CLASES ABSTRACTAS:

Recuerda si tenemos una clase Final no puede ser extendida, no puedo hacer una clase que extienda a la que es final.

Por ejemplo, si tengo esta clase:

```
public final class ClaseFinal {


}
```

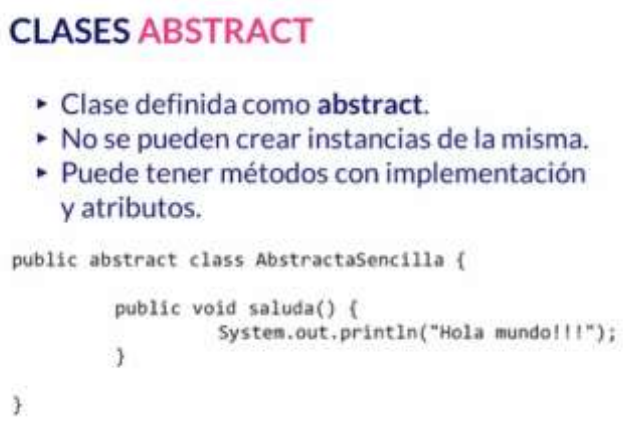
No puedo hacer esto:

```
2
3 public class ClaseExtendida extends ClaseFinal {
4
5 }*/
6
```

Esto da un error porque intento crear una clase que extiende de una que es final.

CLASE ABSTRACTA:

Una clase abstracta, sirven de plantilla para que las clases extiendan de ellas pero no puedo crear objetos de las clases abstractas. Por ejemplo, de nuestro ejercicio de vehículos, si fuera vehículo una clase abstracta, no puedo crear objetos de vehículo, sí de furgoneta, o deportivo… de las clases que extienden de la clase abstracta, sí creo objetos.



Pueden tener métodos que tengan implementación, como este saluda() y también pueden tener métodos abstractos, no tienen código.

MÉTODOS ABSTRACT

- Deben estar en una clase definida como **abstract**.
- Definen la firma del método, pero sin implementación.
- Sus subclases se comprometen a implementarlo.
- Si no lo hacen, también deben ser abstractas.
- Pueden convivir con métodos normales.

```
public abstract class AbstractaConMetodos {

        public abstract void saludo(String s);

        public void saludar() {
                System.out.println("Hola mundo!!!");
        }
}
```

Cualquier clase que herede de una abstracta tiene que implementar a la fuerza todos los métodos abstractos que tengan, a no ser que también sea una clase abstracta.

En resumen, una clase abstracta es una clase de la que no se pueden instanciar objetos, puede tener atributos y métodos (con implementación y/o abstractos) y cualquier clase que herede de ella, sino es abstracta también, debe implementar los métodos abstractos de su clase super.

Ejemplo, tenemos una clase abstracta que tiene un método abstracto y otro implementado:

```
public abstract class AbstractaConMetodos {

    public abstract void saludo(String s);

    public void saludar() {
        System.out.println("Hola mundo!!!");
    }

}
```

Tenemos otra clase que extiende de la anterior, a LA FUERZA TENEMOS QUE IMPLEMENTAR EL MÉTODO SALUDO, sino eclipse nos da error:

```
2
3 public class DerivadaConMetodos extends AbstractaConMetodos {
4
5     @Override
6     public void saludo(String s) {
7         System.out.println("Hola " + s);
8     }
9
0 }
1
```

En mi clase principal, puedo probar a instanciar y los métodos:

```
public class EjemploAbstractaConMetodos {

    /**
     * @param args
     */
    public static void main(String[] args) {

        DerivadaConMetodos derivada = new DerivadaConMetodos();
        derivada.saludar();
        derivada.saludo("Pepe");

    }

}
```

¿Se usan las clases abstractas desde que aparecieron las interfaces? Sí, cuando usar una u otra depende de lo que queramos implementar en cada momento, puede servir esta información para aclararnos, sobre todo cuando ya hayamos estudiado las interfaces, recuerda repasar esto, después de estudiarlas:

INTERFACES	CLASES ABSTRACTAS
No se pueden instanciar	No se pueden instanciar
Métodos sin implementación	Métodos sin implementación
Métodos con implementación por defecto	Métodos con implementación por defecto
Atributos estáticos o constantes	Cualquier tipo de atributos
Métodos públicos o por defecto	Métodos públicos, privados, protegidos o por defecto.
Una clase puede implementar varios interfaces	Una clase solo puede heredar de otra

¿QUÉ USAR?

INTERFACES	CLASES ABSTRACTAS
Clases no relacionadas podrán implementar los métodos.	Compartir código con clases muy relacionadas.
Si se quiere indicar que existe un tipo de comportamiento, pero no sabemos quien lo implementa.	Las clases derivadas usarán métodos *protected* o *private*.
Si necesitamos tener *herencia múltiple*.	Queremos definir atributos que no sean estáticos o constantes.

Si una clase extiende ya de otra, no podría extender además de una abstracta, sería un caso claro de necesitar interfaces. Se usa clase abstracta cuando las diferentes clases que heredan de ella tienen una clara relación, como los ejercicios que hemos hecho hasta ahora, si tenemos claro que no queremos objetos de la clase padre.