Ejercicios Clases Kotlin

Descargar estos ejercicios

Ejercicio 1

Crearemos una interface llamada **Electrodomestico** con las siguientes características:

Tendrá una **enumeración Color** con los colores `[BLANCO, NEGRO, ROJO, AZUL, GRIS]`
Una clase de tipo excepción **LetraConsumoInvalidaException** con el constructor del mensaje.

Sus propiedades son:

- propiedad abstracta **consumo** energético de tipo char y que solo podrá contener letras entre A y F (esto se resolverá a través de un método de interface llamado compruebaConsumo, explicado posteriormente)
- propiedad regular **precio** de tipo float inmutable y calculada. Devolverá mediante un when de expresión el precio del electrodoméstico, de forma que dependerá del consumo y el peso (consumo + peso * 2). Donde el consumo se mapeará de forma (consumo->precio) con la siguiente correspondencia `[A->100, B->80, C->60, E->30, otros->10]`
- propiedad regular inmutable **color** de tipo enumerado Color y que devolverá por defecto el color BLANCO
- propiedad abstracta **peso** de tipo Float.

Además tendrá una función regular **compruebaConsumo** a la que le llegará un Char y comprobará que está entre las letras A y F (teniendo en cuenta las mayúsculas), devolverá la letra del consumo en mayúsculas si es correcta y una excepción del tipo `LetraConsumoInvalidaException` en otro caso.

Además tendremos una **clase Lavadora** que implementará la interface de forma que:

- El consumo seguirá siendo inmutable, pero al ser abstracto en la interfaz ahora tendremos que implementar su get, se asignará en el constructor `init{}` después de comprobar que es correcto con el método regular de la interfaz *compruebaConsumo*.
- El peso también deberá de implementarse, pero será de acceso y modificación.
- Podemos definir la nueva propiedad carga, como mutable dentro del propio constructor por defecto.
- El color se asignará en el segundo constructor una vez se haya comprobado que es correcta la cadena de entrada con el tipo Color. Para indicar que se inicializará posteriormente, la señalaremos como `lateinit`. Además para que este modificador no nos de excepción por no inicialialización, deberemos asignar la propiedad en el constructor por defecto con la propiedad del padre.

- Anulará el precio añadiendo 50 al precio base si la carga supera los 6kg, seguirá siendo una propiedad de solo acceso.
- Esta clase tendrá el constructor por defecto al que le llegará el consumo, el peso y la carga, además de otro constructor al que además de los argumentos anteriores también le llegará el color como String.
- Deberás sobreescribir el toString para conseguir una salida de la forma:

```
B GRIS 80.0 5Kg de carga -- 240.0€
```

> **Importante**
>
> El siguiente código deberá ser añadido al programa principal para textear las anteriores implementaciones. Fíjate en la diferencia de la salida, usando las distintas funciones Scope. Si no entiendes bien su funcionalidad, repasa el uso investigando en la red.

```
val lavadora=Lavadora('B',"Gris",80f,5)
println(lavadora)
al x=lavadora.let {
    it.carga=8
    println(it)
}
println(x)
val lavadora2=Lavadora('A',60f,9).apply {
        println(this)
        color=Electrodomestico.Color.ROJO
        println(this)
    }
println(lavadora2)
```

Ejercicio 2

Crearemos una **data class Serie** con los siguientes atributos: codigo de tipo Int, titulo de tipo String, numeroTemporadas de tipo Int, numeroCapitulosTemporada de tipo Int, temporada de tipo Int, capitulo de tipo Int, acabada Boolean, genero de tipo Genero y creador de tipo String. Además tendremos una enumeración Genero con un constructor al que se le pasará el codigo

del género de tipo Int y los ejemplos de tipo String. Los valores de la enumeración serán los siguientes:

```
0  INDEFINIDO "Indefinidos"
11 THRILLER "Black Mirror, Breaking Bad, Sons of Anarchy, Casa de Papel..."
21 ANIMACION "Rick y Morty, Bojack Horseman, Mr. Pickles..."
31 FICCION "The 100, Orphan Black, The Leftovers, Stranger Things..."
41 DRAMA "Homeland, The Good Wife, House of Cards..."
51 COMEDIA "Arrested Development, The Bid Bang Theory, Orange Is The New Black, Sex Educ
61 TERROR "American Horror Story, The Outsider, The Haunting of Hill House..."
```

Además la enumeración anulará el método toString para conseguir una salida del tipo:

```
genero=FICCION como [The 100, Orphan Black, The Leftovers, Stranger Things...]
```

Añade al programa principal, como mínimo, un objeto serie con la propiedad acabada a false y muestra la salida, modifica la propiedad acabada a true mediante el constructor copia correspondiente y vuelve a mostrar la salida.

Ejercicio 3

Vamos a crear nuestra primera aplicación usando un tipo sellado, a diferencia de en los apuntes donde se ejemplifica con una clase sellada, aquí nos crearemos una **interface selled SeriesEvent** (googlea para ver la pequeña diferencia con las clases selladas). Esta interface nos permitirá gestionar más fácilmente los eventos que ocurrirán en una futura aplicación móvil para la administración de series.

Deberá de poder gestionar los siguientes eventos:

- Listar todas las series
- Listar las series acabadas
- Listar las series por género, a este evento le llegará un string género.
- Marcar una serie como acabada, a este evento le llegará un código de la serie.
- Marcar el seguimiento de una serie, a este evento le llegará un código de la serie, el número de temporada y el número de capitulo por el que nos hemos quedado.

Tendremos una función **onSeriesEvent(seriesEvent: SeriesEvent)** al que le llegará el evento y se encargará de gestionar cada uno de ellos mediante when, mostrando el mensaje con los datos que llegan.

> **Aviso**
>
> Por ahora solamente vamos a mostrar un mensaje para cada evento con los datos que le llegan a este, en ejercicios posteriores ampliaremos la aplicación.

El código de la función principal será el siguiente:

```
do
{
    println("1. Listar todas mis series")
    println("2. Listar mis series acabadas")
    println("3. Listar mis series por género")
    println("4. Marcar serie como acabada")
    println("5. Marcar seguimiento de la serie")
    println("6. Salir")
    val opcion=readln()
    accionSeleccion(opcion)
} while(opcion!="6")
```

Donde **accionSeleccion** será la función encargada de recoger los datos necesario y ejecutar el evento correspondiente según estos datos.

Una posible salida para el evento Marcar seguimiento de la serie, sería:

```
1. Listar todas mis series
2. Listar mis series acabadas
3. Listar mis series por género
4. Marcar serie como acabada
5. Marcar seguimiento de la serie
6. Salir
5
Introduce el código de la serie de la que quieres marcar su posición
1123
Introduce la temporada de la serie a marcar
3
Introduce el capítulo de la serie a marcar
5
Se señala la serie con código: 1123
como vista hasta la temporada - 3 y capítulo 5
```