3. Clases anidadas

Date	Empty
Status	Not started
Type	Empty
Units	7. Uso avanzado de clases

Clases anidadas

- Clases anidadas no estáticas o clases internas
- Clases anidadas estáticas

Clases anidadas

Las clases anidadas en java se pueden definir cómo la creación de una clase que está dentro de otra. Por regla general, cada archivo representa una clase individual, pero como ya vimos en el tema de los Enum también es posible unos dentro de otros. El uso de este tipo de clases es el siguiente:

- Permite agrupar lógicamente clases que solo se utilizan en un lugar, sin la necesidad de tener que crear gran cantidad de archivos
- Marca una relación de dependencia, ya que la clase anidada no puede existen sin la existencia de la clase principal (a no ser que la

clase anidada sea estática)

- El concepto de encapsulación se sigue manteniendo ya que una clase anidada puede acceder a todos los elementos de la clase en la que está declarada, aunque estos sean estáticos

Como regla general existen dos tipos de clases anidadas: estáticas y no estáticas o también llamadas internas

Clases anidadas no estáticas o clases internas

Se define una clase interna como aquella que está definida dentro de una clase y cumple los siguientes requisitos:

- Puede tener acceso a todos los miembros de la clase a la que pertenece (incluso los declarados como privados)
- Solo puede crearse un objeto de la clase interna si previamente existe un objeto de la clase donde es anidada

Para poder verlo mejor vamos hacer un ejemplo:

Imaginad la siguiente clase

```
public class ClaseExterna { private String dato; private int
numero; public ClaseExterna(String dato, int numero) { this.d
ato = dato; this.numero = numero; } public void mostrarDatos
(){ System.out.printf(String.format("%d %s %n",getNumero(),ge
tDato())); } public String getDato() { return dato; } public
void setDato(String dato) { this.dato = dato; } public int ge
tNumero() { return numero; } public void setNumero(int numer
o) { this.numero = numero; } }
```

Esta clase está representada en un archivo .java y tienen un constructor, getter y setter, y un método mostrarDatos(). Es importante fijarse en que las variables que están definidas en la clase son privadas lo que quiere decir que nadie podrá acceder a ellas, ni siquiera un objeto que está definido desde fuera (para eso están los getter // setter - concepto de encapsulación)

```
public class Entrada { public static void main(String[] args)
{ ClaseExterna claseExterna = new ClaseExterna("dato extern
o",1); claseExterna.mostrarDatos(); } }
```

En otro archivo .java queda representado el método main para poder acceder a todos aquellos elementos que no sean privados, como por ejemplo el método mostrarDatos.

Hasta aquí el uso corriente. En algunas ocasiones es interesante que la ClaseExterna tenga asociada otra clase que represente un objeto propio, y como no se va a utilizar en ningún otro sitio se puede definir dentro (concepto de clase anidada o interna). Para ello en el mismo archivo .java donde se ha definido la clase ClaseExterna se crea otra clase adicional con la palabra class seguida del nombre (cuidado con los paréntesis, ya que la clase que se está creando está dentro)

```
public class ClaseExterna { private String dato; private int
numero; public ClaseExterna(String dato, int numero) { this.d
ato = dato; this.numero = numero; } public void mostrarDatos
(){ System.out.printf(String.format("%d %s %n",getNumero(),ge
tDato())); } public String getDato() { return dato; } public
void setDato(String dato) { this.dato = dato; } public int ge
tNumero() { return numero; } public void setNumero(int numer
o) { this.numero = numero; } class ClaseInterna{ // definició
n de todos los elementos de la clase interna } }
```

De esta forma se acaba de crear una clase interna, pudiendo declarar tanto variables como métodos además de acceder a todas las partes de la clase donde se ha definido (en este caso la ClaseExterna)

```
public class ClaseExterna { private String dato; private int
numero; public ClaseExterna(String dato, int numero) { this.d
ato = dato; this.numero = numero; } public void mostrarDatos
(){ System.out.printf(String.format("%d %s %n",getNumero(),ge
tDato())); } public String getDato() { return dato; } public
void setDato(String dato) { this.dato = dato; } public int ge
tNumero() { return numero; } public void setNumero(int numer
o) { this.numero = numero; } class ClaseInterna{ private int
numeroInterno; private String datoInterno; public ClaseIntern
a(int numeroInterno, String datoInterno) { this.numeroInterno
= numeroInterno + numero; this.datoInterno = dato.concat(dato
Interno); } public void mostrarDatosInternos(){ System.out.pr
intf(String.format("%d %s %n",getNumeroInterno(),getDatoInter
no())); } public int getNumeroInterno() { return numeroIntern
o; } public void setNumeroInterno(int numeroInterno) { this.n
umeroInterno = numeroInterno; } public String getDatoInterno
() { return datoInterno; } public void setDatoInterno(String
datoInterno) { this.datoInterno = datoInterno; } } }
```

```
datoInterno) { this.datoInterno = datoInterno; } } }
```

En la definición de la clase interna se crean tantos elementos como sean necesarios, pudiendo acceder a los elementos de la clase superior. En el ejemplo, para poder crear una clase ClaseInterna se pide por constructor un String y un int, los cuales son añadidos a los anteriores, tal y como se puede ver en el constructor.

Una vez hecho esto se puede utilizar un objeto de ClaseInterna, siempre y cuando ya existe un objeto de la clase donde está declarada. Para ello en el la clase Entrada si se intenta crear un objeto de la forma "normal" da error:

```
// ERROR ClaseExterna.ClaseInterna claseInterna = new ClaseIn
terna(1,"asd");
```

Cuando se trabaja con clases internas, es necesario la existencia previa de un objeto de la clase que lo contiene, sino es imposible crearlas

```
// CORRECTO ClaseExterna.ClaseInterna claseInterna = claseExt
erna.new ClaseInterna(1,"dato interno"); claseInterna.mostrar
DatosInternos();
```

El ejemplo completo sería

```
public class Entrada { public static void main(String[] args)
{ ClaseExterna claseExterna = new ClaseExterna("dato extern
o",1); claseExterna.mostrarDatos(); ClaseExterna.ClaseInterna
claseInterna = claseExterna.new ClaseInterna(1,"dato intern
o"); claseInterna.mostrarDatosInternos(); } }
```

Y la salida

```
1 dato externo 2 dato externo dato interno
```

Clases anidadas estáticas

El uso de clases anidadas estáticas es muy parecido a las vistas con

El uso de clases anidadas estáticas es muy parecido a las vistas con anterioridad, con las siguientes diferencias:

- Se utiliza el modificador static en las clases anidadas para poder crearlas
- Una clase anidada estática no puede acceder a los métodos y/o variables de su clase contenedora a no ser que estos estén declarados como estáticos
- Una clase anidada estática puede ser utilizada de forma directa, por lo que se pueden crear objetos de su tipo sin necesidad de la existencia de un objeto de la clase contenedora.

Basándonos en el ejemplo anterior tendríamos la clase ClaseExternaExt, definida en un archivo .java

```
public class ClaseExternaExt { private String dato; private s
tatic int numero; public ClaseExternaExt(String dato, int num
ero) { this.dato = dato; this.numero = numero; } public void
mostrarDatos(){ System.out.printf(String.format("%d %s %n",ge
tNumero(),getDato())); } public String getDato() { return dat
o; } public void setDato(String dato) { this.dato = dato; } p
ublic int getNumero() { return numero; } public void setNumer
o(int numero) { this.numero = numero; } }
```

Dentro de este mismo archivo se puede definir una clase anidada estática con la palabra static, pero que no puede acceder a ninguno de los método ni variables definidas en la clase ClaseExternaExt

```
public class ClaseExternaExt { private String dato; private i
nt numero; private static boolean datoAccesible; public Clase
ExternaExt(String dato, int numero) { this.dato = dato; this.
numero = numero; if (numero>0){ datoAccesible = true; } else
{ datoAccesible = false; } } public String getDato() { return
dato; } public void setDato(String dato) { this.dato = dato;
} public int getNumero() { return numero; } public void setNu
mero(int numero) { this.numero = numero; } static class Clase
InternaExt{ private int numeroInterno; private String datoInt
erno; public ClaseInternaExt(int numeroInterno, String datoIn
terno) { if (datoAccesible==true){ this.numeroInterno = numer
oInterno; this.datoInterno = datoInterno;} System.out.println
("el objeto se creará con los parámetros por defecto"); } pub
lic int getNumeroInterno() { return numeroInterno; } public S
tring getDatoInterno() { return datoInterno; } } }
```

La modificación con el ejemplo anterior es que la clase contenedora tiene un atributo de tipo booleano que se declara como static. Este atributo es el único que será accesible por la clase estática. Como se puede ver, el atributo datoAccesible se utiliza en el constructor. Si bien esta posibilidad existe no es muy recomendable ya que como se verá mas adelante es peligroso ya que se puede crear un objeto de la clase anidada sin necesidad de un objeto de la clase contenedora (por lo que podría dar un nulo en aquellos datos que son utilizados). Por lo tanto para el ejemplo se quitarán:

```
public class ClaseExternaExt { private String dato; private i
nt numero; public ClaseExternaExt(String dato, int numero) {
this.dato = dato; this.numero = numero; } public String getDa
to() { return dato; } public void setDato(String dato) { thi
s.dato = dato; } public int getNumero() { return numero; } pu
blic void setNumero(int numero) { this.numero = numero; } sta
tic class ClaseInternaExt{ private int numeroInterno; private
String datoInterno; public ClaseInternaExt(int numeroInterno,
String datoInterno) { this.numeroInterno = numeroInterno; thi
s.datoInterno = datoInterno; } public void mostrarDatoInterno
(){ System.out.printf(String.format("%d %s %n",getNumeroInter
no(),getDatoInterno())); } public int getNumeroInterno() { re
turn numeroInterno; } public String getDatoInterno() { return
datoInterno; } } }
```

Con esto terminado, en la entrada se puede ver como existe la posibilidad de crear un objeto de la clase anidada (y acceso a todos sus métodos) sin necesidad de tener un objeto de la clase contenedora

```
package github; public class Entrada { public static void mai
n(String[] args) { ClaseExternaExt.ClaseInternaExt claseInter
naExt = new ClaseExternaExt.ClaseInternaExt(1,"dato intern
o"); claseInternaExt.mostrarDatoInterno(); } }
```

Y la salida

```
1 dato interno
```