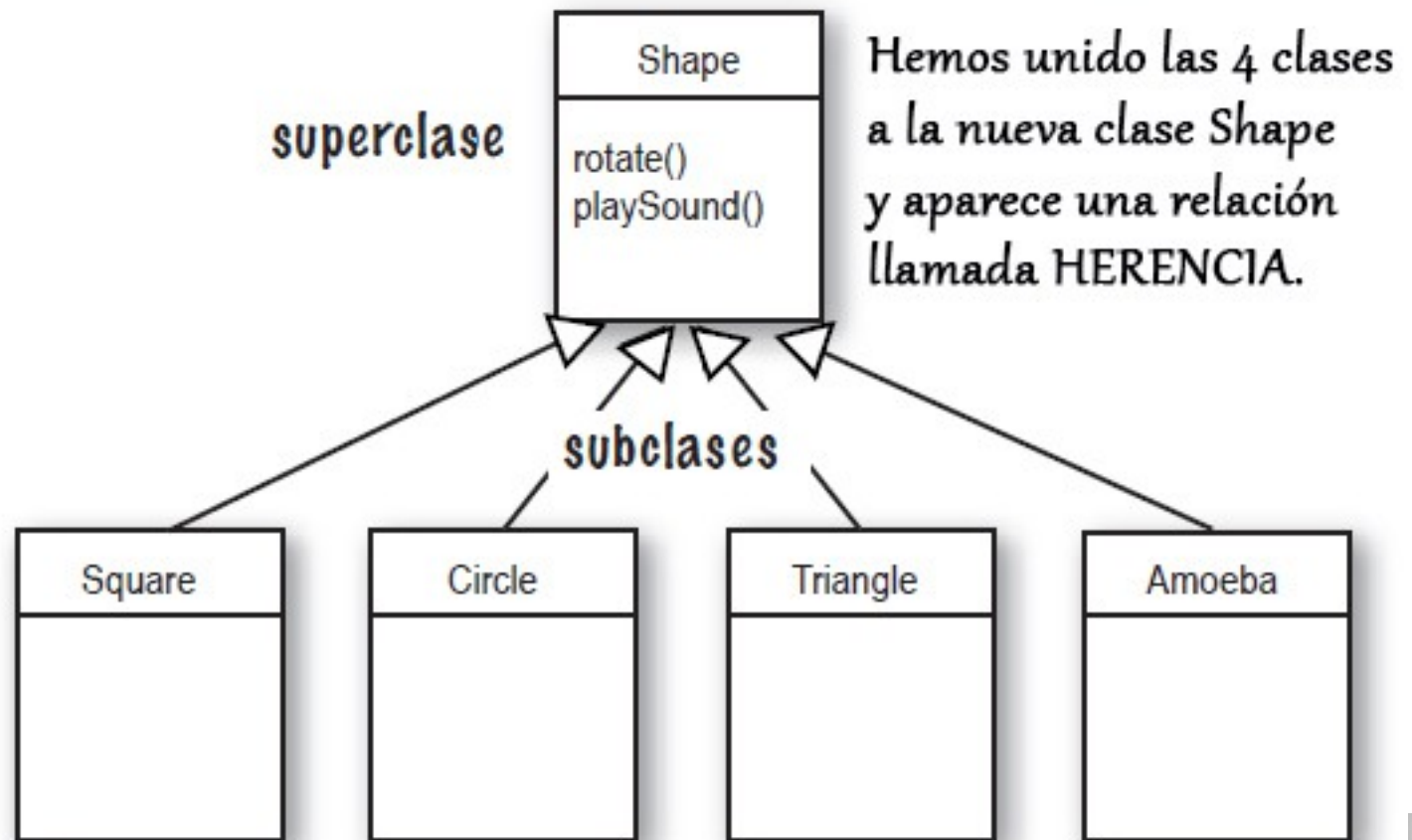
2

Todas ellas son formas, y tienen que rotar y hacer sonar una melodía. Por eso, resumimos las características comunes y las ponemos en una nueva clase llamada 'shape' (forma).



Superclases y subclases

3

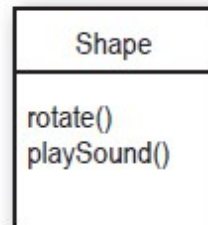


Las 4 subclases heredan de la superclase **Shape** y hereda sus métodos.

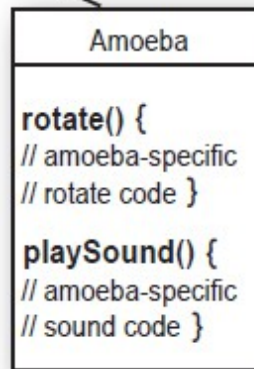
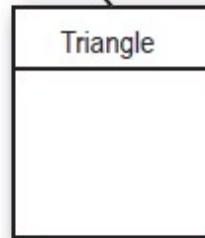
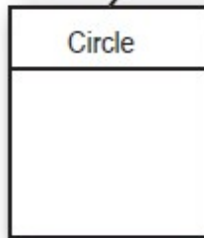
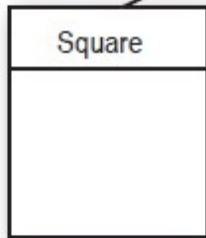
Sobreescribir métodos

4

superclase
(más abstracto)



subclases
(más específico)



En la clase Amoeba se sobreescriben los métodos `rotate()` y `playSound()` de la clase Shape.

Sobreescribir significa que una clase redefina uno de los métodos heredados cuando necesita cambiar o extender el comportamiento de dicho método.



Sobreescribir métodos
(override)

¿Qué te gusta de la POO?

- ✓ **Hablan los expertos:** (programadores y directores de proyecto)
 - ✓ Me ayuda a diseñar de una forma más natural
 - ✓ No perdemos el tiempo con código ya escrito, se añade una nueva característica
 - ✓ Cuando escribo una nueva clase, puedo hacerla flexible para utilizarla de nuevo, en otro momento
 - ✓ Se reduce al máximo, la escritura del nuevo código

Programación orientada a objetos

🖥️ Cuando diseñas una **clase**, piensa en los objetos de la clase que se crearán:

🖥️ Lo que el objeto **sabe**

🖥️ Lo que el objeto **hace**

ShoppingCart
cartContents
addToCart() removeFromCart() checkout()

sabe

hace

Button
label color
setColor() setLabel() dePress() unDepress()

sabe

hace

Alarm
alarmTime alarmMode
setAlarmTime() getAlarmTime() isAlarmSet() snooze()

sabe

hace

Programación orientada a objetos

Lo que el objeto sabe → ***variables de instancia o propiedades o atributos***

Lo que el objeto hace → ***métodos***

variables
de instancia
(estado)

métodos
(comportamiento)




sabe

hace

UML y diagrama de clases

UML: lenguaje de modelado de datos

 **Diagrama de clases:** es un esquema de UML, que utilizaremos para representar las clases, sus variables de instancia y sus métodos

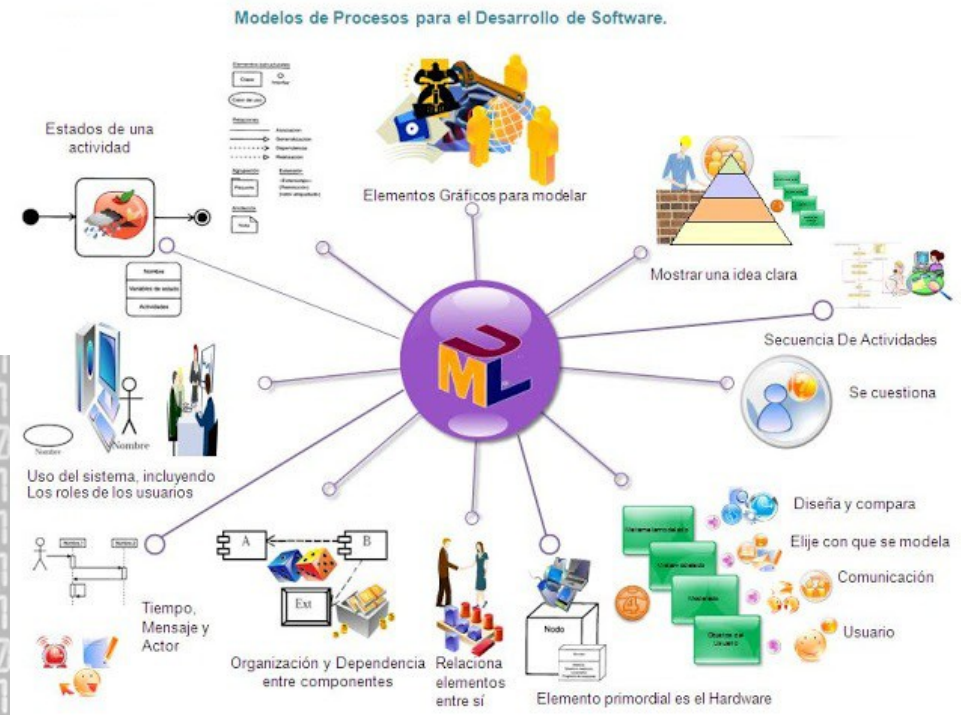
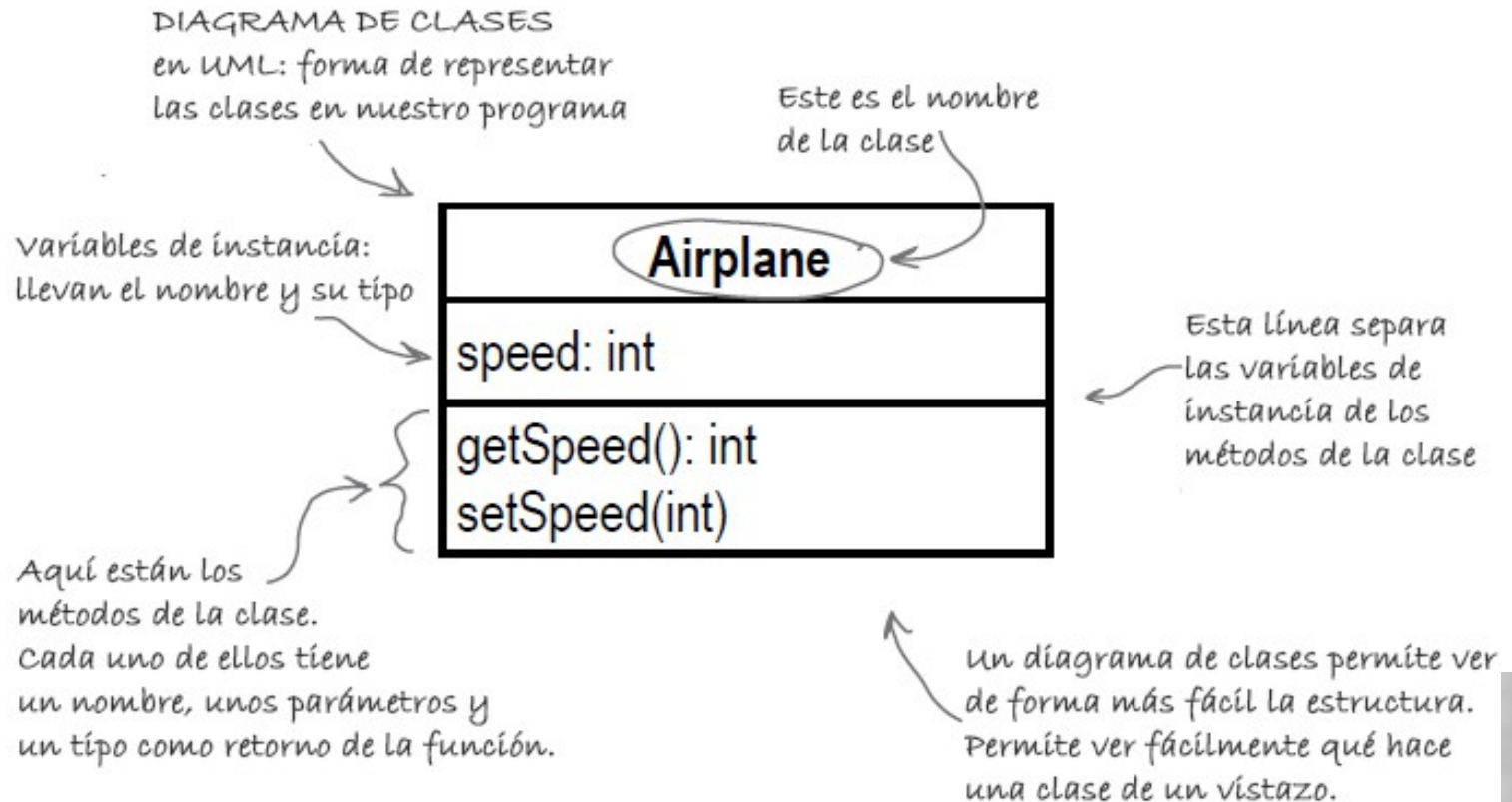


Diagrama de clases en UML



Diagramas de clases con DIA



Dia Diagram Editor

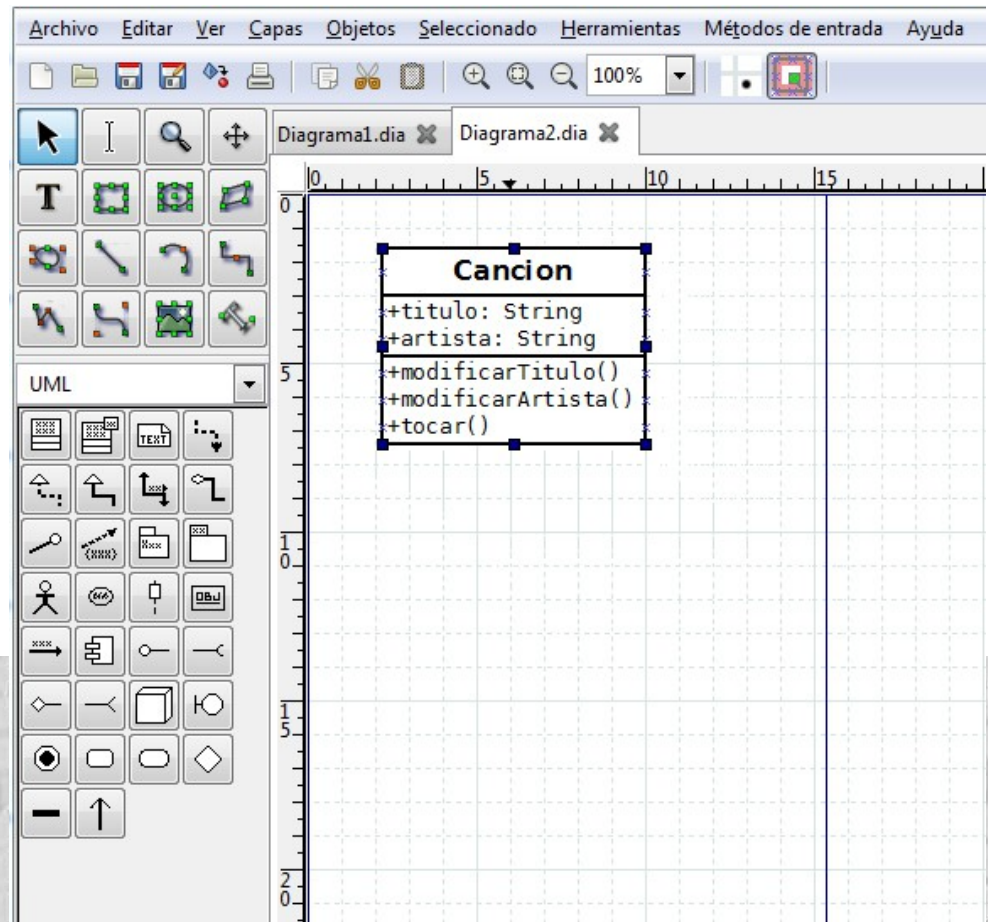
Esquema UML

Elemento clase

Propiedades
(doble clic sobre
la clase):

Atributos

Operaciones



Programación orientada a objetos

🖥️ ***Variables de instancia:*** son los datos, que pueden tomar valores únicos para cada objeto de cada tipo (variables o atributos)

🖥️ ***Instancia:*** es otra forma de decir objeto

🖥️ ***Métodos:*** operaciones a realizar con los datos. Para poder leer y escribir los datos (variables de instancia)

🖥️ Observa las clases anteriores y fíjate en sus variables de instancia y sus métodos.

Práctica P2.0: Clase television



¡Afila tu lápiz!

Rellena en el objeto Television lo que debería saber y debería hacer:



Television

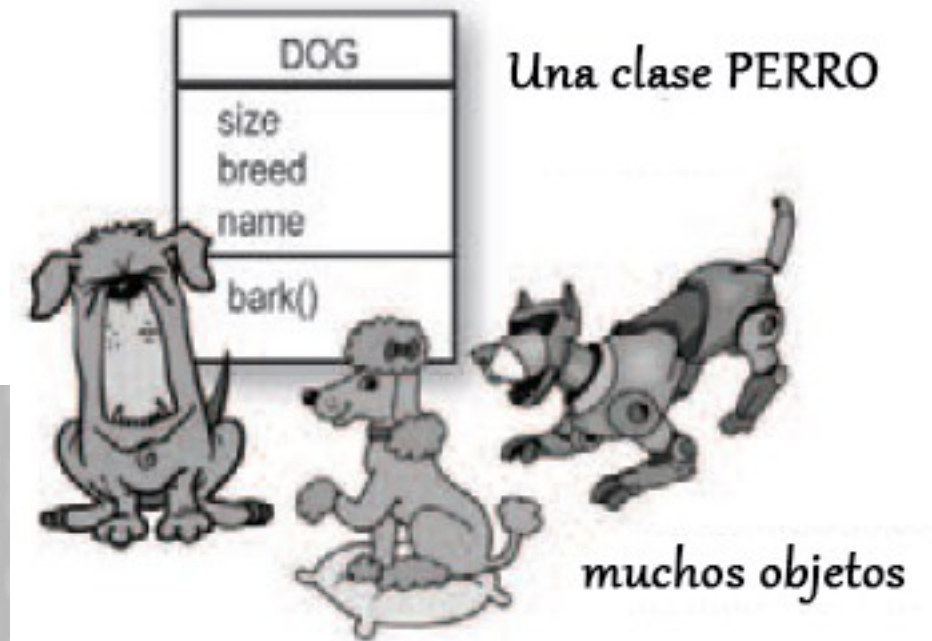
Variables de instancia

Métodos

Diferencia entre clase y objeto

❏ **Una clase no es un objeto...**

❏ ... pero la clase puede ser utilizada **para construir objetos**



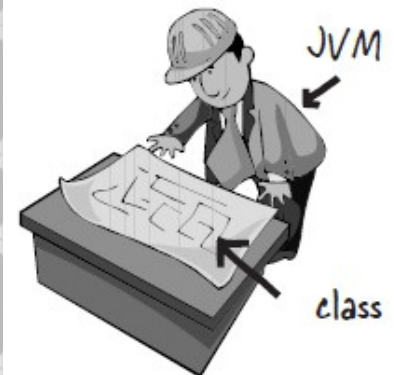
Diferencia entre clase y objeto

📁 ***Una clase es un molde para el objeto***

📁 Indica al JVM cómo crear el objeto de un tipo

📁 Cada objeto se crea desde una clase y tiene sus propios valores para las variables de instancia de esa clase

📁 ***Ejemplo:*** la clase Button puede tener docenas de diferentes objetos Button, con su propio color, tamaño, forma y etiqueta.



Diferencia entre *clase* y *objeto*

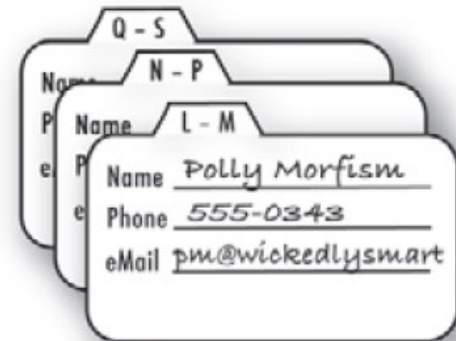
Un objeto es como una entrada en tu **agenda**:

Cada ficha tiene los mismos espacios en blanco (***variables de instancia o atributos***)

Al rellenar una ficha nueva, creamos un **objeto**

Los **métodos** son cosas que podemos hacer con los objetos (***obtenerNombre()***, ***escribirNombre()***)

Cada ficha puede hacer las mismas cosas pero con datos diferentes



El operador punto (.)

El operador punto (.) da acceso al estado y comportamiento del objeto (atributos y métodos)

```
// crear un nuevo objeto
```

```
Perro p = new Perro();
```

```
// llamamos a ladrar usando el .
```

```
p.ladrar();
```

```
// modificamos la altura con el .
```

```
p.altura = 40;
```

Nuestro primer objeto

🖥️ ¿Qué se necesita para **crear y usar un objeto**?

Se necesitan dos clases:

🖥️ Una **clase para el objeto**

🖥️ Una **clase test**, para probar el objeto: que contiene el método **main()** que crea el objeto

🖥️ A las clases test, les pondremos el mismo nombre que a las clases de tipo, pero con el sufijo **TestDrive**

🖥️ **IMPORTANTE:** Se ejecutará el archivo del programa java de la clase que contiene el método **main()**

Definición de clase y clase *TestDrive*

📁 Tendremos en los archivos **.java** las dos clases, la definición de clase y su clase ***TestDrive***

📁 Las clases definidas en cada archivo archivo:

📁 ***Ejemplo.java***

📁 ***EjemploTestDrive.java***

📁 Se crearán dos archivos, uno para cada clase:

📁 ***Ejemplo.class***

📁 ***EjemploTestDrive.class*** (se ejecutará éste: ***main()***)

Pasos para crear una clase

1 Escribe tu clase

```
class Dog {  
  
    int size;  
    String breed;  
    String name;  
  
    void bark() {  
        System.out.println("Ruff! Ruff!");  
    }  
}
```

variables de
instancia

un método

DOG
size breed name
bark()

Pasos para crear una clase

2 Escribe una clase test (TestDrive)

un método público
(más adelante se
escribirá el código)

```
class DogTestDrive {  
    public static void main (String[] args) {  
        // el código de la clase test irá aquí  
    }  
}
```

3 En la clase test, crear el objeto y acceder a sus variables y métodos

```
class DogTestDrive {  
    public static void main (String[] args) {
```

```
        Dog d = new Dog();
```

← crear el objeto Dog

```
        d.size = 40;
```

← usar el operador punto (.)
para modificar la variable size

```
        d.bark();
```

← y llamar al método Bark()

operador
punto (.)

```
    }  
}
```

Práctica P2.1: Creando películas

```
class pelicula {
    String titulo;
    String genero;
    int calificacion;

    void proyectar() {
        System.out.println("Proyectando la película");
    }
}

public class peliculaTesDrive {
    public static void main (String[] args) {
        pelicula uno = new pelicula();
        uno.titulo="Blancanieves y los siete enanitos";
        uno.genero="infantil";
        uno.calificacion=-2;
        pelicula dos = new pelicula();
        dos.titulo="No habrá paz para los malvados";
        dos.genero="Thriller";
        dos.calificacion=127;
        dos.proyectar();
        pelicula tres = new pelicula();
        tres.titulo="Bailando bajo la lluvia";
        tres.genero="musical";
        tres.calificacion=5;
    }
}
```

Práctica P2.1: Rellena los valores

PELICULA
titulo
genero
calificacion
proyectar()



¡Afila tu lápiz!

objeto 1

titulo
genero
calificacion

objeto 2

titulo
genero
calificacion

objeto 3

titulo
genero
calificacion

Puntos importantes (II)

- ❑ La **POO** permite extender un programa sin tener que tocar el código ya testeado y que funcione.
- ❑ Todo en Java se define en una **clase**
- ❑ Una clase describe cómo crear un objeto del tipo de la clase. La clase es como un **molde**.
- ❑ Un objeto **sabe** cosas y **hace** cosas
- ❑ Las cosas que un objeto sabe: **variables de instancia o atributos**. Representan el **estado** del objeto.
- ❑ Las cosas que hace un objeto: **métodos**. Representan el **comportamiento** del objeto.
- ❑ En ejecución, un programa en Java no es más que los **objetos hablan a otros objetos**

Práctica P2.2: Sé el compilador



De cada uno de los siguientes programas, comprueba cuál de ellos compila, ¿Podrías identificar cuál compila y cuáles no y encontrar los errores?

```
A
class TapeDeck {

    boolean canRecord = false;

    void playTape() {
        System.out.println("tape playing");
    }

    void recordTape() {
        System.out.println("tape recording");
    }
}

class TapeDeckTestDrive {
    public static void main(String [] args) {

        t.canRecord = true;
        t.playTape();

        if (t.canRecord == true) {
            t.recordTape();
        }
    }
}
```

```
B
class DVDPlayer {

    boolean canRecord = false;

    void recordDVD() {
        System.out.println("DVD recording");
    }
}

class DVDPlayerTestDrive {
    public static void main(String [] args) {

        DVDPlayer d = new DVDPlayer();
        d.canRecord = true;
        d.playDVD();

        if (d.canRecord == true) {
            d.recordDVD();
        }
    }
}
```



Intenta resolver el ejercicio observando el código, y después comprueba tus resultados con el compilador de Java. Crea el diagrama de clases UML, con el DIA.

Práctica P2.3: Ordena el código

 Ordena las sentencias del siguiente programa para obtener esta salida:

```
d.playSnare();  
DrumKit d = new DrumKit();  
boolean topHat = true;  
boolean snare = true;  
  
void playSnare() {  
    System.out.println("bang bang ba-bang");  
}  
  
public static void main(String [] args) {
```

```
    if (d.snare == true) {  
        d.playSnare();  
    }
```

```
    d.snare = false;
```

```
    class DrumKitTestDrive {
```


```
        d.playTopHat();
```

```
        class DrumKit {
```


```
            void playTopHat () {  
                System.out.println("ding ding da-ding");  
            }  
        }  
    }  
}
```

File Edit Window Help Dance

```
% java DrumKitTestDrive  
bang bang ba-bang  
ding ding da-ding
```

 Crea el diagrama de clases con el DIA, y prueba el programa para que funcione como se espera.

Práctica P2.4: Puzzle en la piscina

 Escoge los fragmentos de código de la piscina y colócalos en los espacios en blanco, de forma que compile y la ejecución produzca la salida esperada.

```
public class EchoTestDrive {
    public static void main(String [] args) {
        Echo e1 = new Echo();

        _____

        int x = 0;
        while ( _____ ) {
            e1.hello();

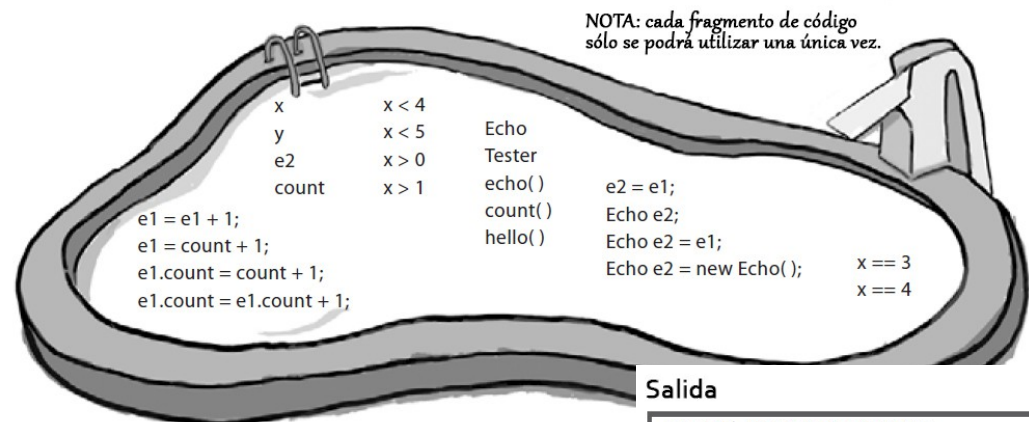
            _____

            if ( _____ ) {
                e2.count = e2.count + 1;
            }

            if ( _____ ) {
                e2.count = e2.count + e1.count;
            }

            x = x + 1;
        }
        System.out.println(e2.count);
    }
}
```

```
class _____ {
    int _____ = 0;
    void _____ {
        System.out.println("helloooo... ");
    }
}
```











Salida

```
File Edit Window Help Implode
%java EchoTestDrive
helloooo...
helloooo...
helloooo...
helloooo...
10
```

 Crea también el diagrama de clases.

Práctica P2.4: Puzzle en la piscina

Consideraciones a tener en cuenta:

-  Esta actividad es más complicada de lo que parece, no te desanimes
-  Cada fragmento de solución sólo podrá ser usado una vez
-  No será necesario utilizar todos los fragmentos
-  Cada línea representa una línea de código
-  Las líneas largas son para instrucciones largas
-  Las líneas cortas para instrucciones cortas
-  Las condiciones van dentro de bucles o tests condicionales
-  Las sentencias van sueltas