

## Clases anidadas

Puri Arenas Sánchez (Grupos A y D)  
Yolanda García Ruiz (Grupo E)  
Facultad de Informática (UCM)  
Curso 2019/2020

1

## Clases internas

- Son clases que se implementan DENTRO de otras clases o interfaces
- ¿Por qué?
  - **Organización de código**  
Clases que no tienen sentido si no son en conjunción con otras
  - **Acceso a miembros privados**  
No queremos dar acceso a métodos / atributos privados a otras clases pero tenemos alguna clase que necesita dicho acceso
  - **Clases de ayuda**  
Sólo se usan en la propia clase que las define

## Clases internas – Utilidad?

- Las clases internas son interesantes porque nos permiten agrupar clases relacionadas y controlar la visibilidad mutua de esas clases.
- No hay que pensar en ellas como una forma de ocultar código.
- Una clase interna conoce los detalles de la clase contenedora y puede comunicarse con ella.
- Además el tipo de código que puede escribirse con las clases internas suele ser más elegante.

3

## Tipos de clases

- Las clases (e interfaces) que hemos visto hasta ahora están declaradas en el nivel más externo (top level ) es decir, a nivel de paquete.
  - Top-level package member classes and interfaces
- Existen otros cuatro tipos de clases (o interfaces):
  - Top-level nested classes and interfaces (**Clases internas estáticas**)
  - Non-static inner classes (**Clases internas no estáticas**)
  - Local classes
  - Anonymous Classes
- Las tres últimas necesitan de la clase que las encapsula para poder existir

4

## Top-level nested classes (Clases internas estáticas)

### Clases e interfaces anidados a nivel de paquete

Clases internas declaradas como miembro estático de la clase contenedora

- Se le puede añadir cualquier tipo de modificador (public, private...)
- Se pueden instanciar como cualquier otra clase y **no necesitan de la clase contenedora para poder existir**
- Si una clase anidada posee un atributo (no constante) estático, entonces debemos marcarla como clase estática.

```
public class ClaseExterna {
    private static class ClaseInternal1{
        private int valor;
        public ClaseInternal1(int v) {
            this.valor = v;
        }
    }

    public static class ClaseInternal2{
        private String name;
        public ClaseInternal2() {
            this.name = "Julian";
        }
    }
}
```

5

## Top-level nested classes

- Clases internas declaradas como **miembro estático** de la clase contenedor

```
public class ClaseExterna {
    private static Vector<Integer> valores = new Vector<Integer>();

    private static class ClaseInternal1{
        private int valor;
        public ClaseInternal1(int v) {
            this.valor = v;
        }
        public int metodoAcceso(){
            return ClaseExterna.valores.size();
        }
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ClaseInternal1 obj1 = new ClaseInternal1(9);
        System.out.println(obj1.metodoAcceso());
    }
}
```

- Tienen acceso a los miembros estáticos de la clase que la contiene
- Tienen acceso a los miembros no estáticos a través de una instancia de la clase contenedora

6

## Top-level nested classes

- El nombre de la clase incluye el nombre de la clase en la que está definida la clase anidada.

```
public class Main {
    public static void main(String[] args) {
        ClaseExterna c = new ClaseExterna();
        ClaseExterna.ClaseInterna1 f1 = new ClaseExterna.ClaseInterna1();
        ClaseExterna.ClaseInterna2 f2 = new ClaseExterna.ClaseInterna2();
    }
}
```

ERROR →

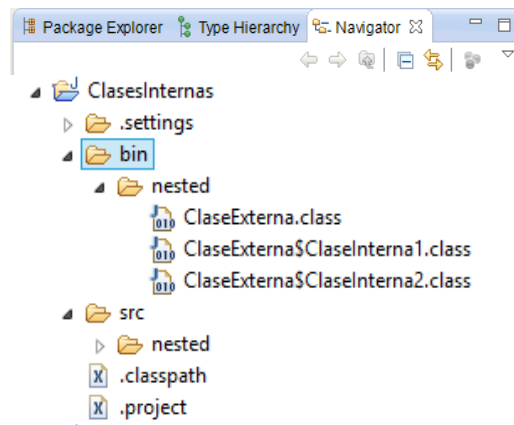
ClaseInterna1 tiene visibilidad private

- Si la visibilidad lo permite, las clases internas se podrán utilizar desde otras clases. Para referirse a ellas se utiliza el “punto”(.) para separar el nombre de la clase externa y el de la interna.

7

## Top-level nested classes

- El compilador crea un fichero (.class) por cada clase



8

## Top-level nested classes

- Los métodos de una clase interna pueden tener acceso a los métodos y atributos *estáticos* de la clase externa.

```
public class ClaseExterna2 {
    private static int i;
    private int j;
    private static void staticMethod() {
        System.out.println("Método estático en la clase Externa");
    }

    private static class ClaseInternaNivel1 {

        public static void staticMethod() {
            nonStaticMethod(); // MAL ERROR
            System.out.println(i);
            System.out.println(j); // MAL ERROR
            System.out.println("Método estático en la clase interna nivel 1");
        }

        protected static class ClaseInternaNivel2 {
            public void nonStaticMethod() {
                staticMethod(); // OK
            }

            public static void main(String[] args) {
                ClaseInternaNivel2 u = new ClaseInternaNivel2();
                u.nonStaticMethod();
                System.out.println("Hola");
            }
        } // Fin clase nivel 3
    } // fin clase nivel 2
}
```

9

```
public class OuterClass {
    static private int outerA;
    private int outerB;
    static public int c;
}
```

```
    public static class NestedClass {
        static private int nestedA;
        private int nestedB;

        public void nestedMethod() {
            OuterClass.outerA = 3;
            OuterClass.outerB = 15;
            OuterClass.c = 9;
            OuterClass o1 = new OuterClass();
            o1.outerB = 3;
            o1.outerMethod();
        }
    }
```

Error.  
Acceso a atributo no estático

```
    public void outerMethod() {
        NestedClass.nestedA = 3;
        NestedClass.nestedB = 12;
        NestedClass o2 = new NestedClass();
        o2.nestedB = 10;
        o2.nestedMethod();
    }
```

Error.  
Acceso a atributo no estático

10

## Ejemplo:

```
public class Cuental {

    private int numCuenta;
    private int saldo;

    public Cuental(int nc, int s) {
        this.numCuenta = nc;
        this.saldo = s;
    }

    public static class OrdenaPorSaldo implements Comparator<Cuental> {
        @Override
        public int compare(Cuental cuenta1, Cuental cuenta2) {
            if (cuenta1.getSaldo() == cuenta2.getSaldo()) return 0;
            else if (cuenta1.getSaldo() < cuenta2.getSaldo()) return -1;
            else return 1;
        }
    }

    public int getNumCuenta() { return this.numCuenta; }
    public int getSaldo() { return this.saldo; }
    public String toString() {
        return "Numero cuenta: " + this.numCuenta + "\n"
            + "Saldo: " + this.saldo;
    }
}
```

11

## Ejemplo:

```
public class Clientel {
    private int nif;
    private TreeSet<Cuental> cuentas = new TreeSet<Cuental>();

    public Clientel(int nif) {
        this.nif = nif;
        Cuental.OrdenaPorSaldo comp = new Cuental.OrdenaPorSaldo();
        cuentas = new TreeSet<Cuental>(comp); // pasamos un orden
    }

    public boolean addCuenta(Cuental c) { return cuentas.add(c); }

    public String toString() {
        Iterator<Cuental> it = cuentas.iterator();
        String s = "";
        while (it.hasNext()) {
            Cuental c = it.next();
            s = s + c.toString() + "\n";
        }
        return s;
    }

    public static void main(String[] args) {
        Clientel c = new Clientel(2);
        Cuental cuenta1 = new Cuental(1,1000);
        c.addCuenta(cuenta1);
        Cuental cuenta3 = new Cuental(3,3000);
        c.addCuenta(cuenta3);
        Cuental cuenta2 = new Cuental(2,2000);
        c.addCuenta(cuenta2);
        System.out.println(c.toString());
    }
}
```

12

## Ejemplo

```
public class Externa {
    public static int f1;
    protected int f3;
    static class NestedClass1 {
        private static int f2;
        private static class NestedClass2 {
            private void moo() {
                Externa.f1 = 1;
                Externa.NestedClass1.f2 = 1;
                f3 = 1; ← Error
            }
        }
    }
}
```

13

## Ejemplo

```
public class Externa {
    static class NestedClass1 {
        public static int f1;
        private int f4;
        static class NestedClass2 {
            private static int f2;
            public int f3;
        }
    }

    void foo() {
        NestedClass1.f1 = 1;
        NestedClass1.NestedClass2.f2 = 1;
        NestedClass1.NestedClass2.f3 = 1; ← Error
        NestedClass1.f4 = 1; ← Error
    }
}
```

14

## Creación de objetos de una clase anidada desde otras clases

```

public class A {
    public static int f1;
    public int f2;
    private int f3;

    static public class A1 {
        public static int g1;
        public int g2;
        private int g3;

        static public class A2 {
            public static int h1;
            public int h2;
            private int h3;
        }
    }
}

void metodoClaseA(){
    A a = new A();
    a.f2 = 1;
    a.f3 = 2;
    A.f1 = 3;
    A.A1 a1 = new A.A1();
    a1.g2 = 4;
    a1.g3 = 5;
    A.A1.g1 = 6;
    a1.f2 = 7;
    a1.h2 = 8;
    A.A1.A2 a2 = new A.A1.A2();
    a2.h2 = 9;
    a2.h3 = 10;
    A.A1.A2.h1 = 11;
    a2.f2 = 13;
    a2.g2 = 14;
}

```

15

## Creación de objetos de una clase anidada desde otras clases

```

public class Externa { // paquete 1
    static class NestedClass1 {
        public static int f1;
        private int f4;

        static class NestedClass2 {
            private static int f2;
            public int f3;
        }
    }
}

public class OtherClass { // paquete1
    void foo() {
        Externa.NestedClass1 x = new Externa.NestedClass1();
        Externa.NestedClass1.NestedClass2 y = new
            Externa.NestedClass1.NestedClass2();
        Externa.NestedClass1.f1 = 3;
        x.f4 = 3;
        Externa.NestedClass1.NestedClass2.f2 = 12;
        y.f3 = 15;
    }
}

```

16



## Creación de objetos de una clase anidada desde otras clases

```

public class Externa { // paquete 1
    static class NestedClass1 {
        public static int f1;
        private int f4;

        static class NestedClass2 {
            private static int f2;
            public int f3;
        }
    }
}

public class OtherClass { // paquete2
    void foo() {
        Externa.NestedClass1 x = new Externa.NestedClass1();
        Externa.NestedClass1.NestedClass2 y = new
            Externa.NestedClass1.NestedClass2();
        Externa.NestedClass1.f1 = 3;
        x.f4 = 3;
        Externa.NestedClass1.NestedClass2.f2 = 12;
        y.f3 = 15;
    }
}

```

17

## Creación de objetos de una clase anidada desde otras clases

```

public class Externa { // paquete 1
    static public class NestedClass1 {
        public static int f1;
        private int f4;

        static public class NestedClass2 {
            private static int f2;
            public int f3;
        }
    }
}

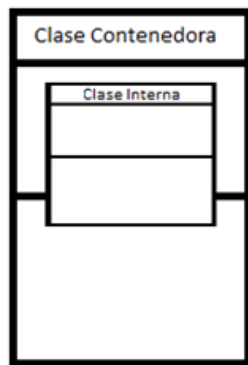
public class OtherClass { // paquete2
    void foo() {
        Externa.NestedClass1 x = new Externa.NestedClass1();
        Externa.NestedClass1.NestedClass2 y = new
            Externa.NestedClass1.NestedClass2();
        Externa.NestedClass1.f1 = 3;
        x.f4 = 3;
        Externa.NestedClass1.NestedClass2.f2 = 12;
        y.f3 = 15;
    }
}

```

18

## Non-static inner classes (Clases internas no estáticas)

- Clases internas no estáticas declaradas dentro de la clase contenedora
- Se le puede añadir cualquier tipo de modificador de visibilidad
- Sólo pueden existir en el contexto de una instancia de la clase externa



- Se puede crear un objeto de la clase interna sólo si se crea un objeto de la clase externa

ERROR

```
Externa.Interna a = new Externa.Interna(..);
```

Correcto

```
Externa.Interna a = (new Externa(..)).new Interna(...);
```

19

## Non-static inner classes (Clases internas no estáticas)

- No puede tener miembros `static`
- Tienen acceso a todos los miembros de la clase externa
  - Guardan una referencia oculta a la instancia de la clase externa
  - Si la clase interna y externa tienen atributos con el mismo nombre, el interno tapa al externo
  - Se puede acceder con `ClaseExterna.this.tribExterno`

Las clases internas, junto con las clases anónimas son las más utilizadas

20

## Non-static inner classes

```
public class ClaseExterna4 {
    public String msg = "Mensaje";

    public InternaNoEstatica1 crea () {
        return new InternaNoEstatica1 ();
    }


    public class InternaNoEstatica1 {
        private String str ;

        public InternaNoEstatica1 () {
            str = msg;
        }

        public void printMsg () {
            System.out.println(str);
            System.out.println(msg);
        }
    }

    public static void main(String[] args) {
        ClaseExterna4 c = new ClaseExterna4();
        ClaseExterna4.InternaNoEstatica1 c1 = new ClaseExterna4.InternaNoEstatica1();
        ClaseExterna4.InternaNoEstatica1 c2 = c.crea();
    }
}
```

**ERROR**



21

## Non-static inner classes

```
public class Class1 {
    Externa x = new Externa(1);
    Externa y = new Externa(1);

    Externa.Interna b = x.foo();
    Externa.Interna c = y.new Interna ();
}
```

```
public void p() {
    x.foo();
    b.getF();
    c.setF(5);
    x.getF(3); // Error
    y.setF(5); // Error
}
```

```
public class Externa {
    private int f;
    public Externa(int n) { this.f = n; }

    public class Interna {
        public int getF() { return f; }
        public void setF(int n) { f = n; }
    }
    Interna foo() { return new Interna(); }
}
```

22

## Non-static inner classes. this

```
public class Externa {

    private int f;
    public Externa (int m) { this.f = m; }

    public class Interna {
        private int f;
        public Interna(int m) { this.f = m; }

        public void test() {
            System.out.println(this.f + " " + Externa.this.f);
        }
    }
}
```

Devuelve 4 1 !!

```
public class Main {
    public static void main(String[] args) {
        Externa x = new Externa(1);
        Externa.Interna a = x.new Interna(4);
        a.test();
    }
}
```

23

## Non-static inner classes. this

```
package paquete1;
```

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

```
package paquete1;
```

```
public class SomeClass {
    Externa x = new Externa();
    Externa.Interna a = x.new Interna();
}
```

Interna es privada

24

## Non-static inner classes. this

```
package paquete1;
```

```
public class Externa {
    protected class Interna {
    }
}
```

```
package paquete1;
```

```
public class SomeClass {
    Externa x = new Externa();
    Externa. Interna a = x.new Interna();
}
```



Correcto.  
Estamos en el mismo paquete

25

## Non-static inner classes. this

```
package paquete1;
```

```
public class Externa {
    protected class Interna {
    }
}
```

```
package paquete2;
```

```
public class SomeClass {
    Externa x = new Externa();
    Externa. Interna a = x.new Interna();
}
```



Error  
Estamos en distinto paquete

26

## Clases locales

- Clases que se definen dentro de un bloque de código: la implementación de un método, un constructor...
- **Visibles sólo dentro del bloque**
- No se puede especificar accesibilidad (son locales al bloque) y sólo pueden ser `static` en bloques estáticos
- No pueden tener miembros estáticos
- No se pueden crear instancias de ellas fuera del bloque que las define

27

## Clases locales

- Guardan una referencia oculta a la instancia de la clase externa
- Tienen acceso a los parámetros y variables locales `final` del bloque
- No se pueden usar fuera del bloque por lo que no tiene sentido darlas nombre (no se puede usar `Clase.ClaseLocal`)
- Este tipo de clases suele utilizarse junto con la herencia:
  - La clase local implementa un interfaz o extiende una clase que sí es accesible desde fuera y el bloque devuelve un objeto de esa clase padre o interfaz.

28

```

public class OuterClass {
    private int k1,k2;

    public void m(int x,int y){
        final int k = 5;
        int h = 12;

        class Point {
            private int xx; yy;

            Point(int a, int b){ this.xx = a + k1; this.yy = b + k2; }

            Public Point p(){
                this.xx = x + k; // acceso a los parámetros de m y a la constante k
                this.yy = y + h; // acceso a los parámetros de m y a la variable h
                h = 12; // mal. No se puede modificar h en Point
                return new Point(this.xx, this.yy);
            }
        }
        Point p = new Point(3,4); // en m se crea un objeto de la clase local
    }
}

```

29

## Clases locales

```

public class Shape {
    void draw() {
        System.out.println("Dibuja una forma");
    }
}

```

```

public class Main {

    public static void main ( String args []) {
        Vector <Shape> v = new Vector <Shape>();
        for ( Shape d : v)
            d.draw();
    }
}

```

30

## Clases locales

- Podemos tener una clase **Pintor** que cree objetos que extienden la clase **Shape** como un círculo de radio *r* o un mapa.
- En vez de implementar esas clases de formas en ficheros independientes o clases internas, las podemos hacer locales a los métodos respectivos:

```
public class Pintor {
    public Shape crearMapa() {
        class Map extends Shape {
            void draw() {
                System.out.println("Dibuja un mapa");
            }
        }
        return new Map();
    }
    public Shape crearCirculo(final float radio) {
        class Circle extends Shape {
            void draw() {
                System.out.println("Dibuja un círculo de radio" + radio);
            }
        }
        return new Circle();
    }
}
```

Acceso a los parámetros  
y variables locales  
final del bloque

31

## Clases locales

- Se podrían haber utilizado clases internas (no locales).

Pero ...

- Existe una característica de las clases locales que *no* está disponible en las clases internas no estáticas:
  - Las implementaciones de los métodos de las clases locales pueden acceder a parámetros y variables locales del bloque declaradas como final.
  - En el ejemplo anterior tenemos un método que crea un círculo cuyo radio se pasa como parámetro al propio método.
  - La implementación del método `draw` accede al valor de ese parámetro.
  - Eso significa que cuando el usuario llame al método `draw`, éste accederá a un valor cuyo tiempo de vida estaba en principio limitada al momento de la invocación al método de creación del círculo.

32



## Clases anónimas ( ... la última)

- **Clases sin nombre** que se definen e instancian a la vez
  - Sólo se puede crear una instancia de la clase
- Usan sintaxis especial del operador **new**
- Se usan para crear objetos “al vuelo” para devolver valores, pasar argumentos o inicializar atributos
- Las clases anónimas no pueden tener constructores (al fin y al cabo no tienen nombre...),
- **Siempre extenderán de otra clase o implementarán un interfaz**, de forma que la definición sobrescribe o implementa esos métodos que serán los que puedan llamarse desde fuera.
- Se suelen emplear para el manejo de eventos

33

## Clases anónimas

- No se puede especificar accesibilidad (son locales al bloque)
- Sólo pueden ser `static` en bloques estáticos
- No pueden tener miembros estáticos
- Guardan una referencia oculta a la instancia de la clase externa
- Tienen acceso a los parámetros y variables locales `final` del bloque
- No se pueden usar fuera del bloque que las define por lo que no tiene sentido darlas nombre (no se puede usar `Clase.ClaseLocal`)

34

## Clases anónimas

```
public class Pintor {
    public Shape crearMapa() {
        class Map extends Shape {
            void draw() {
                System.out.println("Dibuja un mapa");
            }
        }
        return new Map();
    }
    public Shape crearCirculo(final float radio) {
        class C {
            void draw() {
                System.out.println("Dibuja un círculo de radio " + radio);
            }
        }
        return new C();
    }
}

public class Pintor{
    public static Shape crearMapaBis() {
        return new Shape(){
            public void draw(){
                System.out.println(" Dibujo de un mapa ");
            }
        };
    }
    public Shape crearCirculoBis(final float radio) {
        return new Shape() {
            public void draw() {
                System.out.println("Dibuja un círculo de radio" + radio);
            }
        };
    }
}
```

35

## Clases anónimas

```
public class ButtonExample implements ActionListener {
    private JButton redButton;

    public JPanel createContentPane() {
        redButton = new JButton("Red Score!");
        redButton.addActionListener(this);
        ...
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==redButton) ...
    }
}
```

Quando se hace "click" sobre el botón se ejecuta el método `actionPerformed` del objeto `this`.

- `actionPerformed` es un método de `ActionListener`

36

## Clases anónimas

```

public class ButtonExample {
    private JButton redButton;
    public JPanel createContentPane() {
        redButton = new JButton("Red Score!");
        redButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) ...
        });
        ...
    }
}

```

Usando clase anónima

37

## Ejemplo.

```

public class A {
    public void f() { g();}
    public void g() {
        System.out.println("Hola");
    }
}

public static void main(String[] args) {
    A x = new A() {
        @Override
        public void g() {
            System.out.println("hola hola");
        }
    };
    x.f();
}

```

Una clase anónima que  
sobreescribe el método g  
de la clase A

38