cpifp Los Enlaces

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

2º DESARROLLO DE APLICACIONES MULTIPLATAFORMA

TEMA 9. RESUMEN ROOM

cpifp Los Enlaces

T9. RESUMEN ROOM

1. COMPONENTES PRINCIPALES

Estos son los tres componentes principales de Room:

- La **clase de la base de datos** que contiene la base de datos y sirve como punto de acceso principal para la conexión subyacente a los datos persistentes de la app.
- Las **entidades** de datos que representan tablas de la base de datos de tu app.
- Los **objetos de acceso a datos** (DAOs) que proporcionan métodos que tu app puede usar para consultar, actualizar, insertar y borrar datos en la base de datos.

cpifp Los Enlaces

T9. RESUMEN ROOM

2. EJEMPLO ANDROID DEVELOPERS MODIFICADO

Entidad de datos: El siguiente código define una entidad de datos UserEntity. Cada instancia de UserEntity representa una fila en una tabla *user_table* en la base de datos de la app.

```
@Entity(tableName = "user_table")
data class UserEntity(
   @PrimaryKey(autoGenerate = true) @ColumnInfo(name = "id") val id: Int = 0,
   @ColumnInfo(name = "first_name") val firstName: String?,
   @ColumnInfo(name = "last_name") val lastName: String?
)

fun UserData.toDatabase() = UserEntity( firstName = name, lastName= surname)
                                        /* campo_tabla <- atributo_objeto_UserData */
```

T9. RESUMEN ROOM

2. EJEMPLO ANDROID DEVELOPERS MODIFICADO

Objeto de acceso a datos (DAO): El siguiente código define un DAO llamado UserDao que proporciona los métodos que el resto de la app usa para interactuar con los datos de la tabla *user_table*.

```
@Dao
interface UserDao {
    @Query("SELECT * FROM user_table")
    fun getAll(): List<UserEntity>

    @Query("SELECT * FROM user_table
WHERE id IN (:userIds)")
    fun loadAllByIds(userIds: IntArray):
List<UserEntity>
```

```
    @Query("SELECT * FROM user_table
WHERE first_name LIKE :first AND " +
"last_name LIKE :last LIMIT 1")
    fun findByName(first: String, last:
String): UserEntity
    @Insert
    fun insertAll(vararg users: UserEntity)
    @Delete
    fun delete(user: UserEntity)
}
```

cpi fp Los Enlaces

T9. RESUMEN ROOM

2. EJEMPLO ANDROID DEVELOPERS MODIFICADO

Base de datos: Se define una clase AppDatabase para contener la base de datos que define la configuración de la misma y sirve como el punto de acceso principal de la app a los datos persistentes.

```
@Database(entities = [UserEntity::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
   abstract fun userDao(): UserDao
}
// La versión es para hacer migraciones (como un control de versiones para BD)
```

T9. RESUMEN ROOM

2. EJEMPLO ANDROID DEVELOPERS MODIFICADO

La clase de la base de datos debe cumplir con las siguientes condiciones:

- La clase debe tener una anotación **@Database** que incluya un array entities que enumere todas las entidades de datos asociados con la base de datos.
- Debe ser una clase abstracta que extienda **RoomDatabase**.
- Para cada clase **DAO** que se asoció con la base de datos, esta base de datos debe definir un método abstracto que tenga cero argumentos y muestre una instancia de la clase DAO.

cpi fp Los Enlaces

T9. RESUMEN ROOM

2. EJEMPLO ANDROID DEVELOPERS MODIFICADO

Uso: Después de definir la entidad de datos, el DAO y el objeto de base de datos, puedes usar el siguiente código para crear una instancia de la base de datos.

```
val db = Room.databaseBuilder(
    applicationContext,
    AppDatabase::class.java, "database-name"
).build()

db.userDao().insertAll(entityList)
db.userDao().delete(entityList[0])
val name = "alba"  val surname = "marquez"
db.userDao().findByName("%$name%", "%$surname%")
```

T9. RESUMEN ROOM

3. ALMACENAMIENTO EN ROOM

Mapeo: Para transformar cualquier tipo de datos a objetos de la base de datos se ha de implementar el método `toDatabase()` para definir dicha transformación.

```
// En la clase UserEntity
fun UserData.toDatabase() = UserEntity( firstName = name, lastName= surname)

// En la actividad principal
val dataList = listOf<UserData>(
   UserData("Cristobalina", "Torralba Turrao"),
   UserData("Paulino", "Buey Mancho")
)
val entityList = dataList.map { it.toDatabase() }
```

cpifp Los Enlaces

T9. RESUMEN ROOM

3. ALMACENAMIENTO EN ROOM

Mapeo: También se puede almacenar un único registro si se define el método adecuado en el DAO correspondiente.

```
// En la interfaz UserDao
   @Insert(onConflict = OnConflictStrategy.REPLACE)
   fun insertAll(users: List<UserEntity>)
   @Insert
   fun insertByObject(user: UserEntity)

// En la actividad principal
   val entityList = dataList.map { it.toDatabase() }
   db.userDao().insertAll(entityList)
   val user1 = UserData("Patrocinio", "Gistau Belzuz")
   db.userDao().insertByObject(user1.toDatabase())
```

T9. RESUMEN ROOM

4. CONSULTAS EN ROOM

Consultas habituales: Dentro de un DAO se pueden definir algunos tipos de consultas útiles para cualquier tabla.

```
// En la interfaz UserDao
   @Query("SELECT * FROM user_table")
   fun getAll(): List<UserEntity>

// En la actividad principal obtendríamos una lista de objetos tipo UserEntity con la
que podemos interactuar
   val usersList = db.userDao().getAll()
   val firstUserName = usersList[0].firstName
// O directamente...
   val firstUserName = db.userDao().getAll()[0].firstName
```

T9. RESUMEN ROOM

4. CONSULTAS EN ROOM

Consultas habituales: Dentro de un DAO se pueden definir algunos tipos de consultas útiles para cualquier tabla.

```
// En la interfaz UserDao
   @Query("SELECT * FROM user_table WHERE id IN (:userIds)")
   fun loadAllByIds(userIds: IntArray): List<UserEntity>

// En la actividad principal obtendríamos una lista de objetos tipo UserEntity con la
que podemos interactuar
   val users = [2, 5, 7]
   val firstUserSelected: UserEntity = db.userDao().loadAllByIds()[0]

   val firstUser: String = firstUserSelected.firstName + firstUserSelected.lastName
```

T9. RESUMEN ROOM

4. CONSULTAS EN ROOM

Consultas habituales: Dentro de un DAO se pueden definir algunos tipos de consultas útiles para cualquier tabla.

```
// En la interfaz UserDao
   @Query("SELECT * FROM user_table WHERE id = :userId")
   fun getUser(userId: Int): UserEntity

// En la actividad principal obtendríamos un objeto tipo UserEntity
   val id = 1
   val user: UserEntity = db.userDao().getUser(id)
   val name = user.firstName
   val surname = user.lastName
```

cpifp Los Enlaces

T9. RESUMEN ROOM

4. CONSULTAS EN ROOM

Consultas habituales: Dentro de un DAO se pueden definir algunos tipos de consultas útiles para cualquier tabla.

```
// En la interfaz UserDao
   @Query("SELECT * FROM user_table WHERE first_name LIKE :first AND "+ "last_name
LIKE :last")
   fun findByName(first: String, last: String): List<UserEntity>

// En la actividad principal obtendríamos una lista de objetos tipo UserEntity
   val users = db.userDao().findByName("alf", "martín")
// En este caso, si hay varios usuarios que coincidan con la búsqueda (e.g. Alfredo
Martín y Alfonso Martínez), creará una lista ordenada
   val secondUserName = users[1].firstName
// Esta variable almacenará "Alfredo" en este ejemplo
```

cpifp Los Enlaces

T9. RESUMEN ROOM

4. CONSULTAS EN ROOM

Consultas habituales: Dentro de un DAO se pueden definir algunos tipos de consultas útiles para cualquier tabla.

```
// En la interfaz UserDao
   @Query("SELECT * FROM user_table WHERE first_name LIKE :first AND "+ "last_name
LIKE :last LIMIT 1")
   fun findByName(first: String, last: String): UserEntity

// En la actividad principal obtendríamos un objeto tipo UserEntity
   val user: UserEntity = db.userDao().findByName("alf", "martín")

// En este caso, si hay varios usuarios que coincidan con la búsqueda (e.g. Alfredo
Martín y Alfonso Martínez), tomará el primero que encuentre (Alfonso Martínez, por
orden alfabético en el campo firs_name)
```

4. CONSULTAS EN ROOM

Consultas habituales: Dentro de un DAO se pueden definir algunos tipos de consultas útiles para cualquier tabla.

```
// En la interfaz UserDao
   @Query("SELECT * FROM user_table ORDER BY id DESC LIMIT 5")
   fun newerUsers(): List<UserEntity>

// En la actividad principal obtendríamos una lista de objetos tipo UserEntity con
los 5 últimos usuarios registrados
   val lastRegisteredUsers: List<UserEntity> = db.userDao().newerUsers()

   val lastEntry: UserEntity = lastRegisteredUsers.last()
```