POO 5

Juan Carlos Figueroa Duran

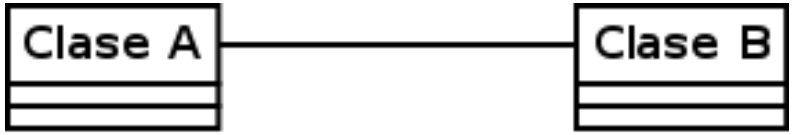
Otoño-2016

Universidad del Bío-Bío
Departamento de Ciencias de la Computación y Tecnología de la Información

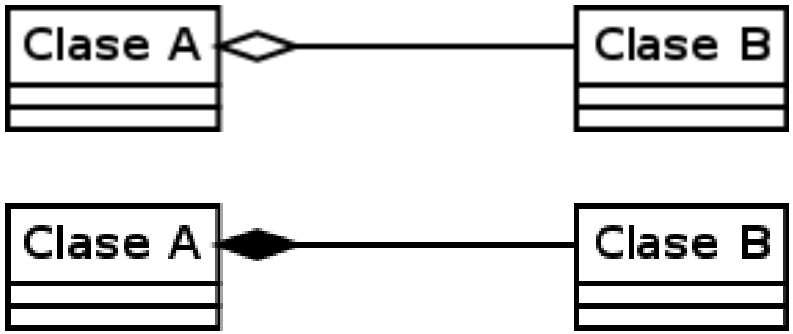
Temario

1. Relaciones
   - Herencia
   - Herencia en Java
   - Polimorfismo
   - Clases Abstractas

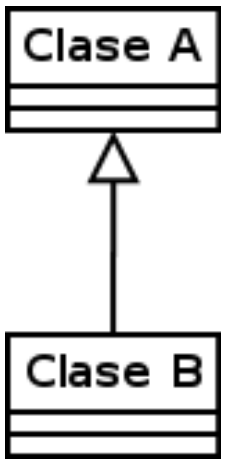
- Asociación



- Agregación y composición

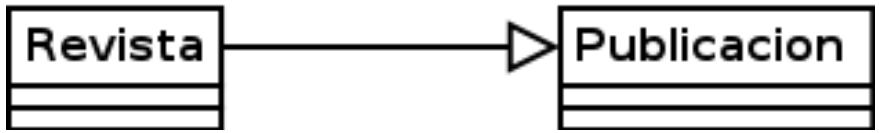


- Herencia



- La herencia relaciona dos objetos indicando que uno se deriva o es una extensión del otro, de tal forma que uno se puede entender como una generalización y el otro como una especialización de un concepto común.
- El verbo asociado a esta relación es:“es un“

Ejemplo

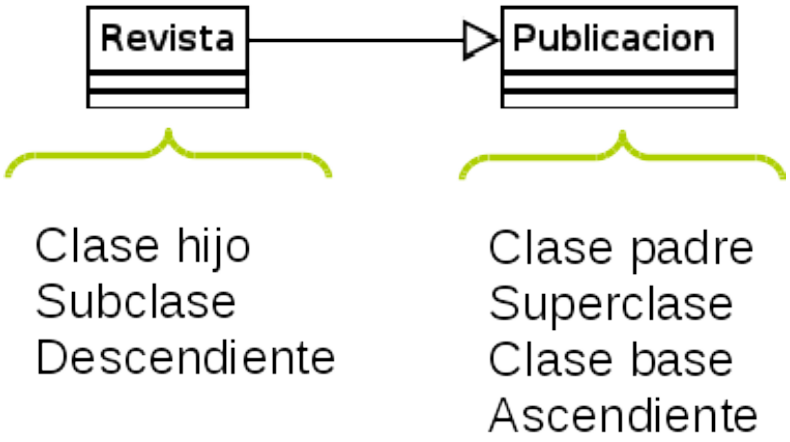


- La clase heredera contendrá automáticamente algunos o todos los atributos y métodos de la clase original.
- Luego es posible agregar nuevos atributos y métodos a la clase nueva o modificarlos para definir de manera apropiada la nueva clase.

Ventajas

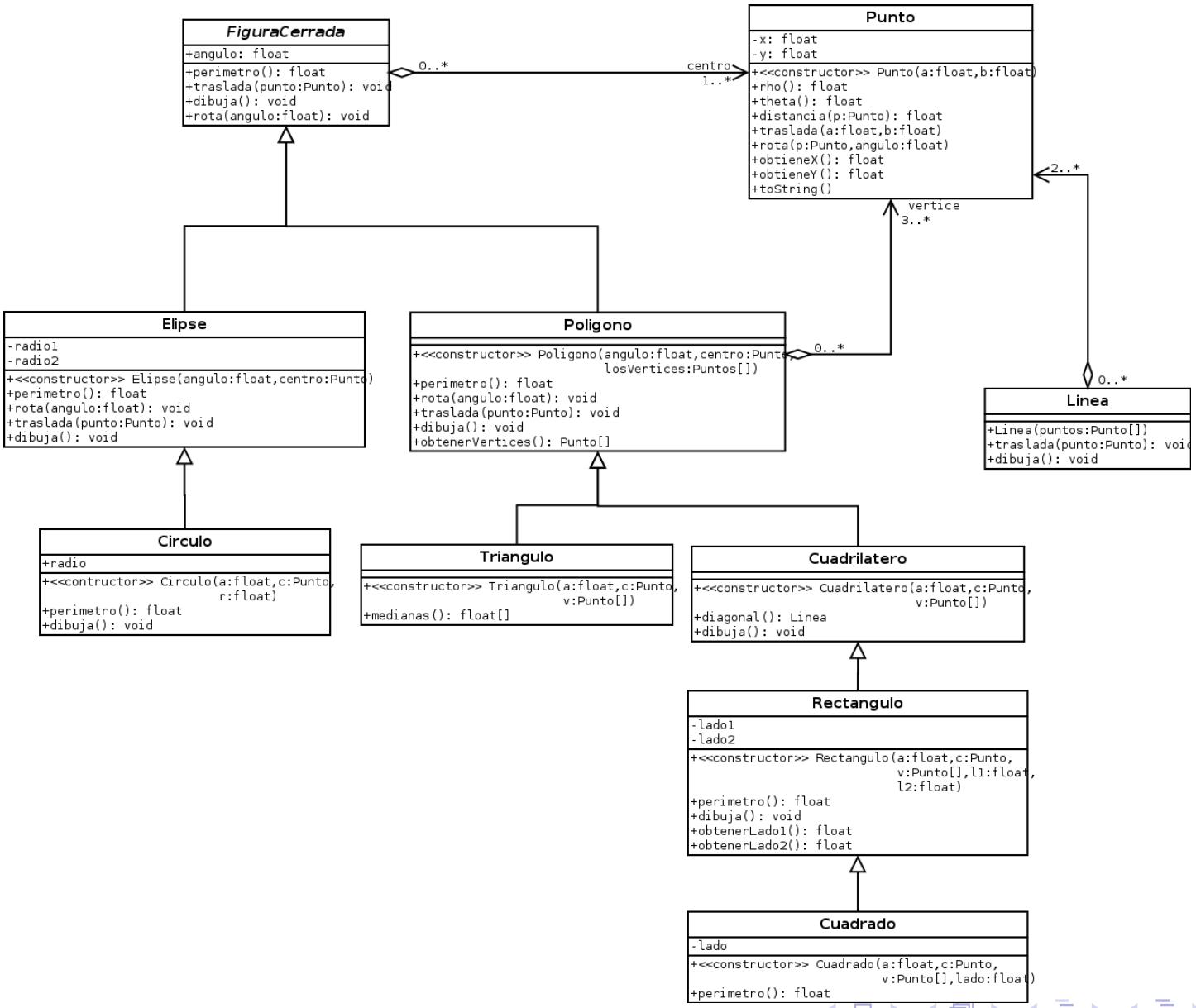
- La construcción de las clases que se crean usando herencia es más rápido y son menos complejas que si tuvieran que incluir todo el código.
- Menos esfuerzo de desarrollo.
- Reutilización del Software.

Notaciones



Todos los objetos de una subclase pertenecen también a la superclase, pero no al revés.

Ejemplo


FiguraCerrada
+angulo: float
+perimetro(): float
+traslada(punto:Punto): void
+dibuja(): void
+rota(angulo:float): void
0..*
centro
1..*
Punto
-x: float
-y: float
+<<constructor>> Punto(a:float,b:float)
+rho(): float
+theta(): float
+distancia(p:Punto): float
+traslada(a:float,b:float)
+rota(p:Punto,angulo:float)
+obtieneX(): float
+obtieneY(): float
+toString()
2..*
vertice
3..*
Elipse
-radio1
-radio2
+<<constructor>> Elipse(angulo:float,centro:Punto)
+perimetro(): float
+rota(angulo:float): void
+traslada(punto:Punto): void
+dibuja(): void
Poligono
+<<constructor>> Poligono(angulo:float,centro:Punto,
losVertices:Puntos[])
+perimetro(): float
+rota(angulo:float): void
+traslada(punto:Punto): void
+dibuja(): void
+obtenerVertices(): Punto[]
0..*
0..*
Linea
+Linea(puntos:Punto[])
+traslada(punto:Punto): voi
+dibuja(): void
Circulo
+radio
+<<contructor>> Circulo(a:float,c:Punto,
r:float)
+perimetro(): float
+dibuja(): void
Triangulo
+<<constructor>> Triangulo(a:float,c:Punto,
v:Punto[])
+medianas(): float[]
Cuadrilatero
+<<constructor>> Cuadrilatero(a:float,c:Punto,
v:Punto[])
+diagonal(): Linea
+dibuja(): void
Rectangulo
-lado1
-lado2
+<<constructor>> Rectangulo(a:float,c:Punto,
v:Punto[],l1:float,
l2:float)
+perimetro(): float
+dibuja(): void
+obtenerLado1(): float
+obtenerLado2(): float
Cuadrado
-lado
+<<constructor>> Cuadrado(a:float,c:Punto,
v:Punto[],lado:float)
+perimetro(): float

En Java, para indicar que una clase se hereda de otra (o es subclase de otra) se utiliza la expresión **extends** al iniciar la definición de la subclase, ejemplo:

```
1 public class RECTANGULO extends CUADRILATERO {
2 // Atributos
3 private float lado1;
4 private float lado2;
5 // Constructor
6 public RECTANGULO(float a, float b) { lado1 = a; lado2 =
7 b; ... }
8 // Metodos
9 public float perimetro() {...}
10 public void dibuja() {...}
```

Accessos

- Las variables y métodos que se heredan viene controlados por los modificadores de visibilidad.
- Los miembros con visibilidad public se heredan.
- Los miembros con visibilidad private no se heradan.
- El acceso protegido significa que el atributo o método podrá ser accedido por la clase donde se define y por todas sus subclases, pero será inaccesible para otras clases:

```
1 protected float angulo;
```

Relaciones entre objetos de subclase y los de superclase

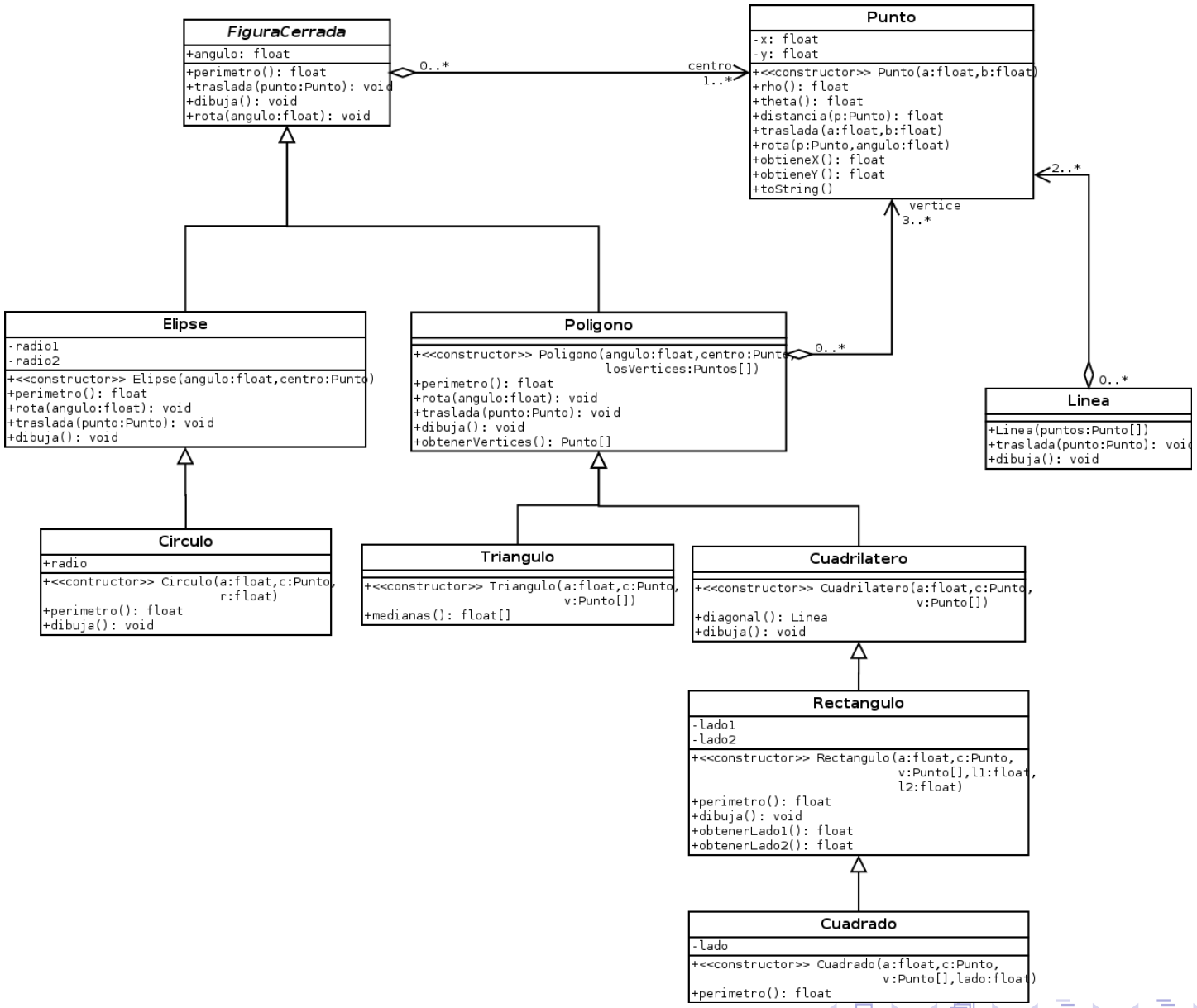
- Un objeto que es una ocurrencia de una superclase no implica que sea un objeto de una subclase particular de esa superclase. Ejemplo, si un objeto d es una ocurrencia de la clase POLIGONO, no implica que dicho objeto sea un CUADRADO, de hecho podría ser una instancia de TRIANGULO.
- Esta propiedad permite tener objetos de distintas subclases en un vector (arreglo, lista, etc.) que se define del tipo de la superclase común y, de este modo, invocar, durante ejecución, los métodos correctos dependiendo del tipo concreto del objeto.

Polimorfismo

Un mismo método en varias clases, en cada una de ellas con una definición distinta

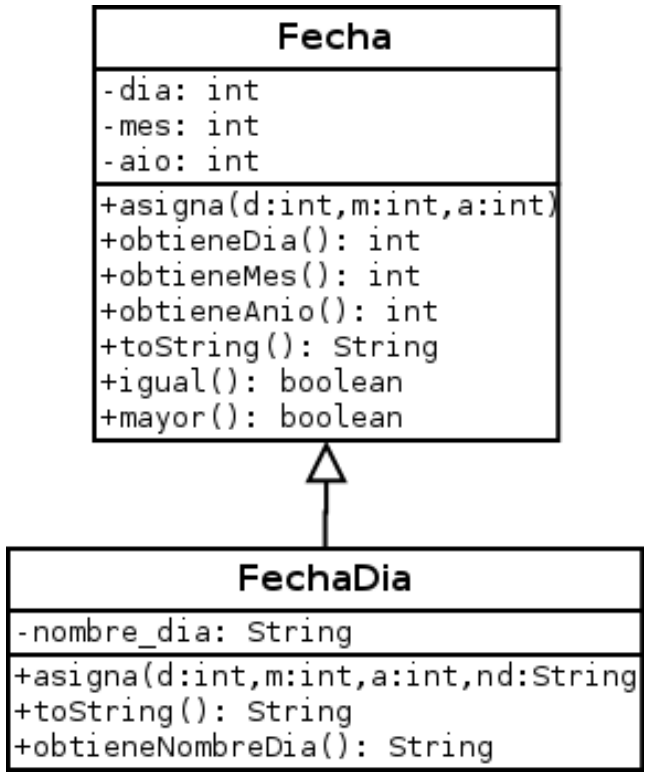
```
1  Public class ....{
2  ...
3 POLIGONO [] figuras = new POLIGONO[10];
4  ...
5 figuras[0] = new CUADRADO(...);
6 figuras[1] = new TRIANGULO(...):
7 figuras[2] = new POLIGONO(...);
8 for (int i=0; i<10; i++)
9
10 System.out.println(figuras[i].perimetros());
11 }
```

Ejemplo


FiguraCerrada
+angulo: float
+perimetro(): float
+traslada(punto:Punto): void
+dibuja(): void
+rota(angulo:float): void
0..*
centro
1..*
Punto
-x: float
-y: float
+<<constructor>> Punto(a:float,b:float)
+rho(): float
+theta(): float
+distancia(p:Punto): float
+traslada(a:float,b:float)
+rota(p:Punto,angulo:float)
+obtieneX(): float
+obtieneY(): float
+toString()
2..*
vertice
3..*
Elipse
-radio1
-radio2
+<<constructor>> Elipse(angulo:float,centro:Punto)
+perimetro(): float
+rota(angulo:float): void
+traslada(punto:Punto): void
+dibuja(): void
Poligono
+<<constructor>> Poligono(angulo:float,centro:Punto,
losVertices:Puntos[])
+perimetro(): float
+rota(angulo:float): void
+traslada(punto:Punto): void
+dibuja(): void
+obtenerVertices(): Punto[]
0..*
0..*
Linea
+Linea(puntos:Punto[])
+traslada(punto:Punto): voi
+dibuja(): void
Circulo
+radio
+<<contructor>> Circulo(a:float,c:Punto,
r:float)
+perimetro(): float
+dibuja(): void
Triangulo
+<<constructor>> Triangulo(a:float,c:Punto,
v:Punto[])
+medianas(): float[]
Cuadrilatero
+<<constructor>> Cuadrilatero(a:float,c:Punto,
v:Punto[])
+diagonal(): Linea
+dibuja(): void
Rectangulo
-lado1
-lado2
+<<constructor>> Rectangulo(a:float,c:Punto,
v:Punto[],l1:float,
l2:float)
+perimetro(): float
+dibuja(): void
+obtenerLado1(): float
+obtenerLado2(): float
Cuadrado
-lado
+<<constructor>> Cuadrado(a:float,c:Punto,
v:Punto[],lado:float)
+perimetro(): float

Ejemplo

```
1  public class FechaDia extends Fecha {
2  // Atributos
3  private String nombre_dia;
4  // Constructores
5  public FechaDia(int d, int m, int a,
       String nd) {
6  // Precondicion: nd<>null, d>=1, d<=31
7  super(d,m,a);
8  nombre_dia = nd;
9  }
10 public void asigna(int d, int m, int a,
       String nd)
11 {
12 // Precondicion: nd<>null, d>=1, d<=31
13 super.asigna(d,m,a);
14 nombre_dia = nd;
15 }
16 public String obtieneNombreDia(){
17 return nombre_dia;
18 }
19 public String toString() {
20 return (nombre_dia + " " +
21 super.toString());
22 }
```



Clases Abstractas

- Una clase es abstracta cuando, siendo una superclase, constituye más bien un concepto creado durante el proceso de análisis del problema y, por lo tanto, supone incorrecto crear objetos de dicha clase.
- Sólo tiene como fin definir una plantilla que cada subclase tomará y concretará, ya sea, agregando atributos, agregando y/o redefiniendo métodos.

Ejemplo

```
public abstract class FIGURA_CERRADA {
// Atributos
protected float angulo;
// Asociaciones
protected PUNTO centro;
// Constructor
public FIGURA_CERRADA(...)
{...}
// Métodos
public float perímetro() {.
return 0;
}
public abstract void rota();
public abstract void traslada();
} /* fin clase FIGURA_CERRADA */
```

Atributos que pueden ser accedidos directamente por las subclases de FIGURA_CERRADA.

Una clase abstracta puede tener constructor, pero no debe ser usado para crear objetos de la clase.

Método concreto que puede ser redefinido en las subclases, cuando corresponda.

Métodos abstractos. Deben definirse, obligatoriamente, en las subclases no abstractas.