Mobile Development :

5 : Android Native Development : Part 3

Databases, Network, Camera, Services,...

Professor Imed Bouchrika
National School of Artificial Intelligence
imed.bouchrika@ensia.edu.dz

Outline :

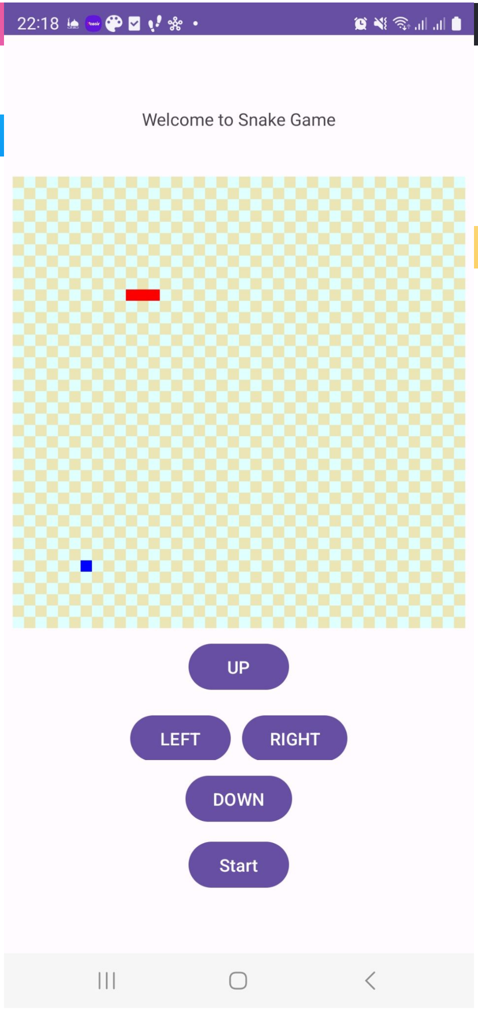
- Asynchronous Prog : Coroutines
- Data Persistence
  - *Preferences*
  - *Relational Databases*
  - *Object Relational Mapping (ORM)*
- Briefly Discuss:
  - *Using Internet Resources*
  - *Background Services*

Recap for Week 4:
Navigation, UIs and Threads

- How to create UI components using XML for Android Apps.
- How to inflate UI Components programmatically inside the Kotlin Code.
- Creating an App with Multiple Activities
- The lifecycle of an Activity
- Using Intent to :
  - *Launch Activities*
  - *Pass data between activities*
  - *Using Intent with an Ack or Callback.*
- Scheduling Method invocation to be called at a later time.

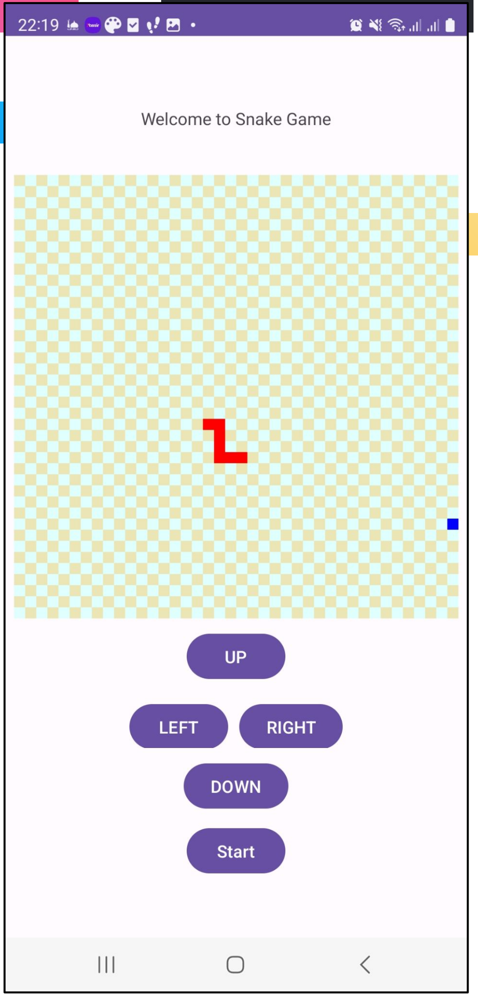
Recap for Week 4: *Navigation, UIs and Threads*

- Snake Game : Plenty of UX/Usability Errors :
    - *Why show the direction button when The game has not started yet ?*



Recap for Week 4: *Navigation, UIs and Threads*

- Snake Game : Problem Solving Exercise
  - *How to eat food ?*
  - *How to create an AI-Competing snake playing Against the human player ?*
    - *This is why you can employ what you have learnt in AI, OR, Optimisation…*



Asynchronous Programming : Coroutines

- Asynchronous programming is a technique used to perform long-running tasks without blocking the main thread of an application.

- When the main thread is busy rendering the user interface and handling user input, you cannot invoke a function to call the network which would block the execution of the main thread causing the UI to freeze

- Traditionally, asynchronous tasks have been implemented using callbacks, which can make the code complex and hard to understand.

Asynchronous Tasks & Coroutines

- **Coroutines**
  - A coroutine is a concurrency design pattern that you can use on Android to simplify code that executes asynchronously
  - Coroutines are lightweight threads:
    - Creating coroutines doesn't allocate new threads. Instead, they use predefined thread pools and smart scheduling for the purpose of which task to execute next.

Asynchronous Tasks & Coroutines

- **Terminologies for Coroutines**
  - Suspending Functions :
    - Functions that can be **suspended** or paused and resumed later without **blocking** the **main thread** .
      - Example :

```
suspend fun doSomethingUsefulOne(): Int {
      delay(1000L)
      return 13
}

suspend fun doSomethingUsefulTwo(): Int {
      delay(1000L)
      return 29
}

fun main() = runBlocking<Unit> {
    val time = measureTimeMillis {
        val one = doSomethingUsefulOne()
        val two = doSomethingUsefulTwo()
        println("The answer is ${one + two}")
    }
    println("Completed in $time ms")
}
```

```
The answer is 42
```

Asynchronous Tasks & Coroutines

- **Terminologies for Coroutines**

```
suspend fun doSomethingUsefulOne(): Int {
      delay(1000L)
      return 13
}

suspend fun doSomethingUsefulTwo(): Int {
```

Suspend functions are only allowed to be called from a coroutine or another suspend function

```
                                                o}")
    }
    println("Completed in $time ms")
}
```

- Example :

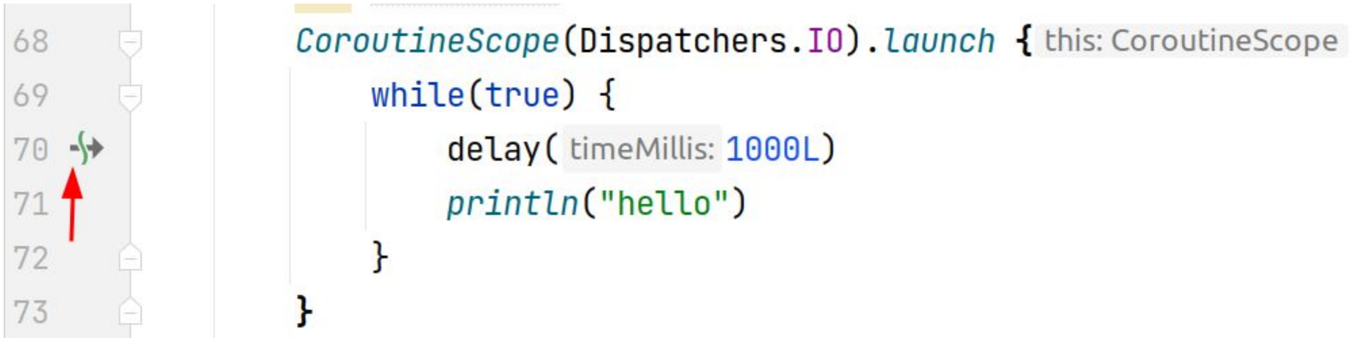
```
The answer is 42
```

Asynchronous Tasks & Coroutines

- **Terminologies for Coroutines**
  - Coroutine Dispatchers :
    - Help coroutines in deciding which thread to use for executing the job.
    - There are four major types of dispatchers :
      - *Main Dispatcher : for the UI*
      - *IO Dispatcher : for all jobs related to reading/writing files or networking*
      - *Default Dispatcher : execute coroutines on a shared background thread*
      - *Unconfined Dispatcher : will use the current active thread.*

Asynchronous Tasks & Coroutines

- **Terminologies for Coroutines**
  - Launching Coroutines :
    - Coroutines are started using either:
      - CoroutineScope.METHOD_NAME(Dispather_TYPE)



```
CoroutineScope(Dispatchers.IO).launch { this: CoroutineScope
    while(true) {
        delay( timeMillis: 1000L)
        println("hello")
    }
}
```

Asynchronous Tasks & Coroutines

- **Terminologies for Coroutines**
  - Launching Coroutines :
    - METHOD_NAME starting coroutines :
      - ***launch*** *: creates and starts a new coroutine. It returns a Job object that can be used to manage the lifecycle of the coroutine. But does not return data …*
      - ***runBlock*** *: blocks the current thread and runs a new coroutine until it completes. Used for testing.*
      - ***async*** *: creates and starts a new coroutine that runs asynchronously. It* ***returns*** *a Deferred (future) object to store data*

Asynchronous Tasks & Coroutines

- **AutoIncrementer App using Coroutines**
  - Launching Coroutines :

```
private fun startCounting(){
    var tx_counter=findViewById<TextView>(R.id.tx_counter)
    var handler = Handler(Looper.getMainLooper())
    val runnable =object : Runnable{
        override fun run() {
                if (!is_running) return
                increment = increment + 1
                tx_counter.setText("" + increment)
                println("running the thread......." )
            handler.postDelayed(this, 1000)
        }
    }
    handler.postDelayed(runnable, 1000)
}
```

Asynchronous Tasks & Coroutines

- **AutoIncrementer App using C**
  - Launching Coroutines :

```
private fun startCounting(){
    var tx_counter=findViewById<TextView>(R.id.tx_counter)
    var handler = Handler(Looper.getMainLooper())
    val runnable =object : Runnable{
        override fun run() {
                while(is_running) {
                    increment = increment + 1
                    tx_counter.setText("" + increment)
                    println("running the thread......." )
                    Thread.sleep(1000)
                }
        }
    }
    handler.postDelayed(runnable, 1000)
}
```

What happens ?

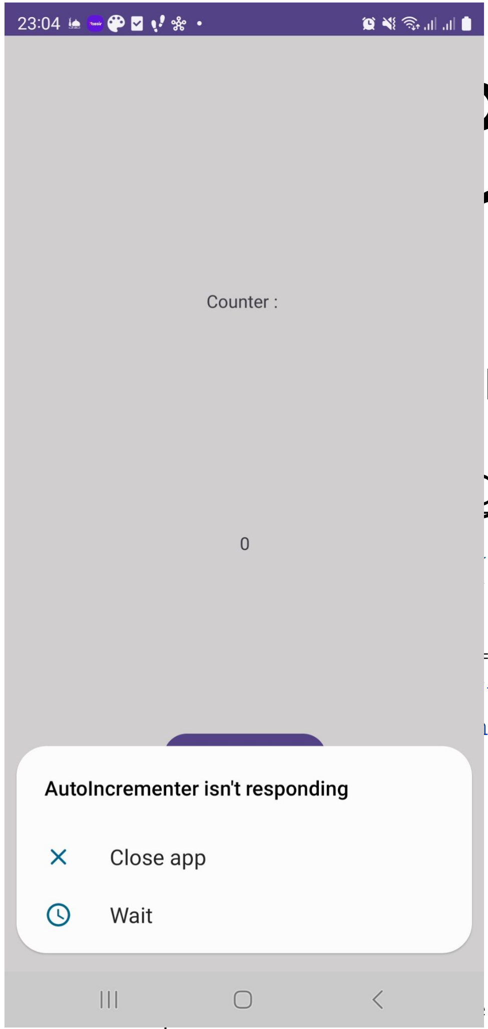
onous Tasks & routines

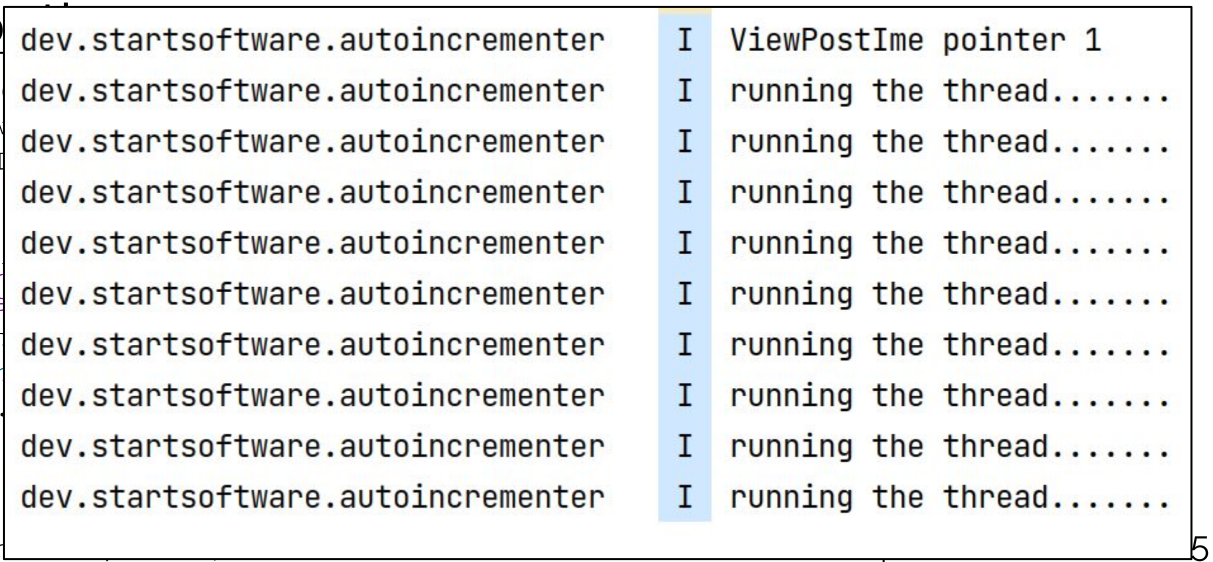
Runs, prints to the console, not updating the UI , the app is crashing…

nter App usin Coroutines

g Coro

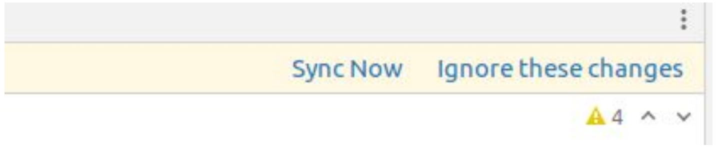
```
Counting
=findView
Handler(I
object :
un run()
ile(is_ru
    increme
    tx coun
    printl
    Thread.

layed(run
}
```




```
dev.startsoftware.autoincrementer   I  ViewPostIme pointer 1
dev.startsoftware.autoincrementer   I  running the thread.......
dev.startsoftware.autoincrementer   I  running the thread.......
dev.startsoftware.autoincrementer   I  running the thread.......
dev.startsoftware.autoincrementer   I  running the thread.......
dev.startsoftware.autoincrementer   I  running the thread.......
dev.startsoftware.autoincrementer   I  running the thread.......
dev.startsoftware.autoincrementer   I  running the thread.......
dev.startsoftware.autoincrementer   I  running the thread.......
dev.startsoftware.autoincrementer   I  running the thread.......
```

Asynchronous Tasks & Coroutines

- **AutoIncrementer App using Coroutines**
  - Update the **build.gradle** File :
  - Make sure you sync the gradle



```
dependencies {

    ...

    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.4.2'
    implementation
'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.4.2'

}
```

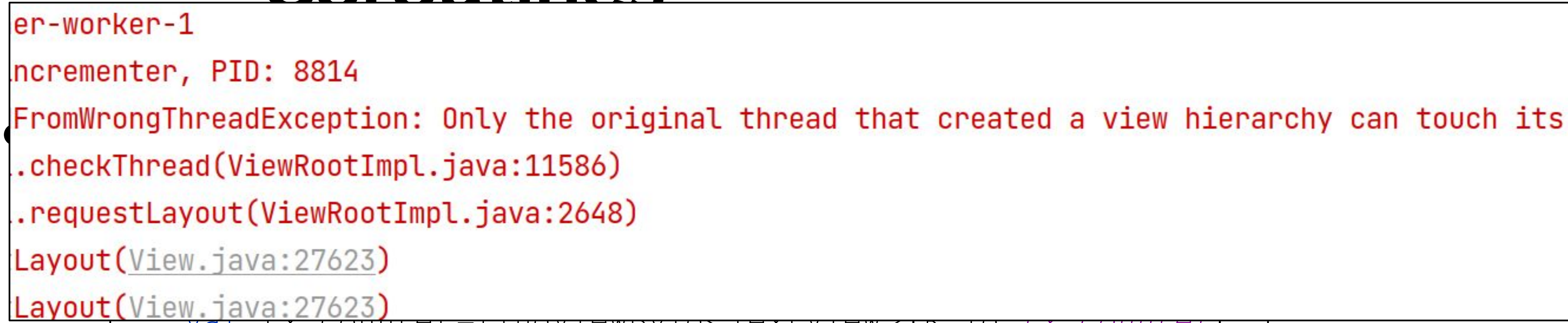
Asynchronous Tasks & Coroutines

- **AutoIncrementer App using Coroutines**
  - Launching Coroutines :

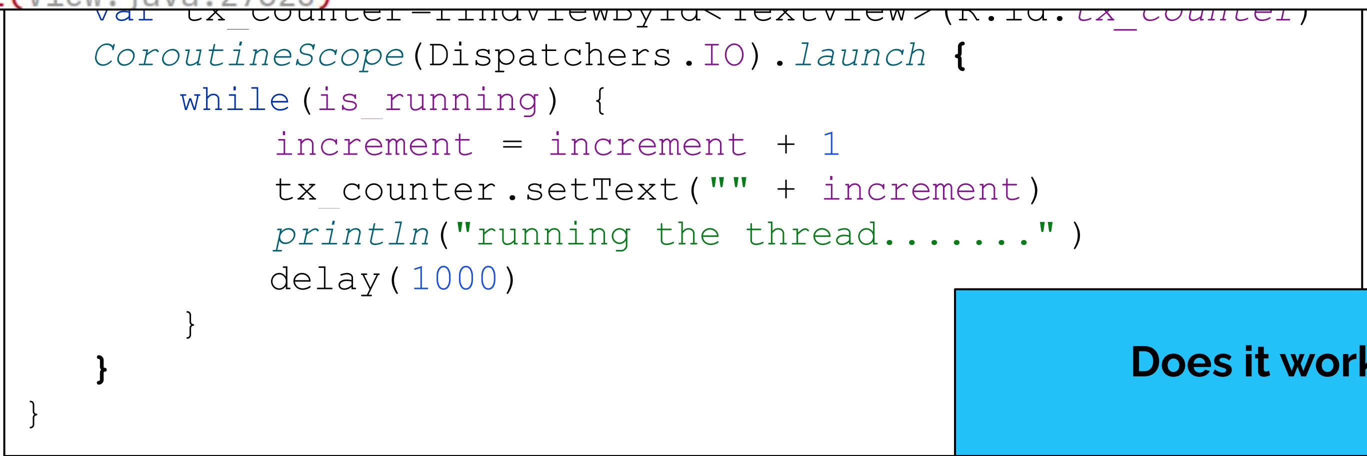
```
private fun startCounting(){
    var tx_counter=findViewById<TextView>(R.id.tx_counter)
    CoroutineScope(Dispatchers.IO).launch {
        while(is_running) {
            increment = increment + 1
            tx_counter.setText("" + increment)
            println("running the thread.......")
            delay(1000)
        }
    }
}
```

Does it work ?

Asynchronous Tasks & Coroutines



```
er-worker-1
ncrementer, PID: 8814
FromWrongThreadException: Only the original thread that created a view hierarchy can touch its
.checkThread(ViewRootImpl.java:11586)
.requestLayout(ViewRootImpl.java:2648)
Layout(View.java:27623)
Layout(View.java:27623)
```



```
    val tx_counter=findViewById<TextView>(R.id.tx_counter)
    CoroutineScope(Dispatchers.IO).launch {
        while(is_running) {
            increment = increment + 1
            tx_counter.setText("" + increment)
            println("running the thread.......")
            delay(1000)
        }
    }
}
```

Does it work ? NO

Asynchronous Tasks & Coroutines

- **AutoIncrementer App using Coroutines**
  - Launching Coroutines :

```
private fun startCounting(){
    var tx_counter=findViewById<TextView>(R.id.tx_counter)
    CoroutineScope(Dispatchers.IO).launch {
        while(is_running) {
             increment = increment + 1
             withContext(Dispatchers.Main) {
                 tx_counter.setText("" + increment)
             }
             println("running the thread.......")
             delay(1000)
        }
    }
}
```

Asynchronous Tasks & Coroutines

- **AutoIncrementer App using Coroutines**
  - Laun

Consider always the use of Progressbar or loading... when necessary...

```
private f
    var tx
    Corout
        wh

              tx_counter.setText("" + increment)
           }
           println("running the thread.......")
           delay(1000)
        }
    }
}
```

Data Persistence

Ways of storing data

- Data can be stored for mobile apps using :
  - Shared Preferences
  - Local Databases
  - As Files in the filesystem
  - Cloud Services :
    - Firebase (To be seen fully with Flutter)
    - AWS + ...

Data Persistence

Shared Preferences

- **Shared Preferences :**
  - It is a way to store primitive data in the form **key:value** using the class ***SharedPreferences***
  - It is recommended to use it for small data
  - Android keeps Shared Preferences in XML file format. The file is called "shared_prefs" that can be accessed at: **Data/data/{application package}**
  - Examples of data that can be stored inside the shared preferences include *App or user settings.*

Data Persistence
Shared Preferences

- **Creating SharedPreference File**
    - Need to specify a given file name in addition to the security mode
    - The editor Object must be initialized to write data.

```
var PREFS_NAME="DATA_INCREMENT"
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView( R.layout.activity_main)

       var sharedpreferences = getSharedPreferences( PREFS_NAME, Context.MODE_PRIVATE)
       val editor: SharedPreferences.Editor = sharedpreferences.edit()
```

Data Persistence
Shared Preferences

- **Creating SharedPreference File**
  - Access Levels of SharedPreferences :
    - There are three levels of access for shared data :
      - *Activity-Level* : **getPreferences()**
      - *Application-Level* : **getSharedPreferences()** **(Recommended to use)**
      - *Android-Level* : **getDefaultSharedPreferences()**

```
var sharedpreferences  = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE)
```

Data Persistence

Shared Preferences

- **Creating SharedPreference File**
  - Security and Private of the sharedPreference File :
    - MODE_PRIVATE (Default)
    - MODE_WORLD_READABLE
    - MODE_WORLD_WRITEABLE
    - .

MODE_WORLD_* are deprecated, use other ways to share data

```
var sharedpreferences = getSharedPreferences( PREFS_NAME,
Context.MODE_PRIVATE)
```

Data Persistence

Shared Preferences

- **Storing Data using Shared Preferences :**
  - To store data in the format : key - value

```
editor.putInt("increment",increment)
editor.putFloat("price",floatVar)
editor.putString("Today","Monday")

editor.commit()
```

  - Commit (or apply()) must be called to save the data.

Data Persistence

Shared Preferences

- **Storing Data using Shared Preferences :**
    - To remove a variable from the sharedPreference storage:

```
editor.remove("increment")
editor.commit()
```

    - To remove all data :

```
editor.clear()
editor.commit()
```

Data Persistence
Shared Preferences

- **Getting Data using Shared Preferences :**
    - The getter method of the sharePreferences are used depending on the type : getInt , getString, getFloat…

```
var sharedpreferences = getSharedPreferences( PREFS_NAME,
Context.MODE_PRIVATE)
increment=sharedpreferences.getInt("increment",0)
var day=sharedpreferences.getString("Today","Sunday")
```

Data Persistence

Shared Preferences

- **Example : Auto Incrementing App**

Where to place the code for :

- Initializing the Shared preferences ?
- Saving the data ? After each thread invocation ?
- Loading the data ?

Data Persistence

Shared Preferences

Does it work ?

- **Example : Auto Incrementing App**

```
var sharedpreferences : SharedPreferences = getSharedPreferences(PREFS_NAME,
Context.MODE_PRIVATE)
var editor: SharedPreferences.Editor = sharedpreferences

override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   setContentView(R.layout.activity_main)


   increment=sharedpreferences.getInt("increment",0)
   var day=sharedpreferences.getString("Today","Sunday")

   findViewById<Button>(R.id.tx_counter).text=""+increment
```

Data Persistence

Shared Preferences

- **Example : Auto Incrementing App**

```
lateinit var sharedpreferences : SharedPreferences
lateinit var editor: SharedPreferences.Editor

override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   setContentView(R.layout.activity_main)

   sharedpreferences = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE)
   editor = sharedpreferences.edit()


   increment=sharedpreferences.getInt("increment",0)
   var day=sharedpreferences.getString("Today","Sunday")

   findViewById<Button>(R.id.tx_counter).text=""+increment
```

Data Persistence

Shared Preferences

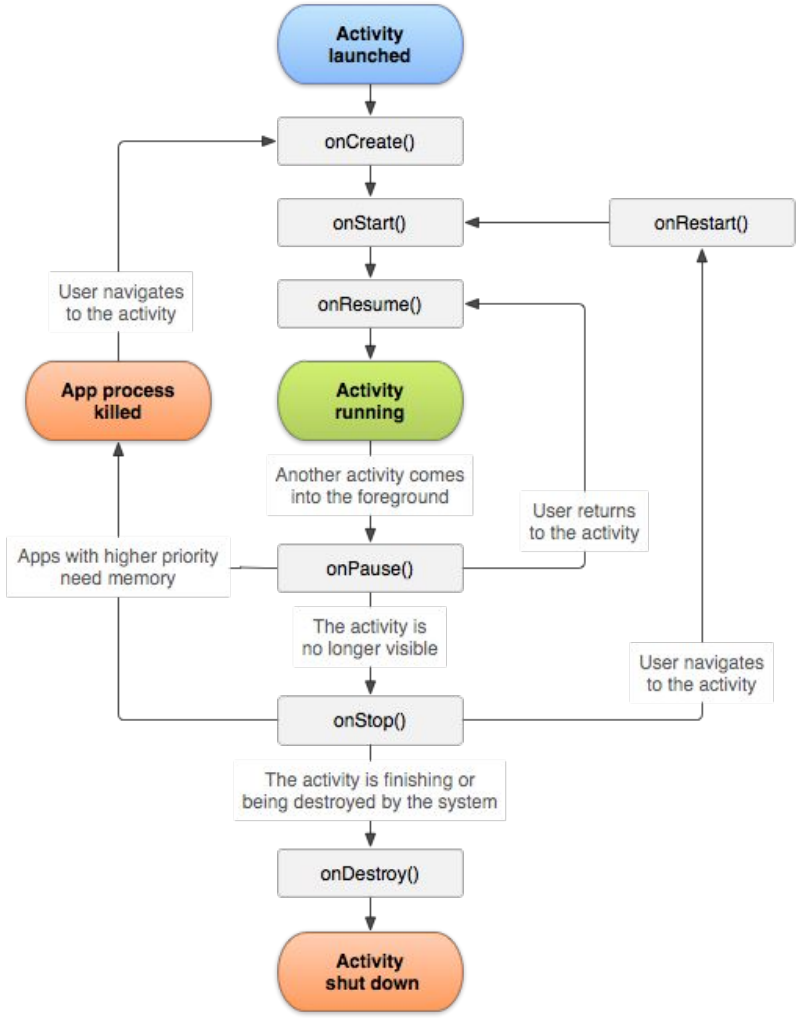
- Example : Auto Incrementing App

When or when to save the data ?

Data Persistence

Shared Preferences

- **Example : Auto Incrementing App**
  - When or Where to save the data ?
    - Lifecycle of the Activity


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Data Persistence

Shared Preferences

- **Example : Auto Incrementing App**
  - The lifecycle functions can be used to inject the code for saving data :

```
override fun onStop() {
    super.onStop()

    var sharedpreferences = getSharedPreferences( PREFS_NAME,
Context.MODE_PRIVATE)
    var editor: SharedPreferences.Editor = sharedpreferences.edit()
    editor.putInt("increment",increment)
    editor.putString("Today","Monday")
    editor.commit()
}
```

Data Persistence

Relational Embedded Databases

- **Relational Databases : SQLite:**
  - SQLite is a well-regarded SQL-based relational database management system (RDBMS). It is
    - Open source
    - Standards-compliant, implementing most of the SQL standard
    - Lightweight
    - Single-tier
    - ACID compliant

Data Persistence

Relational Embedded Databases

- **Relational Databases : SQLite:**
  - SQLite is implemented as a compact C library that's included as part of the Android software stack
  - Each SQLite database is an integrated part of the application that created it. This reduces external dependencies, minimizes latency, and simplifies transaction locking and synchronization.
  - Android databases are stored in the **/data/data/<package_name>/databases folder** on your device (or emulator).

Data Persistence

Relational Embedded Databases

- **SQL Reminder : Creating Tables**

```
CREATE TABLE IF NOT EXISTS  expenses  (
   expense id INTEGER PRIMARY KEY AUTOINCREMENT,
   name TEXT  NOT NULL,
   price REAL NOT NULL,
   date REAL NOT NULL,
   image BLOB NULL
);
```

Data Persistence

Relational Embedded Databases

- **SQL Reminder : Creating Tables**
  - Data Types :
    - VARCHAR(N)
    - TEXT
    - INT
    - LONG
    - DATE
    - ENUM ..

Data Persistence

Relational Embedded Databases

- **SQL Reminder : Searching and Retrieving Data**

```
SELECT table_name.column1,...FROM table_name WHERE table_name.column1>1
```

```
SELECT table_name.column1,...FROM table_name , table_two WHERE
          table_name.foreign_id=table_two.id AND
           table_name.column1>1
```

```
SELECT table_name.column1,...FROM table_name
                    LEFT JOIN table_two ON table_name.foreign_id=table_two.id
          WHERE  table_name.column1>1  ORDER BY table_name.column DESC  LIMIT 10
```

Data Persistence

Relational Embedded Databases

- **SQL Reminder : Updating Data**

```
UPDATE table_name SET
        column_name1='VALUE',
        column_name2='another VALUE',
WHERE
        column_name5='some value'
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
    - Adding Permission inside the **AndroidManifest.xml** File

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - DB Helper

```kotlin
package dev.startsoftware.simpleexpenseappnodb

import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class DBHelper(context: Context, factory: SQLiteDatabase.CursorFactory?) :
    SQLiteOpenHelper(context, DATABASE_NAME, factory, DATABASE_VERSION) {

    companion object{
        private val DATABASE_NAME = "EXPENSE_APP"
        private val DATABASE_VERSION = 1
    }

}
```

```
package dev.startsoftware.simpleexpenseappnodb

import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class DBHelper(context: Context, factory: SQLiteDatabase.CursorFactory?) :
   SQLiteOpenHelper(context, DATABASE_NAME, factory, DATABASE_VERSION) {

   companion object{
       private val DATABASE_NAME = "EXPENSE_APP"
       private val DATABASE_VERSION = 1
   }

   override fun onCreate(db: SQLiteDatabase) {
       val query = ("""
           SQL HERE
       """.trimIndent())
       db.execSQL(query)
   }
   override fun onUpgrade(db: SQLiteDatabase, p1: Int, p2: Int) {
      db.execSQL("DROP TABLE IF EXISTS expenses");
      db.execSQL("or instead, alter some data..");
      onCreate(db)
   }

}
```

```kotlin
package dev.startsoftware.simpleexpenseappnodb

import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class DBHelper(context: Context, factory: SQLiteDatabase.CursorFactory?) :
   SQLiteOpenHelper(context, DATABASE_NAME, factory, DATABASE_VERSION) {

   companion object{
       private val DATABASE_NAME = "EXPENSE_APP"
       private val DATABASE_VERSION = 1
   }

   override fun onCreate(db: SQLiteDatabase) {
       val query = ("""
            CREATE TABLE expenses IF NOT EXISTS (
               expense_id INTEGER PRIMARY KEY AUTOINCREMENT,
               name TEXT,
               price REAL,
               image BLOB
            )
       """.trimIndent())
       db.execSQL(query)
   }
   override fun onUpgrade(db: SQLiteDatabase, p1: Int, p2: Int) {
      db.execSQL("DROP TABLE IF EXISTS expenses");
      db.execSQL("or instead, alter some data..");
      onCreate(db)
   }
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
    - Initialize the DBHelper Instance

```
class MainActivity : AppCompatActivity() {
   companion object {
        var data = Vector<Expense>()
   }
```

```
class MainActivity : AppCompatActivity() {

   companion object {
        lateinit var db: DBHelper ;
   }
   var data = Vector<Expense>()

   override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        db = DBHelper(this, null)
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Inserting an Expense
    - Inside the Expense Activity

```
class NewExpense : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       bt_add.setOnClickListener {
           ...
           MainActivity.db.insertExpense(Expense(nameVal,dateVal,priceVal))
           ...
           this.setResult(RESULT_OK, intent)
           this.finish()
       }
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
    - Inserting an Expense
        - Inside the DBHelper Class :

```
class DBHelper(context: Context, factory: SQLiteDatabase.CursorFactory?) … {
   fun insertExpense(expense : Expense) : Boolean{
       try {
           val db = this.writableDatabase
           val values = ContentValues()
           values.put("name",expense.name)
           values.put("price",expense.price)
           values.put("date",expense.date)
           db.insert("expenses", null, values)
       }catch (e: Exception){  return false     }
       return true
   }
```

Data Persistence
Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Retrieving all Expenses
    - Inside the Main Activity :

```
private fun drawExpense(){
    var lv_expenses=findViewById<ListView>(R.id.lv_expenses)
    var data =    db.getExpenses()
    lv_expenses.adapter=ListExpenseAdapter(this,data)
    lv_expenses.refreshDrawableState()
}
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Retrieving all Expenses
    - Inside the DBHelper Class:

```
fun getExpenses() : Vector<Expense> {
    var data=Vector<Expense>()
    val db = this.readableDatabase
    val res = db.rawQuery("select * from expenses ", null)
    res.moveToFirst()
    while (res.isAfterLast == false) {
        data.add(Expense(
            res.getString(res.getColumnIndex("name").toInt()),
            res.getString(res.getColumnIndex("date").toInt()),

res.getDouble(res.getColumnIndex("price").toInt()),
        ))
        res.moveToNext()
    }
    return data
}
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Retrieving all Expenses
    - Inside the Main Activity :

```
class MainActivity : AppCompatActivity() {
   companion object {
       lateinit var db: DBHelper ;
   }

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)

       db = DBHelper(this, null)
       drawExpense()
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - How to update when DB is used ? Previous code using Vector :
    - Main Activity :

```
lv_expenses.setOnItemClickListener { parent, view, position, id ->
   val intent = Intent(this, EditExpense::class.java)
   intent.putExtra("expense_index", position)
   launchActivityNewExpense.launch(intent)
}
```

    - EditExpense Activity :

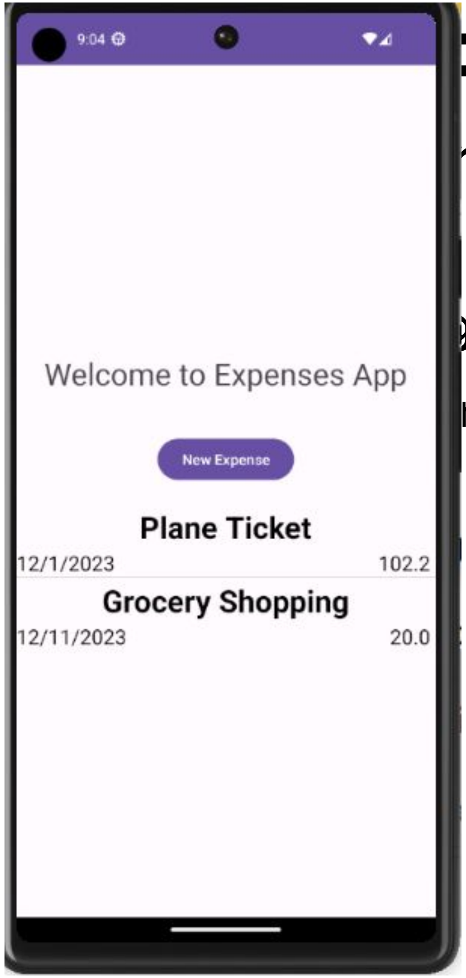
```
var expense : Expense?=null

if (extras != null) {
   position_id = extras.getInt("position_id")
   expense=MainActivity.data.get(position_id) as Expense
```

Persistence

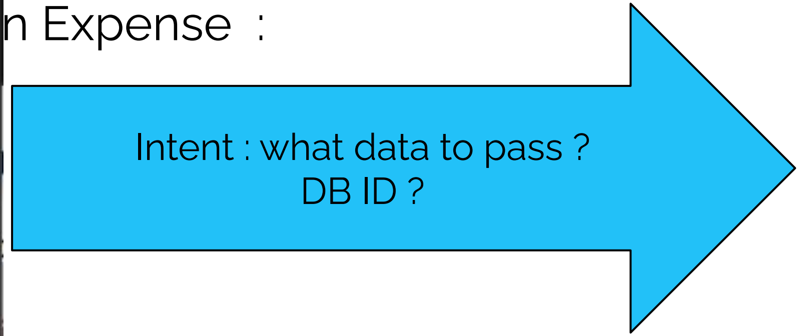
mbedded Databases

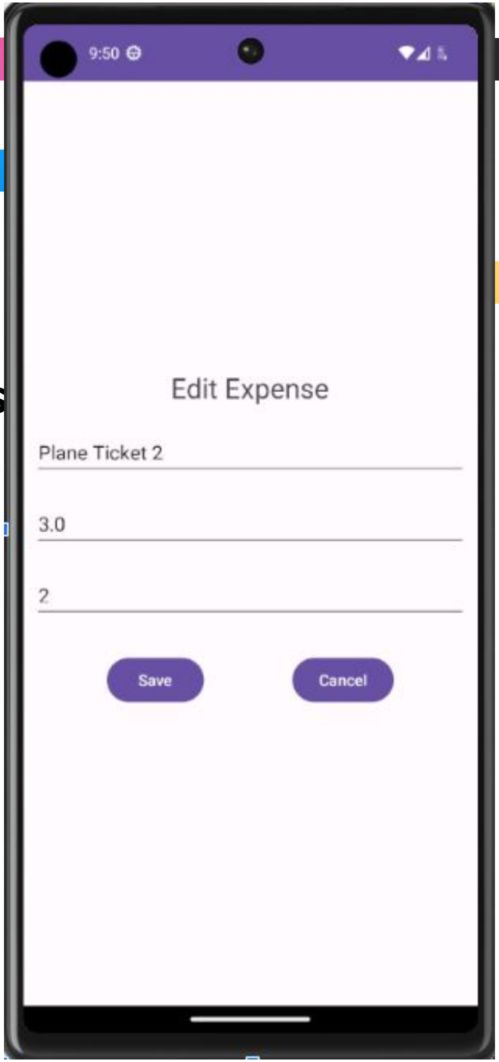
grating the database for the Expens

n Expense :



Plane Ticket
12/1/2023 102.2
Grocery Shopping
12/11/2023 20.0


Intent : what data to pass ?
DB ID ?



Plane Ticket 2
3.0
2

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Updating an Expense

    -
      ```
      lv_expenses.setOnItemClickListener { parent, view, position, id ->
          val intent = Intent(this, EditExpense::class.java)
          intent.putExtra("expense_index", position)
          launchActivityNewExpense.launch(intent)
      }
      ```

```
lv_expenses.setOnItemClickListener { parent, view, position, id ->
    val intent = Intent(this, EditExpense::class.java)
    var expense_db_id = ...
    intent.putExtra("expense_db_id", expense_db_id)
    launchActivityNewExpense.launch(intent)
}
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Updating an Expense
    - Adding db_id into the Expense Class
      - Default value is zero

```
class Expense(var name:String,var date:String,var price:Double,var
db_id:Int=0)
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
    - Updating an Expense
        - When retrieving all expenses inside the DBHelper

```
fun getExpenses() : Vector<Expense> {
   var data=Vector<Expense>()
   val db = this.readableDatabase
   val res = db.rawQuery("select * from expenses ", null)
   res.moveToFirst()
   while (res.isAfterLast == false) {
       data.add(Expense(
           res.getString(res.getColumnIndex("name").toInt()),
           res.getString(res.getColumnIndex("date").toInt()),
           res.getDouble(res.getColumnIndex("price").toInt()),
          db_id=res.getInt(res.getColumnIndex("expense_id").toInt())
       ))
       res.moveToNext()
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Updating an Expense
    - Inside the Main Activity, DB_ID of the Expense is sent with the intent

```
lv_expenses.setOnItemClickListener { parent, view, position, id ->
   val intent = Intent(this, EditExpense::class.java)
   var expense_db_id=data[position].db_id
   intent.putExtra("expense_db_id", expense_db_id)
   launchActivityNewExpense.launch(intent)
}
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Updating an Expense
    - Inside the EditExpense Activity

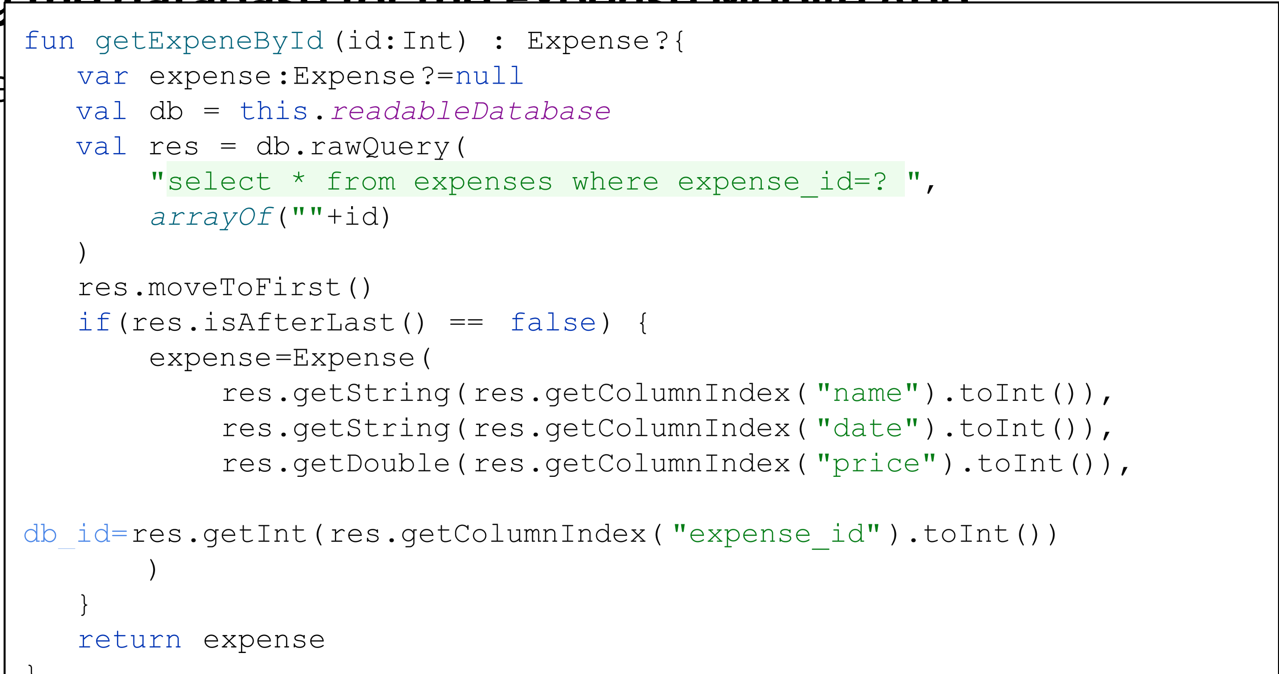
```
if (extras != null) {
   position_id = extras.getInt("position_id")
   expense=MainActivity.data.get(position_id) as Expense
```

```
if (extras != null) {
   expense_db_id = extras.getInt("expense_db_id")
   expense=MainActivity.db.getExpeneById(expense_db_id)
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Updating an Expe
    - Inside the DBHelper Class:



```
fun getExpeneById(id:Int) : Expense?{
    var expense:Expense?=null
    val db = this.readableDatabase
    val res = db.rawQuery(
        "select * from expenses where expense_id=? ",
        arrayOf(""+id)
    )
    res.moveToFirst()
    if(res.isAfterLast() == false) {
        expense=Expense(
            res.getString(res.getColumnIndex("name").toInt()),
            res.getString(res.getColumnIndex("date").toInt()),
            res.getDouble(res.getColumnIndex("price").toInt()),

db_id=res.getInt(res.getColumnIndex("expense_id").toInt())
        )
    }
    return expense
}
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Updating an Expense
    - This is the old code when clicking the button save

```
bt_save.setOnClickListener {
    expense?.name=findViewById<EditText>(R.id.tx_description).text.toString()
    expense?.date=findViewById<EditText>(R.id.tx_date).text.toString()
    expense?.price=findViewById<EditText>(R.id.tx_amount).text.toString().toDouble()
    val intent = Intent()
    this.setResult(RESULT_OK, intent)
    this.finish()
}
```

We have to save to a database

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Updating an Expense
    - This is the old code when clicking the button save

```
bt_save.setOnClickListener {
    expense?.name=findViewById<EditText>(R.id.tx_description).text.toString()
    expense?.date=findViewById<EditText>(R.id.tx_date).text.toString()
    expense?.price=findViewById<EditText>(R.id.tx_amount).text.toString().toDouble()
    MainActivity.db.saveExpense(expense)
    val intent = Intent()
    this.setResult(RESULT_OK, intent)
    this.finish()
}
```

Data Persistence

Relational Embedded Databases

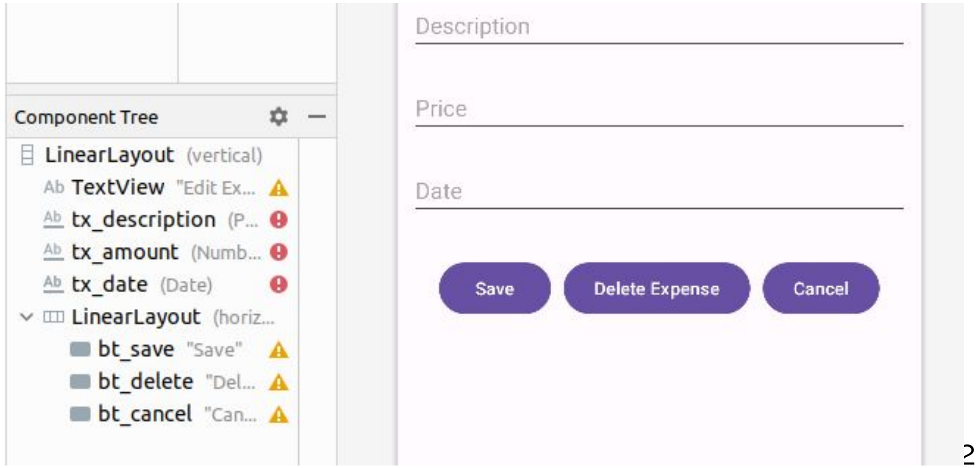
- **Example : Integrating the database for the Expense Mobile App**
  - Updating an Expense
    - DBHelper Class

```
fun saveExpense(expense: Expense): Boolean{
   try {
       val db = this.writableDatabase
       val values = ContentValues()
       values.put("name",expense.name)
       values.put("price",expense.price)
       values.put("date",expense.date)
       db.update(
           "expenses",
           values,
           "expense_id=?",
           arrayOf(""+expense.db_id));
   }catch (e: Exception){
       return false
   }
   return true
}
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Deleting an Expense
    - Inside the XML for the EditExpense
      - Add a Button **bt_delete**



Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Delete an Expense
    - EditExpense Activity

```
var bt_delete=findViewById<Button>(R.id.bt_delete)
bt_delete.setOnClickListener {
    MainActivity.db.deleteExpenseById((expense as Expense).db_id )
    val intent = Intent()
    this.setResult(RESULT_OK, intent)
    Toast.makeText(this, "Expense is deleted successfully" , Toast.LENGTH_SHORT).show()
    this.finish()
}
```

Data Persistence

Relational Embedded Databases

- **Example : Integrating the database for the Expense Mobile App**
  - Delete an Expense
    - DBHelper Class :

```
fun deleteExpenseById(id:Int){
   val db = this.writableDatabase
   db.delete("expenses",
        "expense_id = ? ",
        arrayOf(""+id));
}
```

Data Persistence

Relat

- Exam
  - D

For Simplicity :
I am dumping all the functions related to getting Expenses Data into the DBHelper

What happens if we have other entities : Categories ? Users ? ...

You need to create a utility class for each entity and write the associated mapping function to conduct data logic.

Data Persistence

Relational Embedded Databases

- **Tedious Programming Style**
  - As a result, whenever you wish to store data in an SQLite table, you must first extract the data stored as variables within each object, and convert them into a row of values according to the columns of your table (using Content Values).
  - Similarly, when extracting data from the table, you receive one or more rows of values (as a Cursor), which must be translated into one or more objects.

Data Persistence

Relational Embedded Databases

- **Object Relational Mapping : ORM**
  - It is a technique where relational database rows are accessed inside a programming language as objects where each row is visualized as an object.
    - Column for a row, is the instance variable for the corresponding object.
    - Upon modifying the value for an instance variable, the mapped column in the database table for the corresponding row is updated automatically.
    - Creating an object will insert a row automatically into the table.
    - In short, Rare use of SQL whilst OOP is used instead.

Data Persistence

Relational Embedded Databases

- **ROOM as an ORM over SQLite**
  - Room is a persistence library that simplifies the process of adding a structured SQL database to your app.
  - Room provides an abstraction layer as an ORM over an SQLite backend, making it easier to define and access a database for your app's structured data, while still offering the full power of SQLite.

Data Persistence

Relational Embedded Databases

- **ROOM as an ORM over SQLite**
  - The Room persistence model requires you to define three components:
    - **Entity** : One or more classes, annotated with the **@Entity** annotation, which define the structure of a database table that will be used to store instances of the annotated class.
    - **Data Access Object (Dao)**—A class annotated with the @Dao annotation that will define the methods used to modify or query the database.
    - **Room Database**—An abstract class annotated with the @Database annotation that extends RoomDatabase. This class is the main access point for the underlying SQLite connection

Data Persistence

Relational Embedded Databases

- **ROOM as an ORM over SQLite**
  - Adding the dependencies :

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'kotlin-kapt'
}
...
```

```
dependencies {

     ...

   def room_version = "2.3.0"

   implementation "androidx.room:room-runtime: $room_version"
   kapt "androidx.room:room-compiler: $room version"
   annotationProcessor "androidx.room:room-compiler: $room_version"
   implementation "androidx.room:room-ktx: $room_version"

   implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.4.2'
   implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.4.2'


   def lifecycle version = "2.3.1"
   implementation
"androidx.lifecycle:lifecycle-runtime-ktx: $lifecycle_version"

}
```

Data Persistence

Relational Embedded Databases

- **ROOM as an ORM over SQLite**
  - Creating the Entity Data Class = Creating the Table
    - **ExpenseEntity.kt**

```
import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "expenses")
data class ExpenseEntity(
   @PrimaryKey(autoGenerate = true)
   var expense_id: Int,

   var name: String,
   var date: String,
   var price: Double,
)
```

Data Persistence

Relational Embedded Databases

- **ROOM as an ORM over SQLite**
  - Creating the DAO Interface (Methods)
    - **ExpenseDao.kt**

```
@Dao
interface ExpenseDao {
    @Insert
    fun insertExpense(expense: ExpenseEntity)

    @Query("SELECT * FROM expenses ")
    fun getAllExpenses(): List<ExpenseEntity>

    @Update
    fun updateExpense(expense:ExpenseEntity)

    @Delete
    fun deleteExpense(expense: ExpenseEntity)
}
```

Data Persistence

Relational Embedded Databases

- **ROOM as an ORM over SQLite**
  - Creating the Database Class (Methods)
    - **ExpenseDatabase.kt**
      - We list all entitled + Methods to instances of the Dao of each table

```
@Database(entities = [ExpenseEntity::class], version = 1)
abstract class ExpenseDatabase : RoomDatabase() {
   abstract fun expenseDao(): ExpenseDao
}
```

Data Persistence

Relational Embedded Databases

- **ROOM as an ORM over SQLite**
  - Creating the Database Class (Methods)
    - **Initializing the ROOM**

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    db = Room.databaseBuilder(
        applicationContext,
        ExpenseDatabase::class.java, "simple_database_with_room_section1"
    ).build()
    expenseDao = db.expenseDao()
```

Data Persistence

Relational Embedded Databa

- **ROOM as an ORM over SQLite**
  - Creating the Database Class (Met
    - **Adding an Expense**

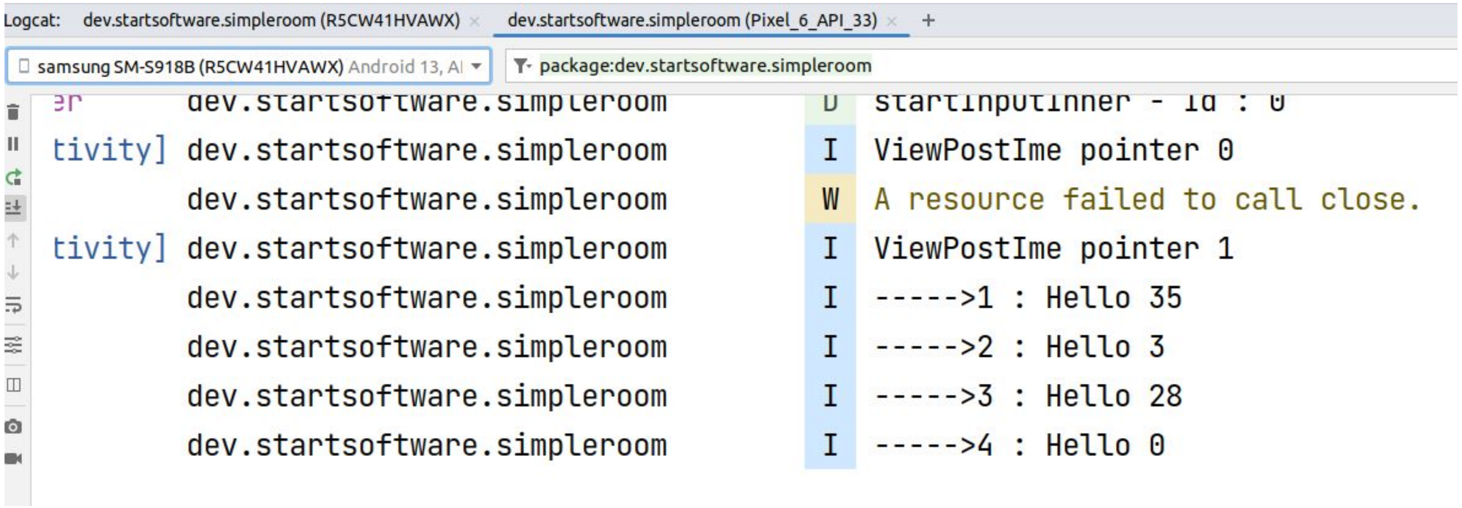
```
var bt_add=findViewById<Button>(R.id.bt_add)
bt_add.setOnClickListener {
    CoroutineScope(Dispatchers.IO).launch {
        val result =  coroutine_insertExpense()
        onResultInsertExpense(result)
    }
}
```

```
suspend fun coroutine_insertExpense():Unit{
    var nameVal="Hello "+(0..100).random()
    var dateVal="2023/11/22"
    var priceVal=(0..100).random().toDouble()
    var ret=expenseDao.insertExpense(
        ExpenseEntity(0, nameVal, dateVal,
priceVal)
    )
    return ret
}

suspend fun
coroutine_listExpense():List<ExpenseEntity>{
    var ret=expenseDao.getAllExpenses()
    for (item in ret){
        println("----->"+item.expense_id+" :
"+item.name)
    }
    return ret
}

fun onResultInsertExpense(result: Unit) {}
```

Data Persistence

Relational Embedded Databases

- **ROOM as an ORM over SQLite**
  - Creating Objects = Inserting Rows into the database Table



Logcat: dev.startsoftware.simpleroom (R5CW41HVAWX) × dev.startsoftware.simpleroom (Pixel_6_API_33) × +
samsung SM-S918B (R5CW41HVAWX) Android 13, AI
package:dev.startsoftware.simpleroom

```
er        dev.startsoftware.simpleroom        D  startInputInner - Id : 0
tivity]   dev.startsoftware.simpleroom        I  ViewPostIme pointer 0
          dev.startsoftware.simpleroom        W  A resource failed to call close.
tivity]   dev.startsoftware.simpleroom        I  ViewPostIme pointer 1
          dev.startsoftware.simpleroom        I  ----->1 : Hello 35
          dev.startsoftware.simpleroom        I  ----->2 : Hello 3
          dev.startsoftware.simpleroom        I  ----->3 : Hello 28
          dev.startsoftware.simpleroom        I  ----->4 : Hello 0
```

Using Internet Resources
Accessing API/Web

- Accessing the web shall be done using
  - Coroutines.
  - HTTP libraries :
    - Retrofit Library
    - Or simply :
      - **val apiResponse = URL("yourUrl").readText()**

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Lecture Demo Apps

- Auto Incrementer with Coroutines :
    - https://www.dropbox.com/scl/fo/6ol8rzpjjt3hm2cx5jigx/h?rlkey=wwt0zk7zo429zeip5g2pimo98&dl=0
- Showing and Hiding Progressbar using Coroutines to simulate heavy tasks
    - https://www.dropbox.com/scl/fo/iyopjlh4lrp11xlqniaew/h?rlkey=1sb17a81bgfav00uq07qjml4z&dl=0
- Auto Incrementer with SharedPreferences and Coroutines
    - https://www.dropbox.com/scl/fo/mzoyipgyvnsovo95rxjsr/h?rlkey=nergvguxfuf9gsxmovphci754&dl=0
- Expense App with a Database SQLite
    - https://www.dropbox.com/scl/fo/sqke9mr64ypmpqbry6kw5/h?rlkey=u4txf2hcy01hfhdvpuoe59oob&dl=0
- Simple Hello World for using ROOM with SQLite.
    - https://www.dropbox.com/scl/fo/teke1td4p7m9yrwxpq3jv/h?rlkey=kubznbzh8u82r43aknhwidmxp&dl=0

Resources

- https://www.sqlite.org/
- https://kotlinlang.org/docs/coroutines-guide.html
- https://developer.android.com/training/data-storage#pref
- https://www.geeksforgeeks.org/android-sqlite-database-in-kotlin/
- https://www.geeksforgeeks.org/json-parsing-in-android-using-volley-library-with-kotlin/
- https://developer.android.com/kotlin/coroutines/coroutines-adv
- https://www.fypsolutions.com/android/kotlin/kotlin-coroutines-for-network-call/
- https://www.geeksforgeeks.org/kotlin-coroutines-on-android/
- https://engineering.monstar-lab.com/en/post/2023/01/06/Introduction-to-Kotlin-Coroutines-for-Android/
- https://developer.android.com/codelabs/kotlin-coroutines#0
- https://github.com/android-java-kotlin/kotlin-coroutines-counter/blob/master/app/src/main/java/com/m7amdelbana/counter/MainActivity.kt

Next on Flutter

- Creating Beautiful Screens
- Creating Databases, Accessing the network,
- Background Services
- Using Firebase : Messaging, Storage..
- Machine Learning Toolkits
- Other Advanced Features.