

Tema 3.8

Clases internas, Reflexión

Análisis y Diseño de Software
2º Ingeniería Informática
Universidad Autónoma de Madrid


Indice

- **Clases internas.**
- Reflexión.

Clases Internas

- Java permite declarar una clase dentro de otra.
- Clases internas, o anidadas.
- Pueden ser estáticas o no.
- Son miembros de la clase externa y se declaran con control de acceso.

```
public class Externa{  
    ...  
    private class Interna {  
        ...  
    }  
}
```



Con acceso a miembros de Externa
(incluso privados)

```
public class Externa{  
    ...  
    static class Interna {  
        ...  
    }  
}
```

Clases Internas estáticas

```
class Pelicula {  
    private String titulo;  
    private Director director;  
  
    public Pelicula( String t,  
                    Director d) {  
        this.titulo = t;  
        this.director = d;  
    }  
}
```

```
public class InternaEstatica {  
    public static void main(String[] args) {  
        Pelicula.Director d = new Pelicula.Director(  
            "John Badham", "British");  
        Pelicula p = new Pelicula("WarGames", d );  
    }  
}
```

```
public static class Director {  
    private String nombre;  
    private String nacionalidad;  
  
    public Director(String nm, String nac) {  
        this.nombre = nm;  
        this.nacionalidad = nac;  
    }  
}
```

Son similares a las clases externas
Simplemente empaquetadas dentro
de otra clase
Acceden a los miembros de la clase
externa como cualquier otra clase.

Clases Internas

- Los objetos de la clase interna “viven” dentro de un objeto de la clase externa.
- Pueden acceder a los miembros de la clase externa (incluso si son privados).

```
public class Externa{  
    ...  
    public class Interna {  
        ...  
    }  
}  
  
Externa.Interna objIn = objOut.new Interna();
```

Clases Internas. Ejemplo (1/2)

```
class Autor {  
    private String nombre;  
    private List<Libro> libros = new ArrayList<Libro>();  
  
    public Autor(String n ) { this.nombre = n; }  
    public void addLibro (String t) { new Libro(t); }  
    @Override public String toString() {  
        return "Autor: "+this.nombre+". Libros= "+this.libros;  
    }  
  
    public class Libro {  
        private String titulo;  
        public Libro(String t) {  
            this.titulo = t;  
            Autor.this.libros.add(this);  
        }  
        @Override public String toString() {return titulo;}  
    }  
}
```

Clases Internas. Ejemplo (2/2)

```
public class Interna {  
    public static void main(String[] args) {  
        Autor p = new Autor("Chuck Palaniuk");  
        p.addLibro("Nana");  
        p.addLibro("El club de la lucha");  
        Autor.Libro lib = p.new Libro("Asfixia");  
        System.out.println(p);  
    }  
}
```

Clases Locales Anónimas

- Una clase que se declara dentro de un bloque de código.
- Una clase anónima es una clase local sin nombre:
 - se declara e instancia sobre la marcha.
 - se definen sobre una clase base, o un interfaz.
- Pueden acceder a los miembros de la clase externa, como en las clases internas, y además a las variables locales finales del método actual (definidas con final), o efectivamente finales.

Clases Anónimas

```
public class SimpleWindow {
```

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

```
        // crear ventana
```

```
        JFrame ventana = new JFrame("Mi GUI");
```

```
        // ...
```

```
        JButton boton = new JButton("Haz click");
```

```
        final JTextField campo = new JTextField(5);
```

```
        // asociar acciones a componentes
```

```
        boton.addActionListener(
```

```
            new ActionListener() {
```

```
                public void actionPerformed(ActionEvent e) {
```

```
                    JOptionPane.showMessageDialog(null, campo.getText());
```

```
                }
```

```
            }
```

```
        );
```

```
        ...
```

```
    }
```

```
}
```

Clases Anónimas y Enumerados

- Es posible declarar métodos abstractos en un *enum*, e implementarlos en cada objeto del enum
- Para ello usamos clases anónimas.

```
enum PuertaLogica {  
    AND {  
        @Override Boolean calculate(Boolean b1, Boolean b2) {  
            return b1 && b2;  
        }  
    }, OR {  
        @Override Boolean calculate(Boolean b1, Boolean b2) {  
            return b1 || b2;  
        }  
    };  
  
    abstract Boolean calculate(Boolean b1, Boolean b2);  
}
```

Indice

- Clases anónimas e internas.
- **Reflexión.**

Reflexión

- Examinar y cambiar el comportamiento de un programa en tiempo de ejecución.
- Técnica muy potente:
 - Podemos examinar el tipo de un objeto, acceder a sus atributos y métodos, etc.
 - Podemos crear objetos simplemente a partir de una cadena con el nombre de la clase.
 - Realizar acciones que de otra manera serían ilegales (e.g., acceder a atributos o métodos privados).
- Uso con prudencia:
 - Menor rendimiento.
 - Restricciones de seguridad (e.g., no posible para Applets).
 - Exposición de miembros internos (e.g. privados).

Operador instanceof

- Operador binario infijo.
- Toma como parámetros una referencia y una clase o interfaz como segundo parámetro.
- Devuelve cierto si el tipo del objeto en tiempo de ejecución es compatible con el tipo.
- Su uso suele indicar un mal diseño del código.

Ejemplo

```
class A {}
```

```
class B extends A {}
```

```
public class Reflexion1 {  
    public static void main(String[] args) {  
        A a = new B();  
        if (a instanceof B)  
            System.out.println("Es de tipo B");  
        else System.out.println("Es de tipo A");  
    }  
}
```

Salida: Es de tipo B

Un Ejemplo de MAL diseño

```
public abstract class Reserva {  
    protected String codigo;  
    //....  
    public String getCodigo() { /*...*/ }  
}
```

```
public class ReservaHotel extends Reserva  
{ //...}
```

```
public class ReservaVuelo extends Reserva  
{ //...}
```

```
public class GestorReservas {  
    private List<Reserva> reservas = new ArrayList<Reserva>();  
    public boolean cancelar(String codigo) {  
        Reserva r = this.getReserva(codigo);  
        if (r instanceof ReservaHotel) { /* cancela Reserva Hotel */ }  
        else if (r instanceof ReservaVuelo) { /* cancela Reserva Vuelo */ }  
        else if (r instanceof ReservaViaje) { /* cancela Reserva Viaje */ }  
        //...  
        return false;  
    }  
    private Reserva getReserva(String codigo) { /*...*/ }  
}
```

Este diseño es MEJOR

```
public abstract class Reserva {  
    protected String codigo;  
    //....  
    public String getCodigo() { /*...*/ }  
    public abstract boolean cancelar();  
}
```

```
public class ReservaHotel extends Reserva  
{  
    public boolean cancelar() { /*...*/ }  
}
```

```
public class ReservaVuelo extends Reserva  
{  
    public boolean cancelar() { /*...*/ }  
}
```

```
public class GestorReservas {  
    private List<Reserva> reservas = new ArrayList<Reserva>();  
    public boolean cancelar(String codigo) {  
        Reserva r = this.getReserva(codigo);  
        if (r==null) return false;  
        return r.cancelar();  
    }  
    private Reserva getReserva(String codigo) { /*...*/ }  
}
```

¿Por qué es mejor?

Clase Class

- Un objeto que representa una clase o interfaz.
- Class no tiene constructor público. Los objetos los construye la máquina virtual de Java.
- Lo más sencillo es usar el método getClass() de Object

```
public class Reflexion2 {  
    public static void main(String[] args) {  
        Class<?> clase = "un string".getClass();  
        System.out.println(" clase = "+clase);  
    }  
}
```

Salida: clase = class java.lang.String

```
import java.lang.reflect.Field;
class Asignatura{
    private String nombre = "PADS";
    public Asignatura() {}
    public Asignatura(String name) { this.nombre = name; }
    @Override public String toString() { return "Asignatura = "+this.nombre; }
}
```

Ejemplo

```
public class Reflexion3 {
    public static void main(String[] args) throws
        ClassNotFoundException, InstantiationException,
        IllegalAccessException, NoSuchFieldException, SecurityException
    {
        Class<?> clase = Class.forName("Asignatura");

        Object asig = clase.newInstance(); // necesita constructor sin parámetros
        System.out.println(asig);
        Field fld = clase.getDeclaredField("nombre");
        fld.setAccessible(true);           // nos podemos saltar la privacidad...
        fld.set(asig, "ADS");              // pero no conviene hacerlo!!!
        System.out.println(asig);
    }
}
```