Variables que pueden ser null

100.- Piensa alguna propiedad de nuestro proyecto que es posible que exista o que no exista y declárala como posible null. Corrige si hay algún problema derivado de ello.

El operador elvis

101.- ¿Qué mostrará este código por pantalla?. Piénsalo y luego escríbelo para comprobarlo.

```
fun main(args: Array<String>) {
    val x: Int? = 2
    val y: Int = 3
    val sum = x?:0 + y
    println(sum)
}
```

Operador !!

102.- Provoca una NullPointerException

Comprobación y conversión de tipos

103.- Crea una función que acepte como argumento una lista con elementos de tipo Any. Haz que la función recorra los elementos de la lista y si son de tipo String, muéstralos por pantalla en mayúsculas, si son de tipo Int, transformalos en String y muéstralos por pantalla en mayúsculas

Conversión de tipos explícita

104.- Crea una función que acepte como argumento una lista con elementos de tipo Any. A cada uno de ellos conviértelos explicitamente en int y muestra el resultado multiplicado por 5. ¿Qué ocurre si la lista

contiene elementos que no son Int? ¿Como podemos asegurar que no se produce una excepción?

Let

105.- Declara una variable que puede ser null. Con la ayuda de *let*, muestra por pantalla el mensaje: *"¡No es null, es x!"* solo en caso de que la variable no sea null. Siendo x el valor de la variable

106.- Declara otra variable que pueda ser null. Con la ayuda de *let*, muestra por pantalla el mensaje *"¡No son null, son x y y!"* solo en el caso de que ambas variables no sean null. Siendo x el valor de la primera variable e y el de la siguiente.

107.- En el primer escenario, con una única variable y también haciendo uso de *let*, muestra por pantalla el mensaje *"¡Es null!"* en el caso de que la variable lo sea.

With

108.- Con la ayuda de *with*, crea o mejora el método que muestra por pantalla el objeto *Direccion* de una Persona

Run

109.- Haciendo uso de run, declara una variable de tipo *String?*. Solo en caso de que no sea null, muestra por pantalla su número de caracteres.

Apply

110.- Declara una clase con 4 propiedades y un método. Las propiedades deben ser declaradas fuera del constructor. Crea un objeto de esa clase e inicializa sus propiedades mediante *apply*. Por último, llama al método de ese objeto.

Also

111.- Crea un objeto de la clase anterior. Inicializa sus propiedades mediante *apply* y usa *also* para mostrar alguna de ellas por pantalla.

TakeIf y takeUnless

112.- Crea una lista de objetos *Alumno*. Recorre la lista y para cada uno llama a *escribirCarta* con el texto: *"Tienes asignaturas que recuperar"* solo si tiene alguna asignatura suspendida. Hazlo de dos maneras diferentes con *takeIf* y con *takeUnless*

Objects

113.- Declara un *object* llamado *Calculadora* con unas constantes y métodos que puedan ser útiles en los cálculos de tu programa.

Companion object

114.- Crea un companion object para la clase Alumno y en él crea un método factoría llamado crear. Implementa este como ayuda para crear tus objetos de tipo Alumno y úsalo.

Inicialización tardía

115.- Crea una clase con una propiedad marcada como lateinit. Crea una función miembro de la clase llamada inicializar que inicialice dicha propiedad. Desde el método main crea un objeto de tu clase y accede a la propiedad ¿Qué ocurre? Ahora antes de acceder a la propiedad, llama al método lateinit. ¿Qué ocurre ahora?