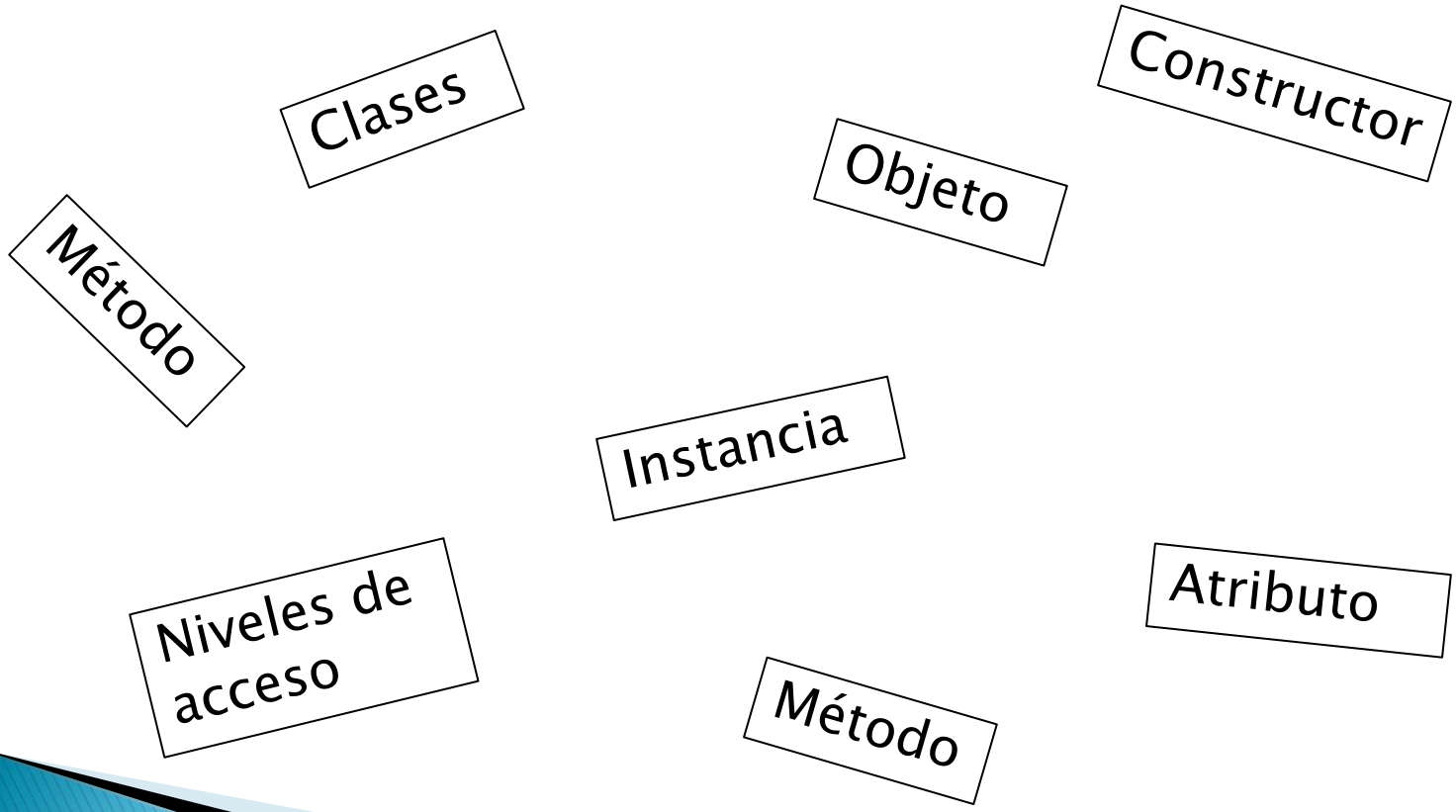
# Algoritmos – TUPAR

Clases – Herencia

# Repasemos…

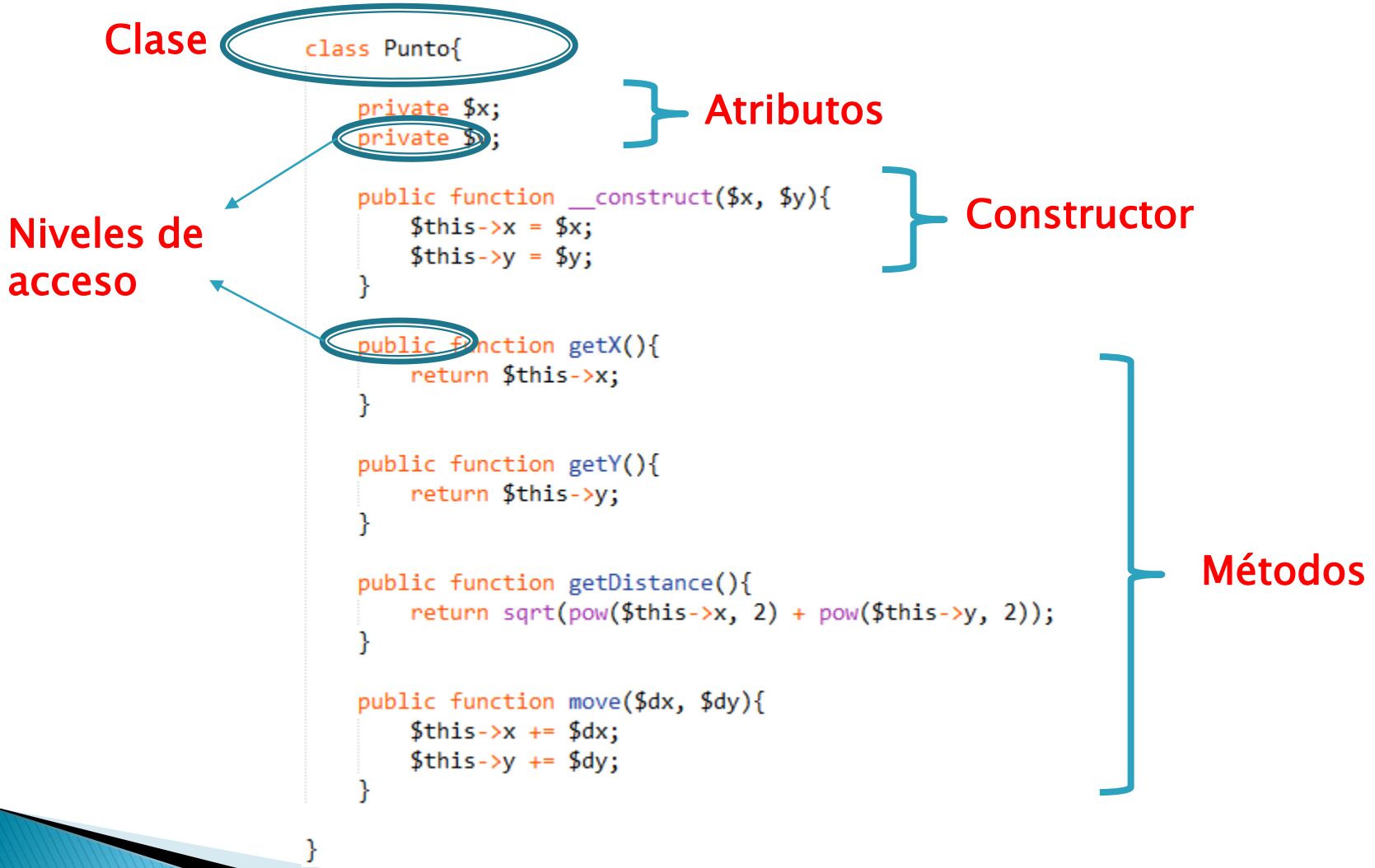
**Algunos conceptos:**



# Repasemos...

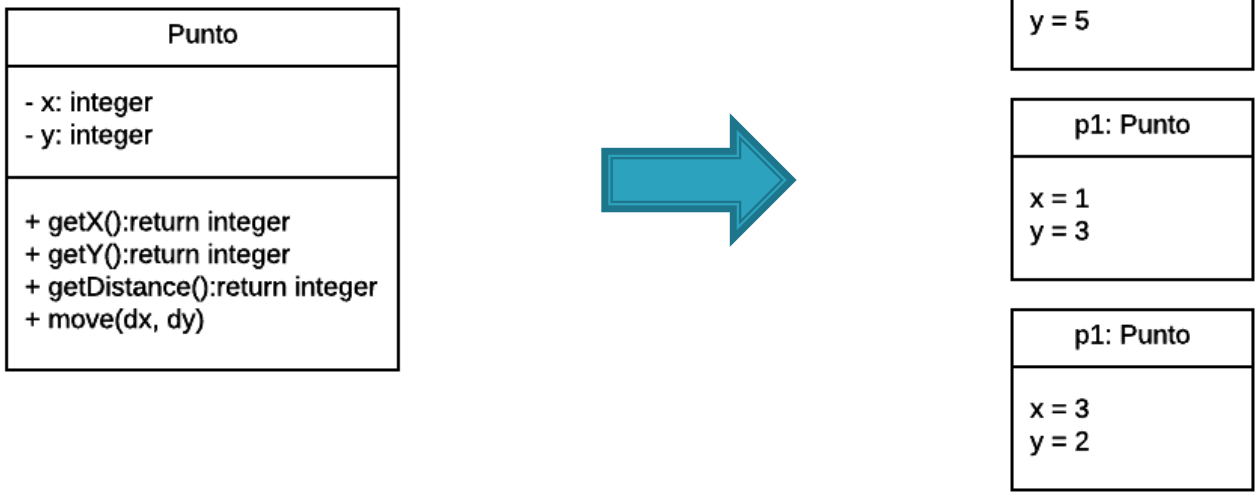
```php
class Punto{

    private $x;
    private $y;

    public function __construct($x, $y){
        $this->x = $x;
        $this->y = $y;
    }

    public function getX(){
        return $this->x;
    }

    public function getY(){
        return $this->y;
    }

    public function getDistance(){
        return sqrt(pow($this->x, 2) + pow($this->y, 2));
    }

    public function move($dx, $dy){
        $this->x += $dx;
        $this->y += $dy;
    }

}
```

# Repasemos...


Clase
class Punto{
private $x;
private $y;
Atributos
Niveles de acceso
public function __construct($x, $y){
$this->x = $x;
$this->y = $y;
}
Constructor
public function getX(){
return $this->x;
}
public function getY(){
return $this->y;
}
public function getDistance(){
return sqrt(pow($this->x, 2) + pow($this->y, 2));
}
public function move($dx, $dy){
$this->x += $dx;
$this->y += $dy;
}
Métodos
}

# Repasemos…

## Clase vs Instancia:




**Clases:** representan entidades o conceptos. Define un conjunto de variables (estado), y métodos apropiados para operarlas (comportamiento)

**Objetos:** instancias de las clases definidas con un estado cierto. Representada en una porción en memoria en ejecución.

# Relaciones

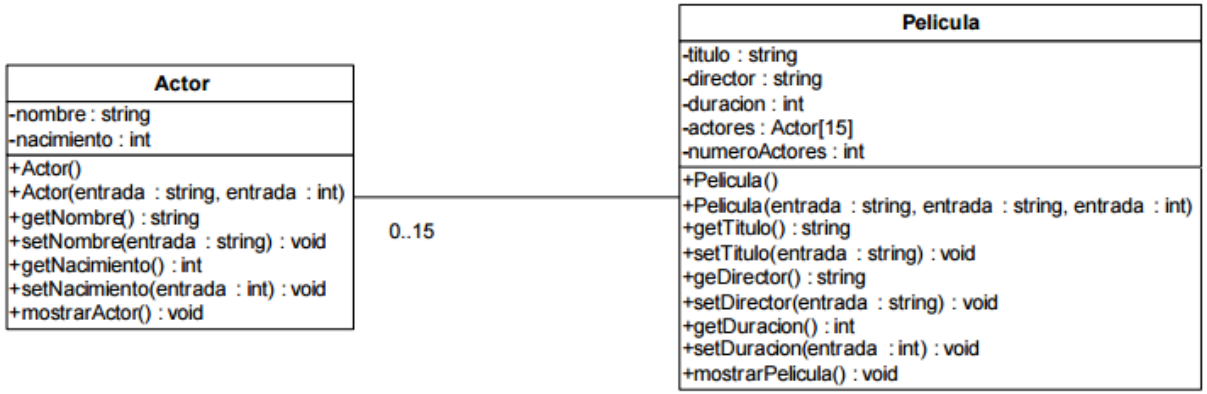
Cualquier vínculo que se dá entre objetos para modelar una representación o aplicación de la realidad.

- **Asociación**: Especifica una relación semántica entre objetos. Dos o más clases tienen una relacion de asociacion cuando una de ellas tenga que requerir o utilizar los servicios que provee la otra.

- **Agregación/composición**: Relación que se basa en la idea de que un objeto está formado estructuralmente por otros.

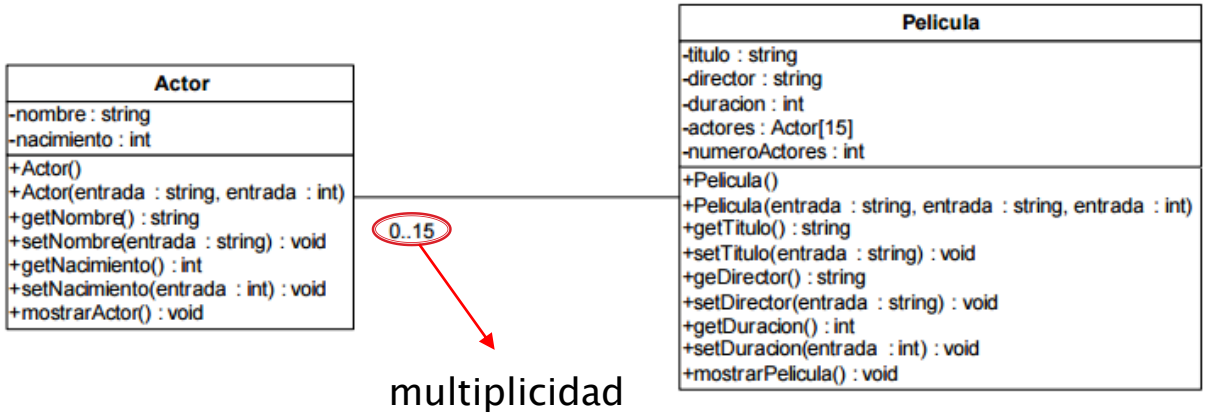
# Relaciones

## Asociación



# Relaciones

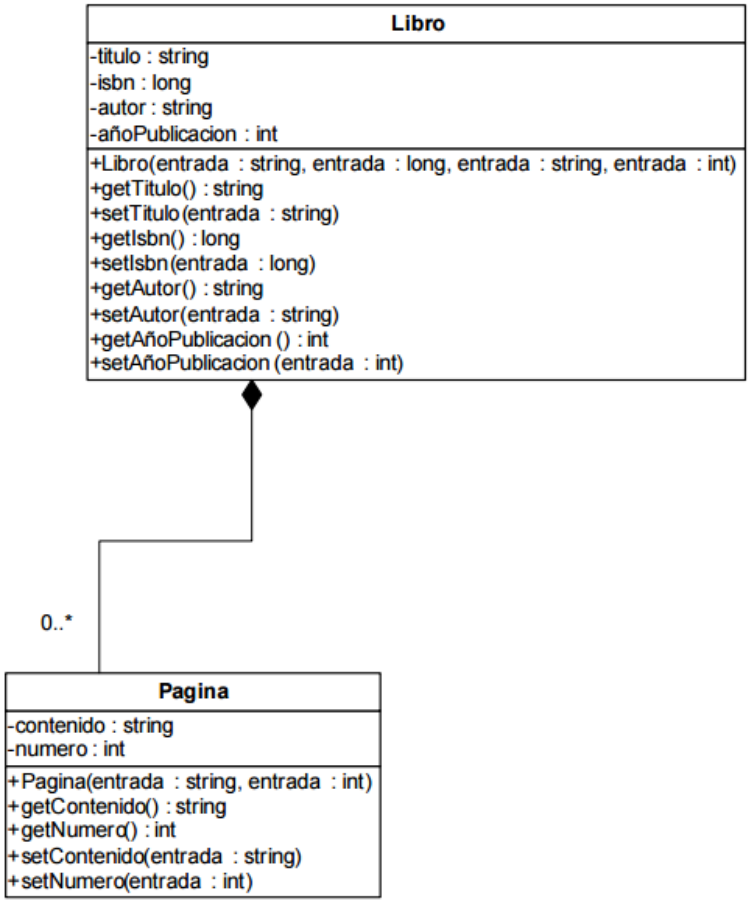
## Asociación




Una pelicula **tiene** actores

# Relaciones

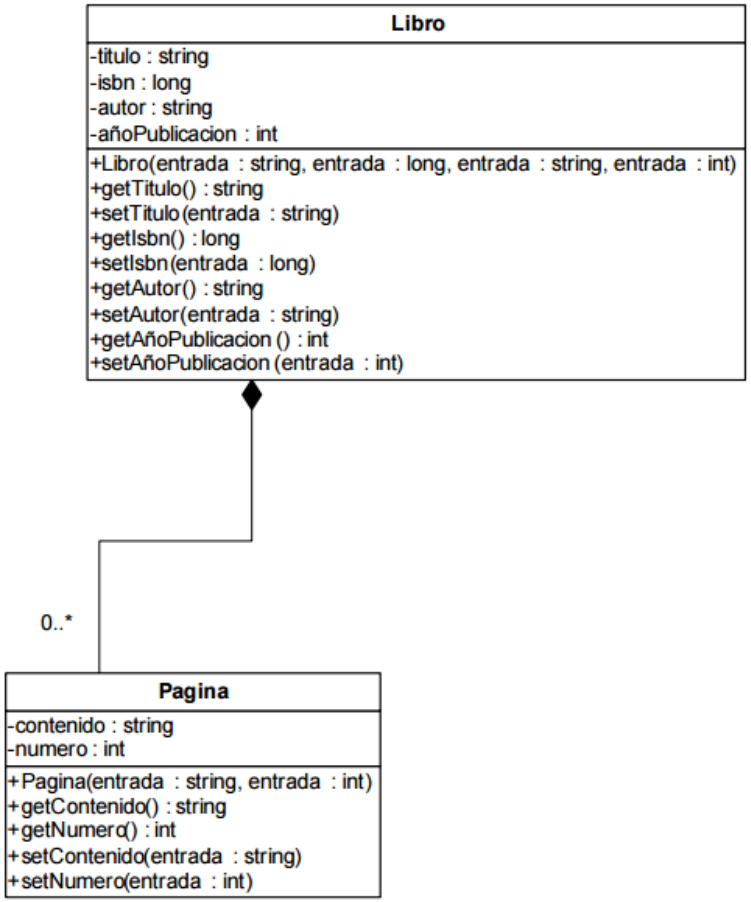
## Agregación/composición



# Relaciones

## Agregación/composición

Un libro **está formado** por páginas

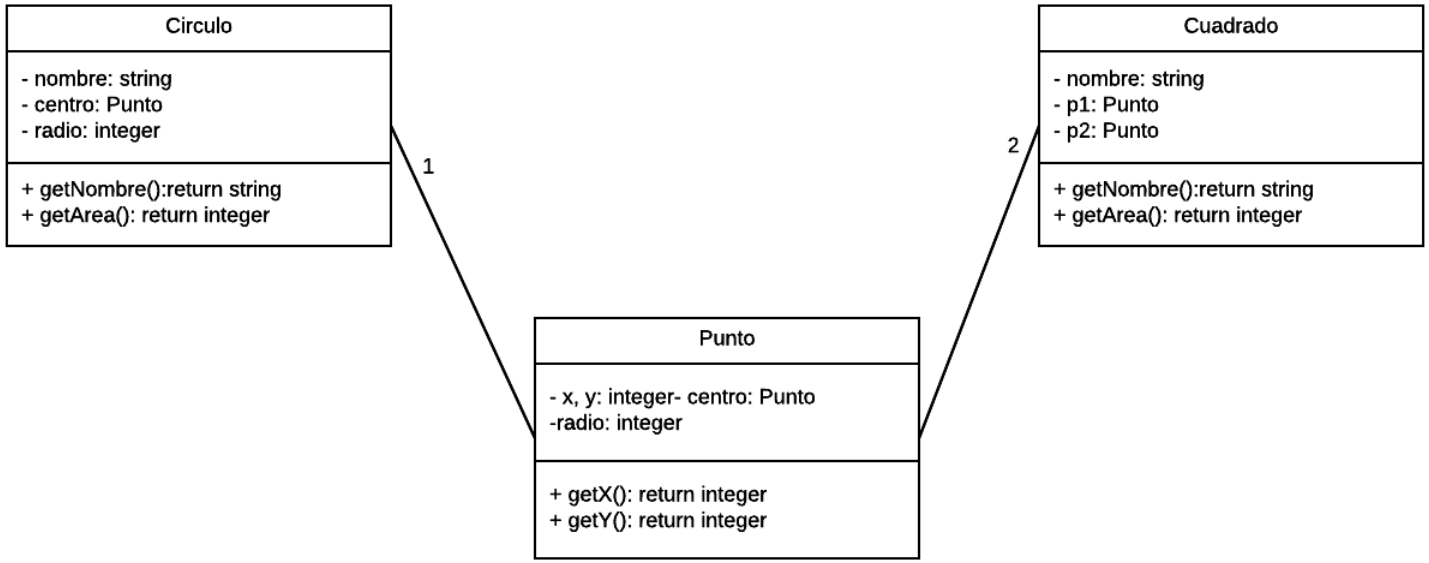


# Herencia

- Una relación de herencia nos permite decir que una clase (hija) obtiene todos los métodos y atributos de otra clase (padre). Luego, la clase hija añade comportamiento propio.

- Permite establecer una jerarquía de clases.

# Herencia


Circulo
- nombre: string
- centro: Punto
- radio: integer
+ getNombre():return string
+ getArea(): return integer
1
Punto
- x, y: integer- centro: Punto
-radio: integer
+ getX(): return integer
+ getY(): return integer
2
Cuadrado
- nombre: string
- p1: Punto
- p2: Punto
+ getNombre():return string
+ getArea(): return integer

# Herencia


Circulo
- nombre: string
- centro: Punto
- radio: integer
+ getNombre():return string
+ getArea(): return integer
Cuadrado
- nombre: string
- p1: Punto
- p2: Punto
+ getNombre():return string
+ getArea(): return integer

# Herencia


Circulo
- nombre: string
- centro: Punto
- radio: integer
+ getNombre():return string
+ getArea(): return integer
Cuadrado
- nombre: string
- p1: Punto
- p2: Punto
+ getNombre():return string
+ getArea(): return integer

# Herencia

```php
class Circulo{
    private $nombre;
    private $centro;
    private $radio;

    public function __construct($nombre, Punto $centro, $radio){
        $this->nombre = $nombre;
        $this->centro = $centro;
        $this->radio = $radio;
    }

    public function getNombre(){
        return $this->nombre;
    }

    public function getArea(){
        return pi()*pow($this->radio, 2);
    }

}
```

```php
class Cuadrado{
    private $nombre;
    private $p1;
    private $p2;

    public function __construct($nombre, Punto $p1, Punto $p2){
        $this->nombre = $nombre;
        $this->p1 = $p1;
        $this->p2 = $p2;
    }

    public function getNombre(){
        return $this->nombre;
    }

    public function getArea(){
        return abs($this->p1->getX() - $this->p2->getX()) *
               abs($this->p1->getY() - $this->p2->getY()) ;
    }

}
```

# Herencia

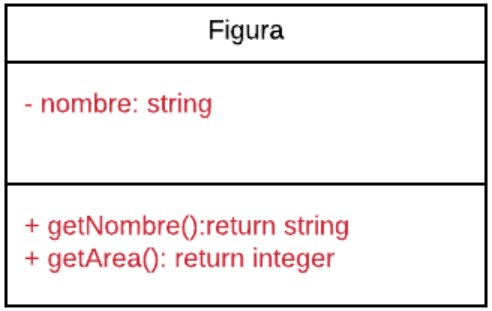
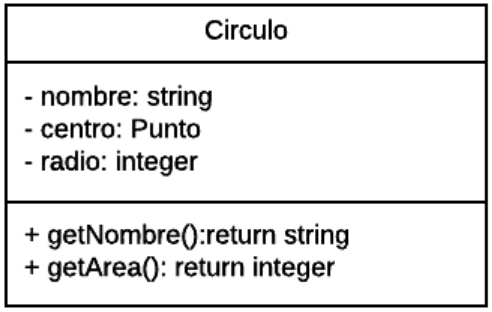
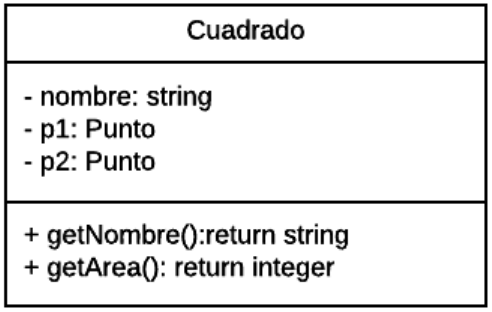


Figura
- nombre: string
+ getNombre():return string
+ getArea(): return integer

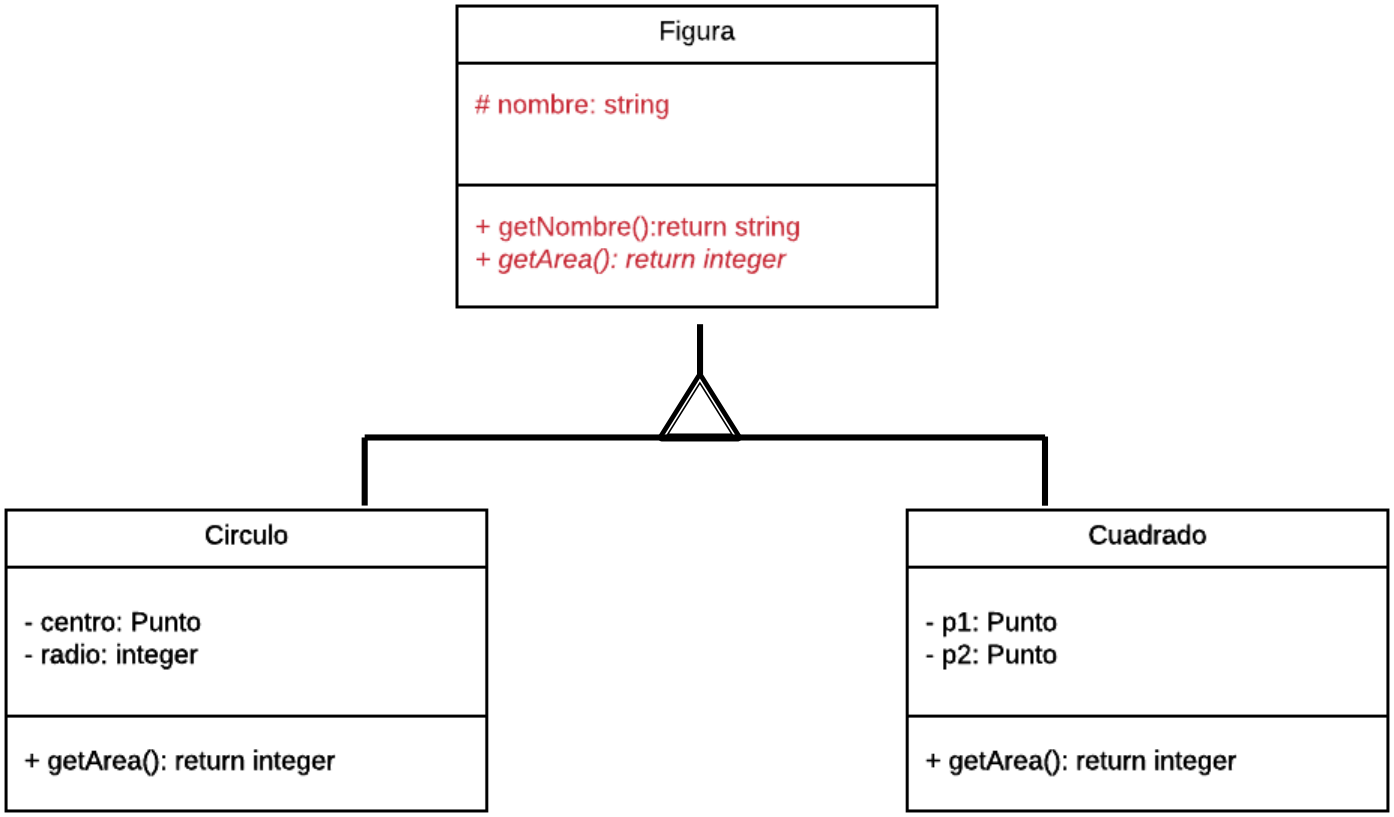

Circulo
- nombre: string
- centro: Punto
- radio: integer
+ getNombre():return string
+ getArea(): return integer


Cuadrado
- nombre: string
- p1: Punto
- p2: Punto
+ getNombre():return string
+ getArea(): return integer

# Herencia

```php
class Figura{

    private $nombre;

    public function __construct($nombre){
        $this->nombre = $nombre;
    }

    public function getNombre(){
        return $this->nombre;
    }

    public function getArea(){
        // ?????
    }

}
```

# Herencia




Un circulo **es una** figura

Un cuadrado **es una** figura

# Herencia

```php
abstract class Figura{

    protected $nombre;

    public function __construct($nombre){
        $this->nombre = $nombre;
    }

    public function getNombre(){
        return $this->nombre;
    }

    public abstract function getArea();

}
```

# Herencia

Si la clase tiene un método abstracto, debo indicar que es abstracta

```php
abstract class Figura{

    protected $nombre;

    public function __construct($nombre){
        $this->nombre = $nombre;
    }

    public function getNombre(){
        return $this->nombre;
    }

    public abstract function getArea();

}
```

# Herencia

```php
class Circulo extends Figura{

    private $centro;
    private $radio;

    public function __construct($nombre, Punto $centro, $radio){
        parent::__construct($nombre);
        $this->centro = $centro;
        $this->radio = $radio;
    }


    public function getArea(){
        return pi()*pow($this->radio, 2);
    }

}
```

```php
class Cuadrado extends Figura{

    private $p1;
    private $p2;

    public function __construct($nombre, Punto $p1, Punto $p2){
        parent::__construct($nombre);
        $this->p1 = $p1;
        $this->p2 = $p2;
    }


    public function getArea(){
        return abs($this->p1->getX() - $this->p2->getX()) *
               abs($this->p1->getY() - $this->p2->getY()) ;
    }

}
```

# Herencia

Indico que hereda de Figura

```php
class Circulo extends Figura{

    private $centro;
    private $radio;

    public function __construct($nombre, Punto $centro, $radio){
        parent::__construct($nombre);
        $this->centro = $centro;
        $this->radio = $radio;
    }


    public function getArea(){
        return pi()*pow($this->radio, 2);
    }

}
```

```php
class Cuadrado extends Figura{

    private $p1;
    private $p2;

    public function __construct($nombre, Punto $p1, Punto $p2){
        parent::__construct($nombre);
        $this->p1 = $p1;
        $this->p2 = $p2;
    }


    public function getArea(){
        return abs($this->p1->getX() - $this->p2->getX()) *
               abs($this->p1->getY() - $this->p2->getY()) ;
    }

}
```

# Herencia

```php
class Circulo extends Figura{

    private $centro;
    private $radio;

    public function __construct($nombre, Punto $centro, $radio){
        parent::__construct($nombre);
        $this->centro = $centro;
        $this->radio = $radio;
    }


    public function getArea(){
        return pi()*pow($this->radio, 2);
    }

}
```

```php
class Cuadrado extends Figura{

    private $p1;
    private $p2;

    public function __construct($nombre, Punto $p1, Punto $p2){
        parent::__construct($nombre);
        $this->p1 = $p1;
        $this->p2 = $p2;
    }


    public function getArea(){
        return abs($this->p1->getX() - $this->p2->getX()) *
               abs($this->p1->getY() - $this->p2->getY()) ;
    }

}
```

Llamado al constructor de la clase padre

# Herencia

```php
class Circulo extends Figura{

    private $centro;
    private $radio;

    public function __construct($nombre, Punto $centro, $radio){
        parent::__construct($nombre);
        $this->centro = $centro;
        $this->radio = $radio;
    }


    public function getArea(){
        return pi()*pow($this->radio, 2);
    }

}
```

```php
class Cuadrado extends Figura{

    private $p1;
    private $p2;

    public function __construct($nombre, Punto $p1, Punto $p2){
        parent::__construct($nombre);
        $this->p1 = $p1;
        $this->p2 = $p2;
    }


    public function getArea(){
        return abs($this->p1->getX() - $this->p2->getX()) *
               abs($this->p1->getY() - $this->p2->getY()) ;
    }

}
```

Las clases hija debe implementar **todos** los métodos abstractos de la clase padre

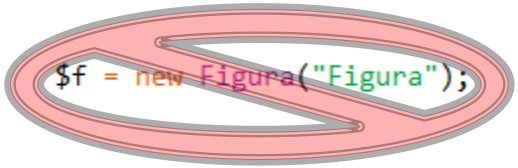
# Herencia

```php
class Circulo extends Figura{

    private $centro;
    private $radio;

    public function __construct($nombre, Punto $centro, $radio){
        parent::__construct($nombre);
        $this->centro = $centro;
        $this->radio = $radio;
    }


    public function getArea(){
        return pi()*pow($this->radio, 2);
    }

}
```

```php
class Cuadrado extends Figura{

    private $p1;
    private $p2;

    public function __construct($nombre, Punto $p1, Punto $p2){
        parent::__construct($nombre);
        $this->p1 = $p1;
        $this->p2 = $p2;
    }


    public function getArea(){
        return abs($this->p1->getX() - $this->p2->getX()) *
               abs($this->p1->getY() - $this->p2->getY()) ;
    }

}
```

# Herencia

- No se puede hacer instancia de un objeto de una clase abstracta.



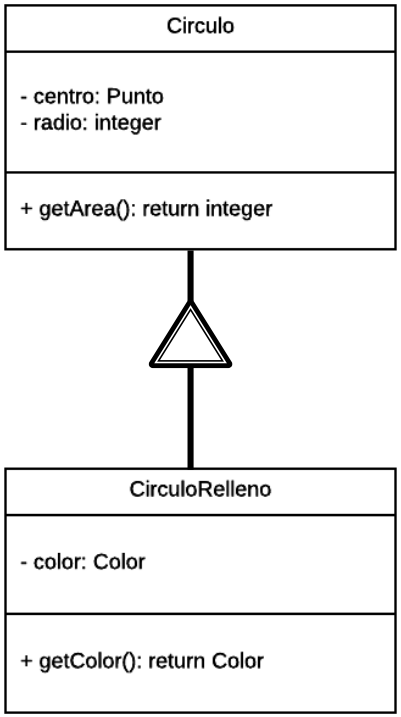

- Pero se puede indicar que un parámeton de una función sea de una clase abstracta.

```
function addFigura(Figura $figura){
    //...
}
```

Puede ser Cuadrado o Circulo (u otra clase que herede de Figura)

# Herencia

- Puedo hacer herencia de una clase concreta.




Defino un tipo especial
de Circulo.
Agrego comportamiento