Ejercicio 1

Descargar este enunciado pdf o html

Partiendo del proyecto **Vehiculos** que contiene ya ciertos módulos y clases que nos van a ser útiles para la realización del examen y entre los cuales **podemos destacar**:

En el paquete `com.vahiculos.data`:

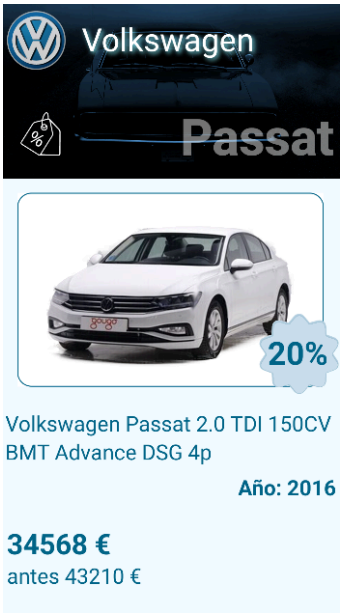
- En el paquete `.coche` están `CocheMock.kt` y `CocheDaoMock.kt` que contiene un listado de coches que vamos a utilizar para la primera carga de datos en la base de datos de Room.

En el paquete `com.vehiculos.ui.composables`:

- Dispones de las clases utilizadas en los materiales del curso denominas `FielTextCommon.kt` e `FilterChipCommon.kt` que puedes utilizar para algunos de los apartados del examen.

En el paquete `com.vehiculos.ui.features`:

- Dispones del data class `CocheUiState.kt` que vamos a utilizar para guardar el estado de la ficha de un coche y de los items de la lista.
- Em el paquete `.fichacoche` dispones de `FichaCocheScreen.kt` que contiene la implementación de la UI de la ficha de un coche y que deberás completar según las especificaciones.
- Em el paquete `.galeriacoches` dispones de `ItemListaCoches.kt` que contiene la definiciónd e la UI para ver un coche en la lista a modo de galería de coches y que puedes ver en la imagen de ejemplo justo abajo.



En el paquete `com.vehiculos.utilities`:

- Dispones de diferentes clases de utilidad utilizadas en los materiales del curso y que puedes utilizar si lo consideras necesario de forma opcional.

Especificaciones

1. Crea un BD con Room para almacenar todos los datos de nuestro modelo denominada "***coches.db***". La BD debe contener una única tabla denominada ***coches*** donde la clave primaria será el campo ***id*** de tipo ***Int*** que tienes en el modelo. Los campos de las tabla deben ir en `snake_casing`

   > **Nota**
   >
   > Fíjate bien el los datos del mock para definir `CocheEntity` y fíjate que la foto es de tipo `String?` que contiene la URL de la imagen o null si no tiene.

   Las operaciones sobre la tablas serán todas asíncronas y debes definir las siguientes:
   - Operaciones básicas `insert`, `delete`, `update` y `count`
   - `get()`: Devuelve todos los coches de la base de datos.
   - `get(Int)`: Devuelve un coche de la base de datos a partir de su id.
   - `getOertas()`: Devuelve todos los coches de la base de datos que tengan un porcentaje de descuento mayor que 0.
     `SELECT * FROM coches WHERE porcentaje_descuento > 0`
   - `getOrdenadosPrecio()`: Devuelve todos los coches de la base de datos ordenados por precio de menor a mayor.
     `SELECT * FROM coches ORDER BY precio ASC`
   - `getOertasOrdenadasPrecio()`: Devuelve todos los coches de la base de datos que tengan un porcentaje de descuento mayor que 0 ordenados por precio de menor a mayor.
2. Prepara los '*providers*' que consideres necesarios de las instancias de Room para la inyección de dependencias con **Hilt**. Teniendo en cuenta que se creará una **instancia única** de cada uno de ellos.
3. Modifica el '*repositorio*' para usar los métodos definidos.
4. Realiza la primera carga de datos de la base de datos de Room a partir de los datos `CocheDaoMock`. **Solo si la BD está vacía**.
   Para ello, realiza la carga de datos en la BD al iniciar la aplicación. Utilizando el método `onCreate` de la clase `VehiculosApplication`.
   Puedes inyectar de forma sencilla los datos de prueba y el Dao o el Repositorio de forma simple utilizando Hilt de la siguiente manera:

```
@HiltAndroidApp
class VehiculosApplication : Application() {
    @Inject
    lateinit var daoMock: CocheDaoMock
    @Inject
    lateinit var daoEntity: CocheDao
    ...
}
```

> **Nota**
>
> Recuerda que los métodos de Room deben ser asíncronos y que debes tener en cuenta que la carga de datos se realiza solo si la BD está vacía.

5. Por último, comprueba con el App Inspection que la carga de datos se ha realizado correctamente.