







## 2. Clases anónimas

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

### Clases anónimas

Como su nombre indica, una clase anónima es aquella que no tiene nombre y cuyo uso es directo, sin necesidad de asignarlo a una variable cualquiera. El uso de este tipo de clase esta dedicado a aquellas operaciones que necesitan de un objeto concreto pero no está justificado su creación, de forma que con su simple uso es suficiente. Esto ahorra tanto memoria como trabajo al programa.

Lo normal es que esto se utilice mediante objetos de tipo interfaz, donde tan solo se define la firma de un método y en las clases anónimas se crea el cuerpo, sin necesidad de implementar la interfaz. Esto no quiere decir que sea uso exclusivo, ya que también se puede hacer mediante clases "normales"

### Implementación mediante clases

Imaginad la siguiente clase

```
public abstract class ClaseA { public void mostrarMensaje(){ System.out.println("mensaje lanzado desde la clase A"); } public abstract void mostrarMensajeAbstracto(); }
```

En este caso la clase es abstracta porque tiene un método que no tiene cuerpo, por lo que su uso está restringido a que la clase que extienda de ClaseA lo implemente. Ese sería su uso normal como se muestra en la siguiente clase:

```
public class ClaseB extends ClaseA { @Override public void mostrarMensajeAbstracto() { System.out.println("Mensaje lanzado desde la clase b"); } }
```

Sin embargo con el uso de clases anónimas esta extensión e implementación no es necesario. Se pueden crear objetos de tipo ClaseA de forma directa, con la única restricción que los métodos abstractos deben ser implementados. Imaginad la siguiente clase

```
public class ClaseC { public void registrarMensajeElemento(ClaseA clase){ clase.mostrarMensaje(); clase.mostrarMensajeAbstracto(); } }
```

Esta clase tiene un método que obtiene como parámetro un objeto de ClaseA, el cual llama a los métodos mostrarMensaje y mostrarMensajeAbstracto(). De estos dos métodos, el primero tiene definición y hará un `System.out.println("mensaje lanzado desde clase a")`, sin embargo el método mostrarMensajeAbstracto no tiene definición por lo que aún no se sabe como se comportará. En la clase Entrada, se puede utilizar directamente mediante una clase anónima

```
public class Entrada { public static void main(String[] args) { ClaseC claseC = new ClaseC(); claseC.registrarMensajeElemento(new ClaseA() { @Override public void mostrarMensajeAbstracto() { } }); } }
```

El parámetro que admite el método registrarMensajeElemento es un

objeto de ClaseA, que a su vez como es abstracta me obliga a escribir el cuerpo de todos aquellos métodos que son abstractos, como lo es el método `mostrarMensajeAbstracto()`. Si se escribe el cuerpo:

```
public class Entrada { public static void main(String[] args)
{ ClaseC claseC = new ClaseC(); claseC.registrarMensajeElemen
to(new ClaseA() { @Override public void mostrarMensajeAbstrac
to() { System.out.println("mensaje lanzado desde la clase mai
n"); } }); } }
```

En la ejecución del main sería

```
mensaje lanzado desde la clase A mensaje lanzado desde la cla
se main
```

La primera línea se ejecuta desde el método `mostrarMensaje` del objeto `ClaseA`, ya que está escrito en la propia clase y llamados desde el método `registrarMensajeElemento` de la `ClaseC`

La segunda línea se ejecuta desde el método `mostrarMensajeAbstracto` del objeto anónimo creado en la propia clase `main`, ya que ahí ha sido escrito el cuerpo

## Implementación mediante interfaces

Se realiza de la misma forma que lo anterior, con la única diferencia que todos los métodos están definidos en la interfaz y por lo tanto tendrán que ser escritos en objeto definido con la clase anónima.

Imaginad la siguiente interfaz:

```
public interface InterfazMensajes { void lanzarMensajeMinuscu
las(); void lanzarMensajeMayusculas(); }
```

Esta interfaz representa la funcionalidad general que necesitan objetos de la clase, pero no es necesario que los objetos que utilizan estos métodos implementen la interfaz (porque no interesa para no "ensuciar la clase", por temas de funcionalidad, porque no se quiere que un objeto tenga varias formas, etc...)

```
public interface InterfazMensajes { void lanzarMensajeMinuscu
las(); void lanzarMensajeMayusculas(); }
```

Al ser métodos en una interfaz, no tienen cuerpo que deberá ser definido donde se utilicen (en las clases que los implementen o en las clases anónimas que los llamen). Para poder utilizar la interfaz se crea una segunda clase que llama a estos métodos

```
public class Mensajes { public void lanzarMensajes(InterfazMe
nsajes interfaz){ interfaz.lanzarMensajeMayusculas(); interfa
z.lanzarMensajeMinusculas(); } }
```

Al no haber implementado la interfaz no se define el comportamiento de los métodos, tan solo se llama a los mismo para que sean ejecutados. El funcionamiento de los métodos será definido cuando se cree un objeto de tipo interfaz. En la clase entrada es donde se juntan las funcionalidades

```
public class Entrada { public static void main(String[] args)
{ Mensajes mensajes = new Mensajes(); String m = "Mensaje eje
mplo para Lanzarlo desde CLASE anónima"; mensajes.lanzarMensa
jes(new InterfazMensajes() { @Override public void lanzarMens
ajeMinusculas() { System.out.println(m.toLowerCase()); } @Ove
rride public void lanzarMensajeMayusculas() { System.out.prin
tln(m.toUpperCase()); } }); }
```

Al crear un objeto de tipo Mensajes y llamar al método lanzarMensajes espera un objeto de tipo InterfazMensaje (tal cual se definió en la clase). Es en este punto donde se crea el objeto, que al ser de tipo interfaz obliga a definir la funcionalidad de los métodos. La salida del main será:

```
MENSAJE EJEMPLO PARA LANZARLO DESDE CLASE ANÓNIMA mensaje eje
mplo para lanzarlo desde clase anónima
```

Ambas líneas son llamadas desde el método lanzarMensajes de la clase Mensajes, pero su definición está escrita en el main cuando se crea el objeto de tipo interfaz

crea el objeto de tipo `Interfaz`.

Esta forma de trabajar es muy utilizada en los escuchadores de las librerías gráficas de java como `awt`, `swing` y `javafx`, algunas de las cuales se verán al final del curso y principios del año que viene

[https://img.freepik.com/premium-photo/programming-code-abstract-technology-background-software-developer-computer-script\\_34663-31.jpg](https://img.freepik.com/premium-photo/programming-code-abstract-technology-background-software-developer-computer-script_34663-31.jpg)