La herencia en Kotlin y construcción de objetos

85.- Vamos a crear una clase llamada *Persona* y vamos a trasladar ahí todas las propiedades y funciones de nuestra clase *Alumno* que sean propiedades de cualquier persona, aunque no fuera estudiante.

86.- Hacer que *Alumno* herede de *Persona*

87.- Crear una función en *Persona* que muestre por pantalla la dirección de esa persona

88.- Crear un objeto de tipo *Alumno* y usa el método que muestra la dirección por pantalla.

Sobreescritura

89.- Como no queremos que se vea dañada la protección de datos de nuestros alumnos y alumnas queremos sobreescribir la función que muestra su dirección y vamos a hacer que en su lugar muestre el mensaje *"Esta es una información sensible"*

Clases abstractas

90.- Crear la clase abstracta Asignatura, esta tendrá dos propiedades: *profesor*, de tipo *Persona* y *aula* de tipo *String*.

91.- Añadir a la clase *Asignatura* una función abstracta, *mostrarHorario*, que muestre por pantalla el horario de la asignatura

92.- Añadir a la clase *Asignatura* una función con implementación por defecto llamado *mostrarInformacion* que muestre por pantalla *"Esta asignatura se imparte en el aula x por parte de y"* siendo x el aula e y el nombre del profesor.

93.- Crear una clase llamada *Matematicas* que extienda *Asignatura* e implementar su función abstracta.

94.- Crear un objeto de tipo *Matematicas* y llamar a sus dos funciones

Interfaces

95.- Crear la interfaz Contactar con dos funciones, la función *llamar* sin ningún parámetro de entrada y la función *escribirCarta* a la que se le pasará un *String* con el texto de la carta.

96.- Hacer que *Persona* implemente Contactar e implementar sus métodos de manera figurada añadiendo un comentario (No podemos hacer que nuestro programa haga una llamada, si fuera una aplicación Android si que podríamos)

97.- Llamar a alguno de los anteriores métodos en un objeto de tipo *Alumno*.

Paquetes

98.- En nuestro proyecto, crear dos paquetes, uno llamado *personas* y otro llamado *asignaturas* y mover a estos las clases o interfaces correspondientes

Modificadores de visibilidad

99.- Decidir qué modificador de visibilidad usar para cada una de nuestras clases, funciones y propiedades