UF7.3
Herencia - Avanzada

Universidad Europea

LAUREATE INTERNATIONAL UNIVERSITIES

Madrid Valencia Canarias

CONTENIDOS

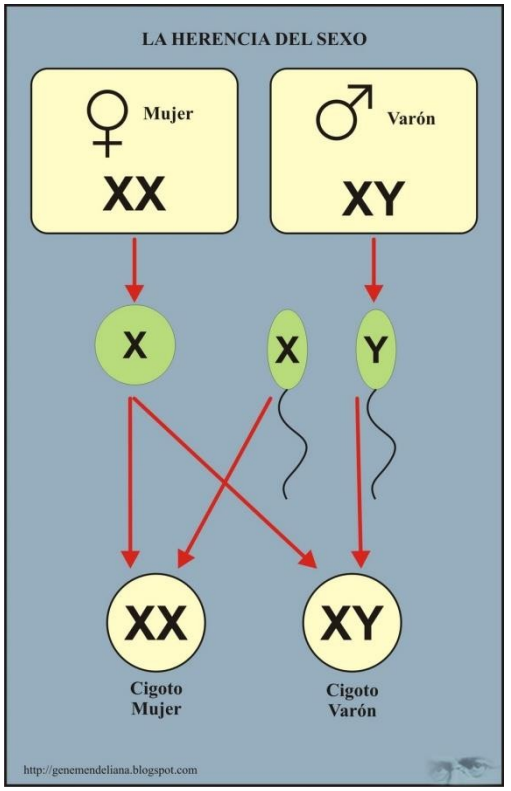
1. Adaptando lo heredado: Adición y redefinición.
2. Super
3. Protección
4. Clases abstractas
5. Interfaces
6. Herencia múltiple

Adaptando lo heredado

La herencia en sí no sería tan interesante si no fuera por la posibilidad de adaptar en el descendiente los miembros heredados.

Hay dos posibilidades en los descendientes:

- Adición. Trivialmente el descendiente puede añadir nuevos atributos y métodos que se suman a los recibidos a través de la herencia
- Redefinición. Es posible redefinir la implementación de una operación heredada para adaptarla a las características de la clase descendiente. También es posible cambiar el tipo de un atributo heredado.



Redefinición

La redefinición se realiza en Java y la mayoría de los lenguajes OO definiendo nuevamente la operación (con los mismos argumentos) en el descendiente.

Las clases descendientes TPReloj, TPEjecucion y TPAviso no están operativas todavía porque la implementación de ejecutarTarea() que contienen es la heredada de TareaPeriodica, que no hace nada en particular.

Es preciso redefinir esta operación en cada una de las subclases para que realicen las tareas correspondientes.

super

Después de la redefinición, en el descendiente es posible llamar a la versión original de la operación en el ascendiente mediante:

```
super.operacionRedefinida()
```

Protección

Hemos visto anteriormente como los distintos niveles de protección limitan el acceso a los miembros de la clase desde el exterior. ¿Pero como afectan estos niveles de protección a los miembros heredados?

- Miembros públicos. Son accesibles desde los descendientes, y se heredan como públicos.
- Miembros privados. No son accesibles desde los descendientes.
- Miembros con acceso a nivel de paquete. Son accesibles desde los descendientes siempre y cuando pertenezcan al mismo paquete que el ascendiente. Se heredan con el mismo nivel de protección

Un nuevo nivel de protección es el de miembros protegidos (protected). Un miembro protegido es accesible únicamente desde los descendientes.

Protección

Además, un miembro protegido mantiene en las subclases el nivel de acceso protegido.

En nuestro ejemplo, los atributos de la clase TareaPeriodica son accesibles desde TPReloj, TPEjecucion y TPAviso porque al pertenecer al mismo paquete son amigas.
Para permitir el acceso a los atributos de la clase TareaPeriodica únicamente desde los descendientes es conveniente marcarlos como protegidos.

Definición

Existen clases que representan conceptos tan genéricos que no tiene sentido su instanciación en objetos.

Además en este tipo de clases puede ser imposible o inútil la implementación de ciertas operaciones.

La utilidad de este tipo de clases está en la aplicación de herencia para obtener clases que representan conceptos concretos para los que sí que tiene sentido su instanciación.

La clase TareaPeriodica es un claro ejemplo: por sí sola no tiene utilidad, pero simplifica mucho la construcción de las otras tres clases. De hecho, la operación ejecutarTarea() en TareaPeriodica no tiene una implementación útil.

CLASES ABSTRACTAS

Definición

Estas clases se denominan clases abstractas y este tipo de operaciones “sin implementación posible”, operaciones abstractas. Las clases abstractas no pueden instanciarse

Las operaciones abstractas deben ser implementadas obligatoriamente en alguna de las subclases para que la clase correspondiente sea instanciable.

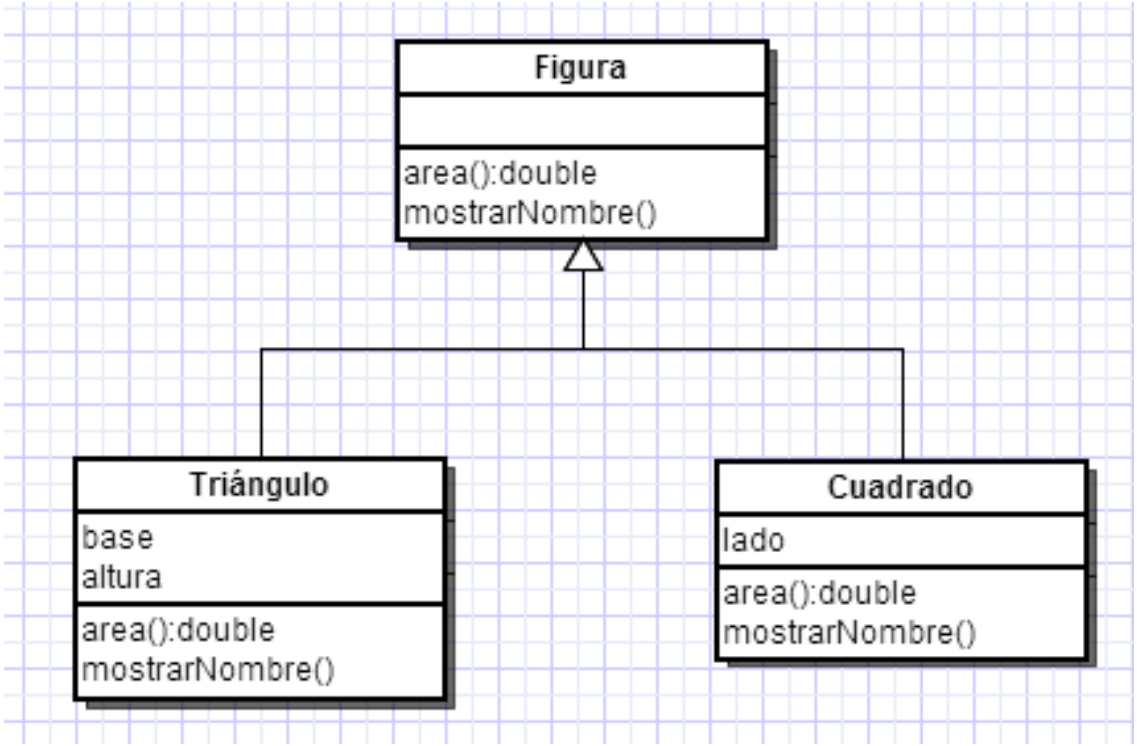
Una clase abstracta puede no tener ninguna operación abstracta, pero una clase que contenga al menos una operación abstracta debe ser declarada como abstracta.

En Java, utilizando la declaración **abstract** podremos establecer una clase o una operación como abstracta.

CLASES ABSTRACTAS

Ejercicio (1/5)

Sea la clase Figura de tipo abstracta y Triángulo y Cuadrado descendientes suyos:



Trata de hacerlo sin mirar la solución

CLASES ABSTRACTAS

Ejercicio (2/5)

```
public abstract class Figura {

        // No tienen implementación
        public abstract double area();
        public abstract void mostrarNombre();
}
```

Ejercicio (3/5)

```
public class Triangulo extends Figura {

        //Atributos
        protected int base, altura;

        //Constructor
        public Triangulo (int ba, int al) {
                base=ba;
                altura=al;
        }

        //Métodos implementados
        public double area() { return base*altura/2; }

        public void mostrarNombre() {
                System.out.println("La figura es un triángulo");
        }

}
```

Ejercicio (4/5)

```
public class Cuadrado extends Figura {

        //Atributo
        protected int lado;

        //Constructor
        public Cuadrado (int lado) { this.lado=lado; }

        //Métodos
        public double area() { return lado*lado; }

        public void mostrarNombre() {
                System.out.println("La figura es un Cuadrado");
        }

}
```

CLASES ABSTRACTAS

Ejercicio (5/5)

```
public class PruebaClaseAbstracta {
        public static void main(String[] args) {
                Figura fig;
                Triangulo tri;
                Cuadrado cua;

                //fig = new Figura(); error porque no se puede instanciar una clase abstracta
                tri = new Triangulo(4,3);
                tri.mostrarNombre();

                fig = tri;
                fig.mostrarNombre();
                System.out.println("Area triangulo: "+fig.area());

                cua = new Cuadrado(5);

                fig = cua;
                fig.mostrarNombre();
                System.out.println("Area cuadrado: "+fig.area());
        }
}
```

INTERFACES

Totalmente abstracto

La idea de clase abstracta, llevada al extremo, nos lleva en Java a las interfaces. Una **interfaz** es similar a una clase totalmente abstracta:

- Todas las operaciones de la interfaz son implícitamente abstractas, es decir, carecen de implementación
- Una interfaz no puede contener atributos

Las interfaces sirven para especificar los métodos que obligatoriamente deben implementar una serie de clases.

La implementación de una interfaz no se realiza mediante herencia (extends) sino mediante implements.

No obstante, el comportamiento es similar al de la herencia, aunque más sencillo.

Otros aspectos

Una clase puede implementar más de una interfaz.
Una interfaz puede heredar de otra interfaz.
¿Cuando utilizar una interfaz en lugar de una clase abstracta?

- Por su sencillez se recomienda utilizar interfaces siempre que sea posible.
- Si la clase debe incorporar atributos, o resulta interesante la implementación de alguna de sus operaciones, entonces declararla como abstracta.

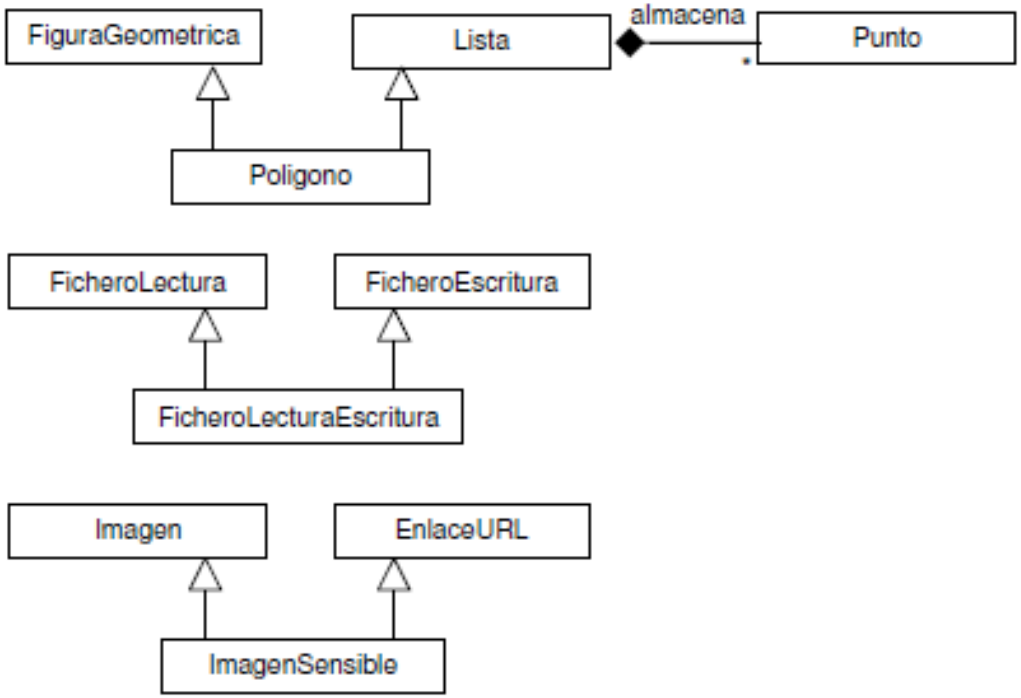
En la biblioteca de clases de Java se hace un uso intensivo de las interfaces para caracterizar las clases.
Algunos ejemplos:

- Para que un objeto pueda ser guardado en un fichero la clase debe implementar la interfaz **Serializable**
- Para que un objeto sea duplicable, su clase debe implementar **Cloneable**
- Para que un objeto sea ordenable, su clase debe implementar **Comparable**

HERENCIA MÚLTIPLE

Varios padres

Consiste en la posibilidad de que una clase tenga varios ascendientes directos. Puede surgir de manera relativamente frecuente y natural durante el diseño.



Nota: En Java no existe en Herencia múltiple