HERENCIA
(2ª parte)

Object

- En Java <u>todas</u> las clases derivan de otra, aunque no se diga explícitamente.

> `Object`:
>
> Es el nombre de la clase raíz del árbol de jerarquías de Java.

- Todos los objetos se pueden considerar instancias de `Object`.
- Cualquier método definido en `Object` se puede aplicar a cualquier objeto.
- Dos métodos interesantes: `equals` y `toString`.

Método `equals`

- Esta pensado para comprobar si dos objetos son exactamente iguales.
- Prototipo de `equals` en la clase `Object`:

```
public boolean equals (Object o)
```

- La implementación de `equals` en la clase `Object` sólo comprueba si el receptor y el el argumento hacen referencia al mismo objeto:
  - ✓ Comprueba igualdad de referencias, no de objetos.
- Para poder comparar bien nuestros objetos hace falta incluir en nuestra clase la redefinición correcta de `equals`.

Método `equals`: Ejemplo

```
class Circulo {
    public Circulo () {}
    public void ponerCentro (int x, int y){}
    public void ponerRadio (int r) {}
    public double longitud () {}
    public double area () {}
    public void dibujar () {}

    public boolean equals (Object o) {
        Circulo c = (Circulo) o;
        if ( x_centro != c.x_centro ) return ( false );
        else
            if ( y_centro != c.y_centro ) return ( false );
            else
                if ( radio != c.radio ) return ( false );
                else return ( true );
    }

    private int x_centro;
    private int y_centro;
    private int radio;
    private static final double PI=3.14159265358979323;
}
```

Dos círculos son iguales si están en la misma posición y tienen el mismo tamaño.

Método `toString`

- **Esta pensado para obtener una representación en un String del objeto sobre el que se ha invocado.**
  - ✓ Finalidad típica: Escribir el objeto con
    `System.out.print (obj)`
- **El método `print` invoca a `obj.toString()`, que siempre se hereda de `Object`:**
  - ✓ Dependiendo de la clase a la que pertenezca `obj`, la salida tendrá sentido o no.
- **Prototipo de `toString` en la clase `Object`:**
  `public String toString ()`
- **Para poder comparar bien nuestros objetos hace falta incluir en nuestra clase la redefinición correcta de `toString`.**

Método `toString`: Ejemplo

```
class ProgramaLavado {
    public ProgramaLavado (//args) {...}
    public void activar () {...}
    public int consumoAgua () {...}
    public String nombre () {...}
    public int duracion () {...}

    public String toString () {
        String cadena;
        cadena = "Programa: " + _nombre + " , ";
        cadena += "Duración: " + _duracion + " , ";
        cadena += "Consumo agua: " + _consumoAgua;
        return ( cadena );
    }

    private String      _nombre;
    private int         _num_fases;
    private int         _duracion;
    private int         _consumo_agua;
    private boolean     _centrifugado;
    private boolean     _prelavado;
}
```

p.toString()

Uso típico:

```
ProgramaLavado p (...);
//...
System.out.print (p);
```

Clases Abstractas

- **Clases abstractas:**
  - ✓ Clases de las que no tiene sentido crear Objetos, pero que agrupan características de clases que derivan de ellas.
  - ✓ Ejemplo:
    - Clase `LavadoraMX`, no tiene sentido que existan objetos (lavadoras) de esta clase:
      - existen lavadoras `MX1`, `MX2` y `MX3deLuxe`.
    - La finalidad de `LavadoraMX` es definir los aspectos comunes a todos los modelos de lavadoras que se fabrican.
  - ✓ Ejemplo:
    - Por la carretera no circulan `Vehículos` en abstracto, circulan `Coches`, `Camiones`, `Bicicletas`, etc...

Clases Abstractas (Java)

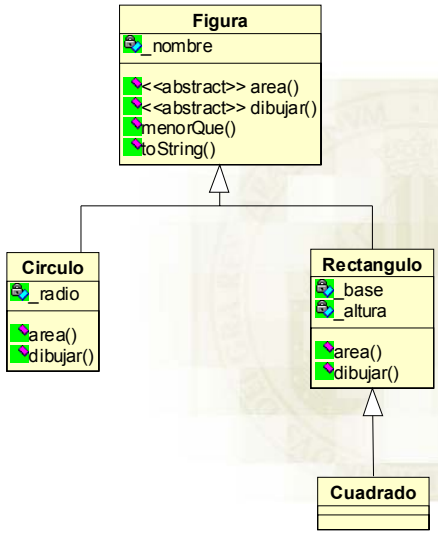
- **En Java se puede indicar explícitamente que una clase es abstracta y no se pueden crear objetos de ella:**

```
abstract class LavadoraMX {
  //...
}

abstract class Vehiculo {
  //...
}
```

Métodos Abstractos

- **Las clases abstractas pueden (y suelen) incluir métodos sin implementación, son los llamados métodos abstractos.**
- **Métodos abstractos:**
  - ✓ Métodos que no tienen implementación y que obligan a ser implementados en las clases derivadas.
- **Si una clase posee un método abstracto → es clase abstracta:**
  - ✓ Si una clase deriva de una clase abstracta y no implementa alguno de los métodos abstractos de la clase base, entonces hereda el método abstracto y se convierte en abstracta:
    - Para poder crear objetos hay que implementar todos los métodos abstractos heredados.

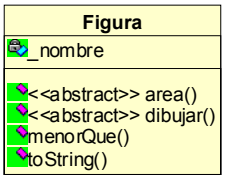
Clases Abstractas: Ejemplo



Clase Figura

No es posible crear Objetos de esta clase

Método sin implementación

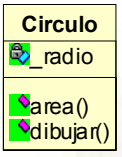



```
abstract class Figura {

    abstract public double area ();

    abstract public void dibujar ();

    public Figura (String nombreFigura) {
        _nombre = nombreFigura;
    }

    public boolean menorQue (Figura f) {
        return ( this.area() < f.area() );
    }

    public String toString () {
        return ( _nombre + " , " + area() );
    }

    private String _nombre;
}
```


Clase Circulo

```
class Circulo extends Figura
{
    public Circulo (double r) {
        super ("Circulo");
        _radio = r;
    }

    public Circulo () {
        this (10.0);
    }

    public double area () {
        return ( PI * _radio * _radio );
    }

    public void dibujar () {
        System.out.println ("Soy un Circulo de radio: " + _radio);
    }

    private double _radio;
    private static final double PI = 3.1415926535879;
}
```

Construye la parte heredada de `Figura`, NO un objeto específico



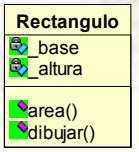
Los métodos abstractos han sido implementados:
Es posible crear objetos `Circulo`

Clase Rectángulo

```
class Rectangulo extends Figura
{
    public Rectangulo (double ancho, double alto) {
        super ("Rectangulo");
        _base = ancho;
        _altura = alto;
    }
    public Rectangulo () {
        this (10.0, 10.0);
    }

    public double area () {
        return ( _base * _altura );
    }

    public void dibujar () {
            System.out.println ("Soy un Rectangulo de: " + _base + "*"
+ _altura);
    }

    private double _base;
    private double _altura;
}
```

Construye la parte heredada de `Figura`, NO un objeto específico

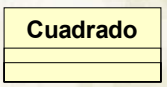
Los métodos abstractos han sido implementados:
Es posible crear objetos `Rectangulo`



Clase Cuadrado

```
class Cuadrado extends Rectangulo
{
    public Cuadrado (double lado) {
        super (lado, lado);
    }

    public Cuadrado () {
        super (10.0, 10.0);
    }
}
```

Deriva de `Rectangulo` que no tiene métodos abstractos:
Es posible crear objetos `Cuadrado`



Ejemplo de utilización de la jerarquía de Figuras

```
class testfiguras {
   public static void main (String[] args) {
      Figura[] fig = new Figura[3];
      int menor;

      fig[0] = new Circulo();
      fig[1] = new Cuadrado(55);
      fig[2] = new Rectangulo(20,30);

      for (int i=0; i<fig.length; i++) {
         System.out.println (fig[i]);
         System.out.println ("Area: " + fig[i].area());
         System.out.println ("Dibujar:");
         fig[i].dibujar();
      }

      if ( fig[0].menorQue (fig[1]) ) menor = 0;
      else menor = 1;
      if ( fig[2].menorQue (fig[menor]) ) menor = 2;
      System.out.println ("La figura menor es:");
      System.out.println (fig[menor]);
   }
}
```

Array de 3 Figuras (no crea objetos)

Se crean 3 Figuras concretas y se almacenan en el array

`fig[i]` aplica los métodos de su clase : `Circulo`, `Cuadrado`, `Rectangulo`

Ejecución

```
Circulo , 314.15926535879
Area: 314.15926535879
Dibujar:
Soy un Circulo de radio: 10.0
Rectangulo , 3025.0
Area: 3025.0
Dibujar:
Soy un Rectangulo de: 55.0*55.
Rectangulo , 600.0
Area: 600.0
Dibujar:
Soy un Rectangulo de: 20.0*30.
La figura menor es:
Circulo , 314.15926535879
```

fig[0] fig[1] fig[2]

```
public static void main (String[] args) {
   Figura[] fig = new Figura[3];
   int menor;

   fig[0] = new Circulo();
   fig[1] = new Cuadrado(55);
   fig[2] = new Rectangulo(20,30);

   for (int i=0; i<fig.length; i++) {
      System.out.println (fig[i]);
      System.out.println ("Area: " + fig[i].area());
      System.out.println ("Dibujar:");
      fig[i].dibujar();
   }

   if ( fig[0].menorQue (fig[1]) ) menor = 0;
   else menor = 1;
   if ( fig[2].menorQue (fig[menor]) ) menor = 2;
   System.out.println ("La figura menor es:");
   System.out.println (fig[menor]);
}
```

Revisión de la Clase Circulo

```
class Circulo extends Figura
{
    public Circulo (double r) {
        super ("Circulo" + _contador);
        _contador ++;
        _radio = r;
    }

    public Circulo () {...}

    public double area () {...}

    public void dibujar () {...}

    private double _radio;
    private static final double PI = 3.1415926535879;
    private static int _contador = 1;
}
```

Ahora el nombre de las Figuras tipo Círculo incluye un número identificativo

Contabiliza el número de círculos creados. Es estático, pertenece a la clase, no a los objetos individuales

Revisión del Test de la jerarquía de Figuras

```
class testfiguras2 {
   public static void main (String[] args) {

      Figura[] fig;
      int num_fig, opcion;
      BufferedReader in = new BufferedReader (new InputStreamReader (System.in));
      final String tab = "     ";

      System.out.print ("Numero de figuras: ");
      num_fig = leerEntero(in);
      fig = new Figura[num_fig];

      for (int i=0; i<fig.length; i++) {
         System.out.print("Figura " + i + " - 1 (Cir.), 2 (Rect.), 3 (Cuad.): ");
         opcion = leerEntero(in);
         switch (opcion) {
            case 1: fig[i] = new Circulo(50); break;
            case 2: fig[i] = new Rectangulo(100,30); break;
            case 3: fig[i] = new Cuadrado(25); break;
            default: fig[i] = new Circulo(); break;
         }
      }

//Continúa...
```

No se sabe el número de Figuras hasta la ejecución del programa

No se sabe el tipo de Figuras hasta la ejecución del programa

Revisión del Test (2)

```
      System.out.println();
      for (int i=0; i<fig.length; i++) {
         System.out.println ("Datos de la figura " + i + " ==>");
         System.out.println (tab + fig[i]);
         if ( fig[i] instanceof Cuadrado )
            System.out.println (tab + "En realidad soy un Cuadrado");
         else {
            System.out.print (tab);
            fig[i].dibujar();
         }
      }
   } //fin main

   private static int leerEntero (BufferedReader f) {
      int x=0;
      String linea;
      try {
         linea = f.readLine();
         x = Integer.parseInt (linea);
      }
      catch (Exception e) { System.err.println(e); }
      return ( x );
   }
} //fin clase
```

¿Es `fig[i]` un Cuadrado?

Ejemplo de Ejecución del 2º test (1)

```
System.out.print ("Numero de figuras: ");
num_fig = leerEntero(in);
fig = new Figura[num_fig];
```

```
Numero de figuras: 3
Figura 0 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 1
Figura 1 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 2
Figura 2 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 3

Datos de la figura 0 ==>
     Circulo 1, 7853.98163396975
     Soy un Circulo de radio: 50.0
Datos de la figura 1 ==>
     Rectangulo, 3000.0
     Soy un Rectangulo de: 100.0*30.0
Datos de la figura 2 ==>
     Rectangulo, 625.0
     En realidad soy un Cuadrado
```

```
fig[0] = new Circulo(50);
fig[1] = new Rectangulo(100,30);
fig[2] = new Cuadrado(25);
```

```
for (int i=0; i<fig.length; i++) {
   System.out.println ("Datos de la
                        figura " + i + " ==>");
   System.out.println (tab + fig[i]);
   if ( fig[i] instanceof Cuadrado )
      System.out.println (tab +
      "En realidad soy un Cuadrado");
   else {
      System.out.print (tab);
      fig[i].dibujar();
   }
}
```

Ejemplo de Ejecución del 2º test (2)

```
System.out.print ("Numero de figuras: ");
num_fig = leerEntero(in);
fig = new Figura[num_fig];
```

```
Numero de figuras: 5
Figura 0 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 3
Figura 1 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 2
Figura 2 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 1
Figura 3 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 2
Figura 4 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 3
```

```
fig[0] = new Cuadrado(25);
fig[1] = new Rectangulo(100,30);
fig[2] = new Circulo(50);
fig[3] = new Rectangulo(100,30);
fig[4] = new Cuadrado(25);
```

```
Datos de la figura 0 ==>
     Rectangulo, 625.0
     En realidad soy un Cuadrado
Datos de la figura 1 ==>
     Rectangulo, 3000.0
     Soy un Rectangulo de: 100.0*30.0
Datos de la figura 2 ==>
     Circulo 1, 7853.98163396975
     Soy un Circulo de radio: 50.0
Datos de la figura 3 ==>
     Rectangulo, 3000.0
     Soy un Rectangulo de: 100.0*30.0
Datos de la figura 4 ==>
     Rectangulo, 625.0
     En realidad soy un Cuadrado
```

```
for (int i=0; i<fig.length; i++) {
   System.out.println ("Datos de la
               figura " + i + " ==>");
   System.out.println (tab + fig[i]);
   if ( fig[i] instanceof Cuadrado )
      System.out.println (tab + "En
         realidad soy un Cuadrado");
   else {
      System.out.print (tab);
      fig[i].dibujar();
   }
}
```


Ejemplo de Ejecución del 2º test (3)

```
System.out.print ("Numero de figuras: ");
num_fig = leerEntero(in);
fig = new Figura[num_fig];
```

```
Numero de figuras: 5
Figura 0 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 1
Figura 1 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 1
Figura 2 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 1
Figura 3 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 1
Figura 4 - 1 (Circ.), 2 (Rect.), 3 (Cuadr.): 1
```

```
fig[0] = new Circulo(50);
fig[1] = new Circulo(50);
fig[2] = new Circulo(50);
fig[3] = new Circulo(50);
fig[4] = new Circulo(50);
```

```
Datos de la figura 0 ==>
     Circulo 1, 7853.98163396975
     Soy un Circulo de radio: 50.0
Datos de la figura 1 ==>
     Circulo 2, 7853.98163396975
     Soy un Circulo de radio: 50.0
Datos de la figura 2 ==>
     Circulo 3, 7853.98163396975
     Soy un Circulo de radio: 50.0
Datos de la figura 3 ==>
     Circulo 4, 7853.98163396975
     Soy un Circulo de radio: 50.0
Datos de la figura 4 ==>
     Circulo 5, 7853.98163396975
     Soy un Circulo de radio: 50.0
```

```
for (int i=0; i<fig.length; i++) {
   System.out.println ("Datos de la
               figura " + i + " ==>");
   System.out.println (tab + fig[i]);
   if ( fig[i] instanceof Cuadrado )
      System.out.println (tab + "En
         realidad soy un Cuadrado");
   else {
      System.out.print (tab);
      fig[i].dibujar();
   }
}
```

Métodos finales

- **En ocasiones interesa indicar que un determinado método no se puede redefinir → Método final**

```
final public boolean menorQue (Figura f) {
       return ( this.area() < f.area() );
}
```

✓ Las clases derivadas NO pueden redefinir **`menorQue`**

- Ojo: redefinir implica mantener la signatura del método.

- **Los métodos finales tienen ligadura estática (compilación).**

Clases finales

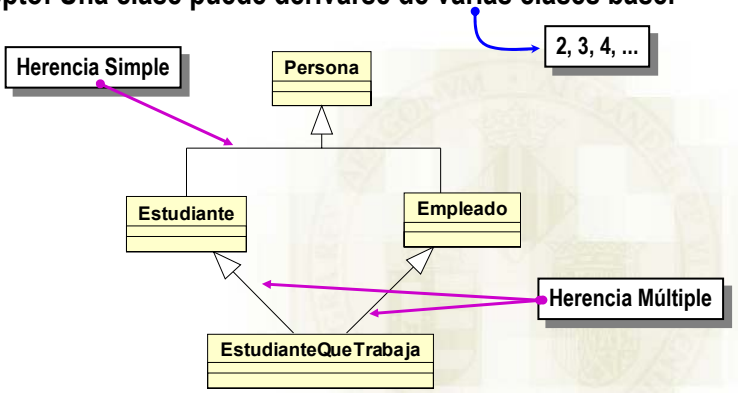
- **Se puede prohibir que una clase sea extendida declarándola como final.**

```
final class Cuadrado extends Rectangulo
{
       public Cuadrado (double lado) {
              super (lado, lado);
       }

       public Cuadrado () {
              super (10.0, 10.0);
       }
}
```

✓ No es posible definir una nueva clase que derive de **`Cuadrado`**

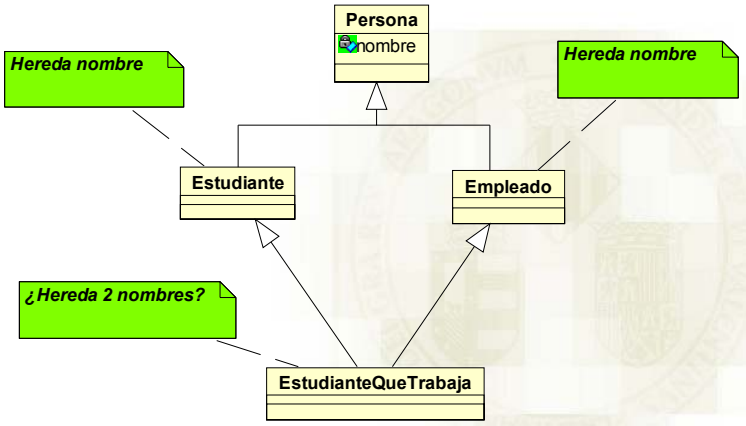
Herencia múltiple

- Concepto: Una clase puede derivarse de varias clases base.

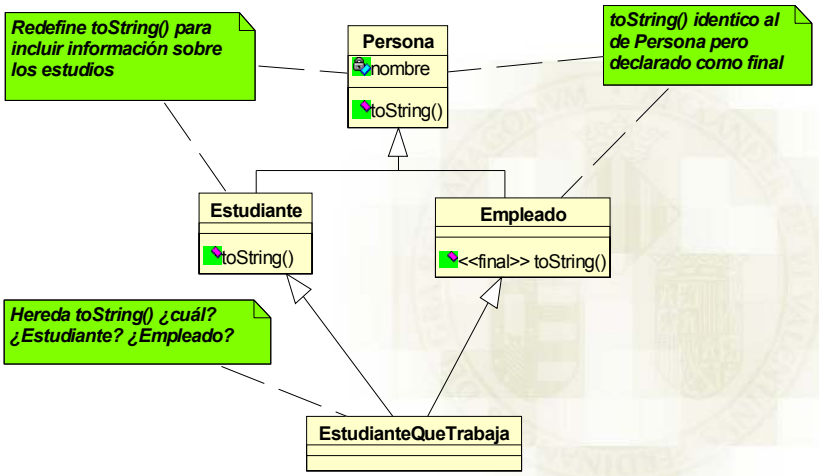


Un `EstudianteQueTrabaja` es un `Estudiante` y es un `Empleado`

Herencia múltiple: Conflictos (1)



Herencia múltiple: Conflictos (2)



Herencia múltiple

- **No siempre es problemática, pero se procura evitar, en la medida de los posible.**
- **Hay lenguajes O.O. que directamente no la soportan:**
  - ✓ *Java*, *Smalltalk*.
- **Otros lenguajes O.O. sí que la admiten:**
  - ✓ *C++*.