Predavanje 12

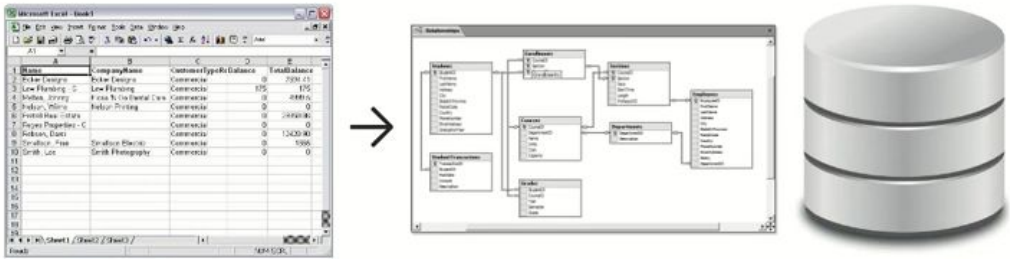
Pohranjivanje podataka

Sadržaj

- Snimanje podataka
- SQL baze podataka
- Rad sa .db i .sql datotekama
- Room biblioteka

Šta je baza podataka?

- Relacijska baza podataka:
- Metoda strukturiranja podataka u obliku tabela koje su međusobno povezane zajedničkim atributima.
- Red tabele odgovara jedinici podataka koja se naziva zapis.
- Kolona predstavlja atribut tog zapisa.
- Baza podataka je skup jedne ili više tabela, uz podršku za uobičajene operacije kao što su:
  - CRUD-a – stvaranje (create), čitanje (read), ažuriranje (update) i brisanje (delete).
  - Brzo pretraživanje



Gdje su podaci?

- Unutar vašeg Android uređaja (lokalna baza podataka)
  - SharedPreferences
  - Datoteke
  - Relacijska baza SQLite
  - NoSQL baze podataka
- Na remote Web serveru
- Na više remote Web servera (cloud)

SharedPreferences

- SharedPreferences je jednostavan mehanizam za pohranu malih količina podataka u formatu ključ-vrijednost.
  - Podaci ostaju sačuvani čak i kada se aplikacija zatvori.
- Uobičajeni primjeri korištenja:
  - Pohrana korisničkih postavki (npr. dark-mode način rada).
  - Čuvanje preferenci aplikacije, zastavica ili stanja.
  - Pohrana primitivnih tipova podataka.
- Pohranjuje key-value parove u XML formatu.
- Podržava primitivne tipove podataka:
  - String, int, float, boolean, long.
- Brz i lagan za male količine podataka.
  - Nije pogodan za velike ili kompleksne strukture podataka.

SharedPreferences

- Dobivanje instance SharedPreferences:
  - Koristi se metoda getSharedPreferences() za kreiranje ili otvaranje fajla preferenci.
- Pohrana podataka:
  - Podaci se pohranjuju u key-value formatu koristeći editor.
  - apply() se koristi za asinhrono spremanje promjena (brže od commit()).
- Čitanje podataka:
  - Metode poput getString(), getInt() i getBoolean() se koriste za dohvat vrijednosti uz opciju podrazumijevane vrijednosti.
- Podaci su sigurni unutar aplikacije, ali nisu šifrirani.
- Nije preporučljivo za osjetljive podatke.

SharedPreferences

- Snimanje podataka

```
val sharedPreferences = getSharedPreferences("MyPrefs", MODE_PRIVATE)
val editor = sharedPreferences.edit()
editor.putString("username", "JohnDoe")
editor.putInt("age", 30)
editor.putBoolean("isLoggedIn", true)
editor.apply() // Save changes asynchronously
```

- Preuzimanje podataka

```
val sharedPreferences = getSharedPreferences("MyPrefs", MODE_PRIVATE)
val username = sharedPreferences.getString("username", "DefaultName")
val age = sharedPreferences.getInt("age", 0)
val isLoggedIn = sharedPreferences.getBoolean("isLoggedIn", false)

Log.d("SharedPrefs", "Username: $username, Age: $age, LoggedIn: $isLoggedIn")
```

Zašto koristiti baze podataka?

- Moćna: Omogućuje pretraživanje, filtriranje i kombiniranje podataka iz više izvora.
- Brza: Pretraživanje i filtriranje baze podataka mnogo je brže u usporedbi s datotekama.
- Velika: Dobro skalira s vrlo velikim količinama podataka.
- Sigurna: Ugrađeni mehanizmi za oporavak od grešaka (transakcije).
- Višekorisnička: Omogućuju da više korisnika istovremeno pregledava i uređuje podatke.
- Apstraktna: Omogućuje sloj apstrakcije između pohranjenih podataka i aplikacija.
- Zajednička sintaksa: Programi za baze podataka koriste iste SQL naredbe.

Softveri za baze podataka

- Oracle
  - komercijalni sistem za velike poslovne aplikacije.
- Microsoft
  - SQL Server – Robustan i moćan sistem za poduzeća.
  - Access – Jednostavan sustav za manje baze podataka i osobnu upotrebu
- PostgreSQL
  - Moćan, kompleksan i besplatan open-source sustav za baze podataka.
- SQLite
  - Prijenosiva, lagana i besplatna open-source baza podataka, često ugrađena u mobilne aplikacije.
- MySQL
  - Jednostavan, besplatan open-source sistem za baze podataka.
  - Čest izbor za "LAMP" stack (Linux, Apache, MySQL, PHP).
  - Wikipedia koristi PHP i MySQL.
- MongoDB
  - Popularna NoSQL baza podataka za pohranu dokumenata (JSON-like format).

Firebase - Firestore

- Firebase je Backend-as-a-Service (BaaS) platforma (Google).
- Firestore je NoSQL cloud baza podataka (dokumenti u JSON-like formatu).
- Podaci se čuvaju u kolekcijama i dokumentima (nema tabela).
- Real-time sinhronizacija – promjene se odmah vide na svim uređajima.
- Odlična integracija sa Android aplikacijama.
- Podrška za:
  - autentifikaciju korisnika (Firebase Auth)
  - offline rad i automatsku sinhronizaciju
- Često se koristi kada:
  - ne želimo pisati vlastiti backend
  - aplikacija treba brzo da se razvije
  - podaci se dijele između više korisnika

Cloud Firestore

Kako pričati sa bazom?

- SQL (Structured Query Language): relacione baze podataka obično koriste SQL da definiraju, upravljaju i pretražuju podatke
  - deklarativni jezik: opisuje koje podatke tražite, a ne tačno kako ih pronaći.

```
SELECT name
FROM   countries
WHERE  population > 20000000;
```

code	name	continent	independence_year	population	gnp	head_of_state	...
AFG	Afghanistan	Asia	1919	22720000	5976.0	Mohammad Omar	...
NLD	Netherlands	Europe	1581	15864000	371362.0	Beatrix	...
...	...	...	...	...	...	...	...

countries

Primjer baze podataka

id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

students

id	name
1234	Krabappel
5678	Hoover
9012	Stepp

teachers

id	name	teacher_id
10001	Computer Science 142	1234
10002	Computer Science 143	5678
10003	Computer Science 190M	9012
10004	Informatics 100	1234

courses

student_id	course_id	grade
123	10001	B-
123	10002	C
456	10001	B+
888	10002	A+
888	10003	A+
404	10004	D+

grades

Primjer baze podataka

code	name	continent	independence_year	population	gnp	head_of_state	...
AFG	Afghanistan	Asia	1919	22720000	5976.0	Mohammad Omar	...
NLD	Netherlands	Europe	1581	15864000	371362.0	Beatrix	...
...	...	...	...	...	...	...	...

countries (Other columns: region, surface_area, life_expectancy, gnp_old, local_name, government_form, capital, code2)

id	name	country_code	district	population
3793	New York	USA	New York	8008278
1	Los Angeles	USA	California	3694820
...	...	...	...	...

cities

country_code	language	official	percentage
AFG	Pashto	T	52.4
NLD	Dutch	T	95.6
...	...	...	...

languages

Primjer baze podataka

id	first_name	last_name	gender
433259	William	Shatner	M
797926	Britney	Spears	F
831289	Sigourney	Weaver	F
...			

actors

id	name	year	rank
112290	Fight Club	1999	8.5
209658	Meet the Parents	2000	7
210511	Memento	2000	8.7
...			

movies

actor_id	movie_id	role
433259	313398	Capt. James T. Kirk
433259	407323	Sgt. T.J. Hooker
797926	342189	Herself
...		

roles

movie_id	genre
209658	Comedy
313398	Action
313398	Sci-Fi
...	

movies_genres

id	first_name	last_name
24758	David	Fincher
66965	Jay	Roach
72723	William	Shatner
...		

directors

director_id	movie_id
24758	112290
66965	209658
72723	313398
...	

movies_directors

Primjeri SQL komandi

Create

```
INSERT INTO colors VALUES ("red", "#FF0000");
```

Read

```
SELECT * from colors;
```

Update

```
UPDATE colors SET hex="#DD0000" WHERE name="red";
```

Delete

```
DELETE FROM colors WHERE name = "red";
```

SELECT komanda

```
SELECT column(s) FROM table WHERE condition;

SELECT name, population FROM cities
                        WHERE country_code = "FSM";
```

- Pretraživanje baze podataka i dobivanje skupa rezultata
  - Naziv(i) kolona nakon SELECT određuju koji će dijelovi redova biti vraćeni.
  - Nazivi tabela i kolona razlikuju velika i mala slova (case-sensitive).
  - SELECT DISTINCT uklanja duplikate iz rezultata.
  - SELECT * vraća sve stupce iz tablice.
- WHERE klauzula filtrira redove na temelju vrijednosti u kolonama.
  - U velikim bazama podataka, WHERE klauzula je ključna za smanjenje veličine rezultata.

WHERE klauzula

```
SELECT name, gnp FROM countries WHERE gnp > 2000000;

SELECT * FROM cities WHERE code = 'USA'
                     AND population >= 2000000;

SELECT code, name, population FROM countries
WHERE  name LIKE 'United%';
```

- WHERE klauzula može koristiti sljedeće operatore:
  - =, >, >=, <, <=
  - <> : nije jednako (neki sistemi podržavaju !=)
  - BETWEEN min AND max : filtrira vrijednosti unutar zadanog raspona
  - LIKE uzorak : koristi se za pretragu prema uzorku (npr. % na početku/kraju za pretragu prefiksa/sufiksa/podstringa)
  - IN (vrijednost, vrijednost, ..., vrijednost) : provjerava je li vrijednost u popisu zadanih vrijednosti
  - condition1 AND condition2 : obje uslova moraju biti zadovoljena
  - condition1 OR condition2 : barem jedan od uslova mora biti zadovoljen

Android SQLiteDatabase


```
val db = openOrCreateDatabase("name", MODE_PRIVATE, null)
db.execSQL("SQL query")
```

- Metoda openOrCreateDatabase stvara novu praznu bazu podataka s tim imenom ili otvara postojeću.
- Kada je baza otvorena, možete koristiti metode za izvršavanje SQL naredbi:
  - rawQuery – ako upit vraća rezultate (npr. SELECT).
  - execSQL – ako upit ne vraća rezultate (npr. INSERT, DELETE).
- Ako upit sadrži neispravan SQL, generiše se SQLiteException.
- SQLite baze podataka pohranjuju se na uređaju u:

  /data/data/packageName/databases/

Primjer

Napravite Android aplikaciju koja pretražuje bazu dječijih imena u BiH

- Zatražite od korisnika unos imena i spola.
- Pretražite tabelu **ranks** za to ime/spol.
- Prikažite rangove vizualno u aplikaciji.
- Također postoji i tabela **meanings** koja sadrži značenja imena.
    - Pretražite tablicu **meanings** za značenje imena koje je korisnik unio i prikažite ga.

ranks

name	gender	rank	year	total
Ahmed	M	1	2018	291
Merjem	F	1	2021	227

meanings

name	meaning
Ahmed	mnogo hvaljen
Merjem	čista, nevina

Učitavanje iz baze - importovanje .sql file-a

- .sql fajl sadrži niz SQL naredbi.
- Uobičajeni format za izvoz cijele baze podataka i njenog sadržaja.
- Koristi se za kreiranje sigurnosne kopije ili za obnavljanje baze podataka na drugi server.
- Kako importovati .sql fajl u Android aplikaciju:
  - Stavite .sql fajl u folder res/raw vaše aplikacije.
  - Otvorite ga pomoću klase Scanner.
  - Čitajte linije dok ne pronađete tačku-zarez (;).
  - Izvršite string koji pročitate kao upit pomoću execSQL.
  - Ponavljajte postupak za svaki upit.

Učitavanje iz baze - importovanje .sql file-a

```kotlin
/* Čita iz `.sql` fajla i izvršava njegove SQL naredbe. */
private fun importDatabase(dbName: String) {
    val db = openOrCreateDatabase(dbName, MODE_PRIVATE, null)
    val resId = resources.getIdentifier(dbName, "raw", packageName)
    val scan = Scanner(resources.openRawResource(resId))

    // Kreiranje i izvršavanje upita
    var query = ""
    while (scan.hasNextLine()) {
        val line = scan.nextLine()
        if (line.trim().startsWith("--")) continue // Ukloni komentare
        query += "$line\n"
        if (query.trim().endsWith(";")) {
            db.execSQL(query)
            query = ""
        }
    }
}
```

Importovanje .db datoteke

- .db fajl sadrži unaprijed popunjenu SQLite bazu podataka.
  - Korisno za isporuku aplikacije sa postojećim podacima.
  - Izbjegava kreiranje i popunjavanje baze tokom rada aplikacije.
- Koraci za importovanje .db datoteke:
  - Postavite .db fajl u assets folder:
    - app/src/main/assets/database.db
  - Kopirajte .db fajl u internu memoriju aplikacije pri prvom pokretanju.
  - Otvorite i izvršavajte upite pomoću SQLiteDatabase.

Importovanje .db datoteke

```
import android.content.Context
import
android.database.sqlite.SQLiteDatabase
import android.util.Log
import java.io.File
import java.io.FileOutputStream
import java.io.IOException
```

```
fun copyDatabaseFromAssets(context: Context, dbName: String) {
    val dbPath = context.getDatabasePath(dbName).absolutePath
    val dbFile = File(dbPath)

    if (dbFile.exists()) {
        Log.d("Database", "Database already exists at $dbPath")
        return
    }

    try {
        val inputStream = context.assets.open(dbName)
        val outputStream = FileOutputStream(dbPath)

        val buffer = ByteArray(1024)
        var length: Int
        while (inputStream.read(buffer).also { length = it } > 0) {
            outputStream.write(buffer, 0, length)
        }

        outputStream.flush()
        outputStream.close()
        inputStream.close()

        Log.d("Database", "Database copied successfully to $dbPath")
    } catch (e: IOException) {
        Log.e("Database", "Error copying database: ${e.message}")
    }
}
```

Cursor

- Android SQLite API vraća objekt nazvan Cursor koji vam omogućuje da iterirate kroz rezultate SELECT upita.
  - Slično konceptu iteratora.
- Kao pokazivač pozicioniran na određeni red iz skupa rezultata.
  - Možete pomaknuti kursor na sljedeći red rezultata.
  - Možete zatražiti od kursora vrijednosti kolona njegovog „trenutnog” reda.

```
SELECT id, email FROM students;
```

	id	name	email
cursor →	123	Bart	bart@fox.com
	456	Milhouse	milhouse@fox.com
	888	Lisa	lisa@fox.com
	404	Ralph	ralph@fox.com

students

Cursor

```
2 val cursor = db.rawQuery("SELECT id, email FROM students", null)
3 while (cursor.moveToNext()) {
4     val id = cursor.getInt(cursor.getColumnIndex("id"))
5     val email = cursor.getString(cursor.getColumnIndex("email"))
6     ...
7 }
8 cursor.close()
```

```
SELECT id, email FROM students;
```

	id	name	email
cursor →	123	Bart	bart@fox.com
	456	Milhouse	milhouse@fox.com
	888	Lisa	lisa@fox.com
	404	Ralph	ralph@fox.com

students

Direktna interakcija sa bazom

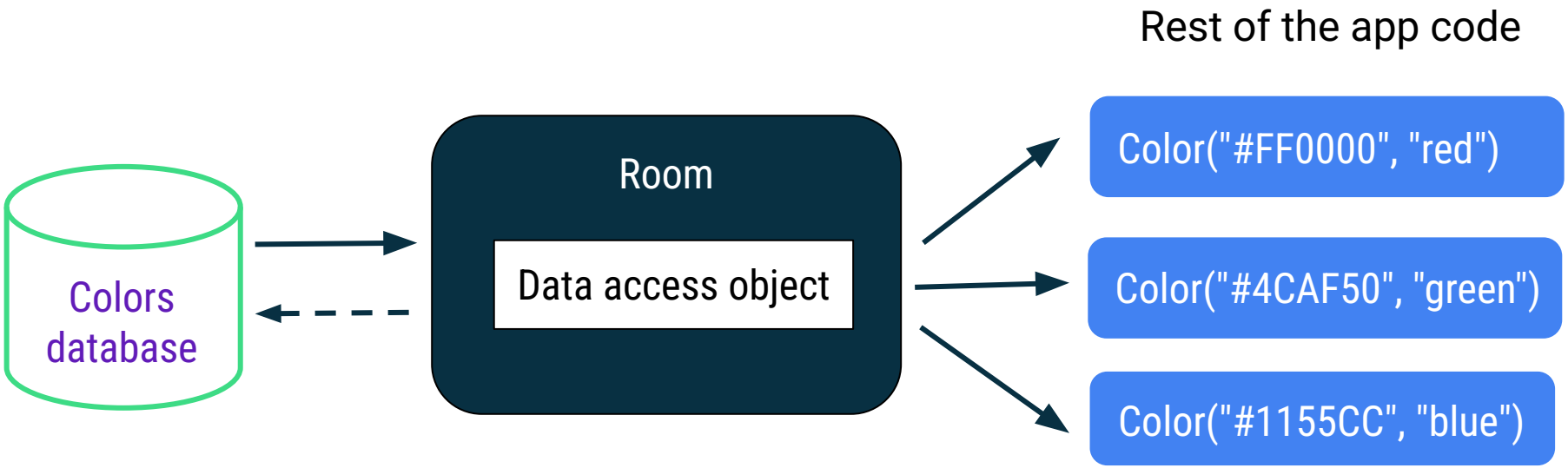
- Nema provjere u vrijeme kompajliranja
  - Greške u SQL upitima (npr. sintaksa ili kolone koje ne postoje) neće biti otkrivene sve do izvođenja aplikacije.
  - To može dovesti do runtime grešaka koje je teže dijagnosticirati.
- Puno dodatnog koda (boilerplate)
  - Potrebno je mnogo koda za ručno mapiranje podataka iz SQL upita u objekte.
  - Za svaki upit morate parsirati kolone i dodjeljivati vrijednosti objektima.

Room

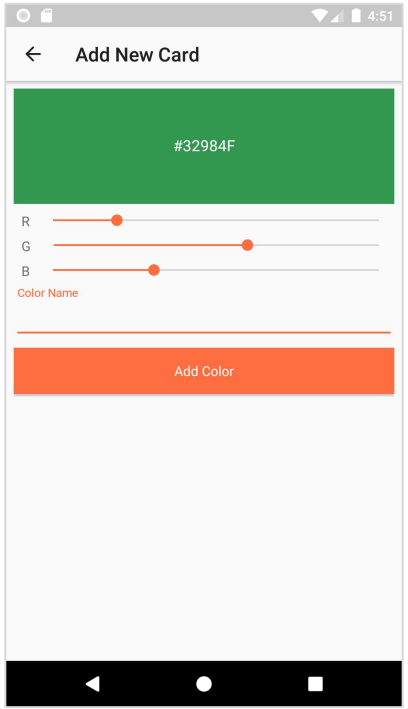
Add Gradle dependencies

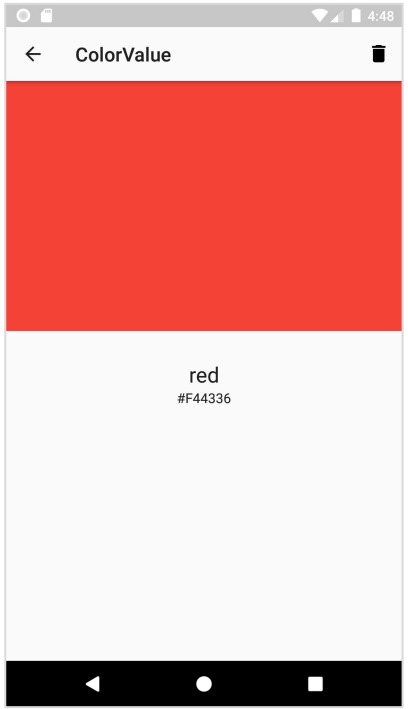
```
plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
    id("kotlin-kapt")
}


dependencies {
    …
    implementation("androidx.room:room-runtime:2.6.1")
    kapt("androidx.room:room-compiler:2.6.1")
    implementation("androidx.room:room-ktx:2.6.1")
    …
 }
```

Room


Rest of the app code
Colors database
Room
Data access object
Color("#FF0000", "red")
Color("#4CAF50", "green")
Color("#1155CC", "blue")

ColorValue app





Room

- Entity `Color`
- DAO `ColorDao`
- Database `ColorDatabase`

Color class

```
data class Color {
    val hex: String,
    val name: String
}
```

Annotations

- Provide extra information to the compiler

  `@Entity` marks entity class, `@Dao` for DAO, `@Database` for database

- Can take parameters

  ```
  @Entity(tableName = "colors")
  ```

- Can autogenerate code for you

Entity

Class that maps to a SQLite database table

- `@Entity`
- `@PrimaryKey`
- `@ColumnInfo`

Example entity

```
@Entity(tableName = "colors")
data class Color {
    @PrimaryKey(autoGenerate = true) val _id: Int,
    @ColumnInfo(name = "hex_color") val hex: String,
    val name: String
}
```

Data access object (DAO)

Work with DAO classes instead of accessing database directly:

- Define database interactions in the DAO.
- Declare DAO as an interface or abstract class.
- Room creates DAO implementation at compile time.
- Room verifies all of your DAO queries at compile-time.

Example DAO

```
@Dao
interface ColorDao {

    @Query("SELECT * FROM colors")
    fun getAll(): Array<Color>

    @Insert
    fun insert(vararg color: Color)

    @Update
    fun update(color: Color)

    @Delete
    fun delete(color: Color)
```

Query

```
@Dao
interface ColorDao {

    @Query("SELECT * FROM colors")
    fun getAll(): List<Color>

    @Query("SELECT * FROM colors WHERE name = :name")
    fun getColorByName(name: String): LiveData<Color>

    @Query("SELECT * FROM colors WHERE hex_color = :hex")
    fun getColorByHex(hex: String): LiveData<Color>
```

Insert

```
@Dao
interface ColorDao {
    ...

    @Insert
    suspend fun insert(vararg color: Color)

    ...
}
```

```
viewModelScope.launch {
    colorDao.insert(
        Color(hex = "#FF0000", name = "red")
    )
}
```

```
viewModelScope.launch {
    colorDao.insert(
        Color("#FF0000", "red"),
        Color("#4CAF50", "green"),
        Color("#1155CC", "blue")
    )
}
```

Update

```
@Dao
interface ColorDao {
    ...

    @Update
    fun update(color: Color)

    ...
}
```

```
@Query("""
        UPDATE colors
        SET name = :newName
        WHERE _id = :id
    """)
suspend fun updateName(id: Int, newName: String)
```

```
viewModelScope.launch {
    colorDao.updateName(
        id = 1,
        newName = "dark red"
    )
}
```

Delete

```
@Dao
interface ColorDao {
    ...

    @Delete
    fun delete(color: Color)

    ...
}
```

Create a Room database

- Annotate class with `@Database` and include list of entities:
  ```
  @Database(entities = [Color::class], version = 1)
  ```

- Declare abstract class that extends `RoomDatabase`:
  ```
  abstract class ColorDatabase : RoomDatabase() {
  ```
  - Declare abstract method with no args that returns the DAO:
    ```
    abstract fun colorDao(): ColorDao
    ```

Example Room database

```
@Database(entities = [Color::class], version = 1)
abstract class ColorDatabase : RoomDatabase() {
    abstract fun colorDao(): ColorDao
    companion object {
        @Volatile
        private var INSTANCE: ColorDatabase? = null
        fun getInstance(context: Context): ColorDatabase {
            ...
        }
    }
    ...
```

Create database instance

```
fun getInstance(context: Context): ColorDatabase {
    return INSTANCE ?: synchronized(this) {
        INSTANCE ?: Room.databaseBuilder(
            context.applicationContext,
            ColorDatabase::class.java, "color_database"
        )
        .fallbackToDestructiveMigration()
        .build()
        .also { INSTANCE = it }
    }
}
```

Get and use a DAO

Get the DAO from the database:

```
val colorDao = ColorDatabase.getInstance(application).colorDao()
```

Create new Color and use DAO to insert it into database:

```
val newColor = Color(hex = "#6200EE", name = "purple")
colorDao.insert(newColor)
```