

Clase 3-clases anidadas e internas

miércoles, 3 de septiembre de 2025 14:46

Clases adentro de clases, se usan cuando una clase es solo útil adentro de otra clase.

Si sirve en otro contexto, no tiene sentido y conviene sacarla afuera.

La clase anidada tiene acceso a todo el objeto que la contiene

La clase contenedora es quien hace el new de la clase.

La clase anidada es como una variable anidada.

Un objeto de una clase anidada tiene una referencia implícita al objeto de la clase que lo instanció

(clase contenedora). A través de dicha referencia tiene acceso al estado completo del objeto

contenedor, inclusive a sus datos privados. Por lo tanto las clases anidadas tienen más privilegios de acceso que las de nivel superior.

Las anidadas me permiten ocultar aún jmás cosas de implementación.

Las clases anidadas pueden ser private o package.

Definiendo a la clase con acceso de default o package: la clase es visible solamente dentro del paquete donde se declaró.

¿Para qué usamos clases anidadas?

```
public interface Contenido {  
    int valor();  
}  
public interface Destino {  
    String leerEtiqueta();  
}
```

```
package turismo;  
public class Viaje {  
    public static void main(String[] args) {  
        Paquete p = new Paquete();  
        Contenido c = p.cont();  
        Destino d = p.hacia("Buenos Aires");  
        System.out.println(d.leerEtiqueta());  
        System.out.println(c.valor());  
    }  
}
```

Clase anidada privada: es accesible solamente desde la clase Paquete

Para proveer **ocultamiento de detalles de implementación**

Se obtiene una referencia, solamente a través **Upcasting/Generalización** a una clase base o interface, públicas.

```
public class Paquete {  
    private class PContenido implements Contenido{  
        private int i=11;  
        public int valor() {return i;}  
    }  
    private class PDestino implements Destino{  
        private String etiqueta;  
        private PDestino(String donde){  
            etiqueta=donde;  
        }  
        public String leerEtiqueta() {  
            return etiqueta;  
        }  
    }  
    public Destino hacia(String s) {  
        return new PDestino(s);  
    }  
    public Contenido cont() {  
        return new PContenido();  
    }  
} // Fin de la clase Paquete
```

Upcasting al tipo de la interface

Las **clases anidadas privadas que implementan interfaces** son completamente invisibles e inaccesibles y de esta manera se **oculta la implementación**. Se evitan dependencias de tipos. Desde afuera de la clase **Paquete** se obtiene una **referencia al tipo de la interface pública**.

Laboratorio de Software – Claudia Queiruga

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.



es como que me permite tener las clases y la implementación en privado y después tener interfaces públicas con las que otras clases usan lo que hice.

Clase local: clase adentro de método.

Es como una variable local. Vuelve a ser para ocultar implementación, se usa si solo un método usa algo.

Ej si la defino en un if solo puedo acceder ahí.

Clases anonimas: crear una clase sin nombre que implementa

Las **clases anónimas** son **clases locales** sin nombre. Se crean extendiendo una clase o implementando una interface. **Combinan** la sintaxis de **definición de clases** con la de **instanciación de objetos**. Las interfaces y los tipos enumerativos NO pueden definirse anónimamente.

El método `cont()` combina la creación del valor de retorno con la definición de la clase que representa el valor retornado.

```
public class Paquete {
    public Contenido cont() {
        return new Contenido() {
            private int i=11;
            public int valor(){
                return i;
            }
        };
    }
} // Fin de la clase Paquete
```

Es una abreviatura de la declaración de una clase que implementa la interface **Contenido**

```
package turismo;
public class Viaje {
    public static void main(String[] args) {
        Paquete p = new Paquete();
        Contenido c = p.cont();
    }
}
```

```
public interface Contenido {
    int valor();
}
```

```
public Contenido cont() {
    class PContenido implements Contenido {
        private int i=11;
        public int valor(){ return i;}
    }
    return new PContenido();
}
```

Este obra está bajo una Licencia Creative Commons Atribución NoComercial CompartirIgual 4.0 Internacional



🔴 Ejemplo normal (clase con nombre)

Supongamos que tengo una interfaz:

java

Copy code

```
public interface Contenido {
    int valor();
}
```

Una forma de implementarla sería:

java

Copy code

```
class MiContenido implements Contenido {
    private int i = 11;
    public int valor() { return i; }
}
```

Y la uso así:

java

Copy code

```
Contenido c = new MiContenido();
```

✦ Lo mismo con clase anónima

En lugar de definir `MiContenido` como clase aparte, hago todo en una sola línea:

java

Copy code

```
public Contenido cont() {  
    return new Contenido() {  
        private int i = 11;  
        public int valor() { return i; }  
    };  
}
```

Fijate:

- No le puse nombre a la clase.
- Directamente digo `new Contenido() { ... }` e implemento los métodos adentro.
- El objeto creado se **upcastea** automáticamente a `Contenido`.

Cuando llamo:

java

Copy code

```
Contenido c = cont();  
System.out.println(c.valor()); // imprime 11
```

Funciona igual que con una clase con nombre, pero ↓ declararla aparte.

No entendí nada de anónimas

Si a una clase anonima le paso como parámetro en un método una variable queda como final, no la puedo cambiar el valor

Las clases anónimas no tienen constructores pero puedo definir bloque de inicialización entre corchetes que se ejecutab cuando hago new.