IT2010 – Mobile Application Development
BSc (Hons) in Information Technology
2nd Year
Faculty of Computing
SLIIT
2023 - Tutorial

Coroutines and Database in Android

What are Coroutines?

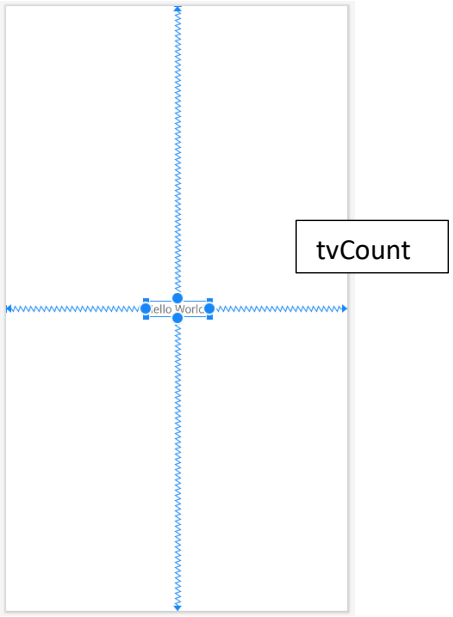
Coroutines are a way to write asynchronous code that looks and behaves like synchronous code. They were introduced in Kotlin 1.3 as an experimental feature and have since been stabilized. Coroutines allow you to write code that suspends and resumes execution at a later time. This can be useful when performing long-running operations like network requests or database operations.

Setting up coroutines in your project

1. Open build.gradle(Module:app)
2. Implement the dependency as follows

```
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.4'
```

3. Go to the xml file and add or modify the existing text view with a proper id



4. Then implement a suspend function as follows

```
suspend fun counter(view: TextView){
    var count = 0
    while(true){
        delay(1000)
        view.setText(count.toString())
        count ++
    }

}
```

5. After that implement the following in the onCreate method and observe the output

```
CoroutineScope(Dispatchers.Main).launch {
    tvCount = findViewById(R.id.tvCount)
    counter(tvCount)
}
```

Suspending Functions

A suspending function is a function that can suspend its execution and return control to the caller. This allows other code to be executed while the function is waiting for a result. To define a suspending function, you need to use the suspend modifier.

Coroutine Scope

A CoroutineScope in Kotlin is an interface that provides a way to manage the lifecycle of coroutines. It is used to launch coroutines and to manage their cancellation.

A CoroutineScope is a context in which coroutines run. It provides a set of rules that define how coroutines should behave within that context. The scope is responsible for creating a coroutine context and managing its lifecycle.

When a coroutine is launched, it is tied to the context of the scope in which it was launched. This means that when the scope is cancelled, all the coroutines launched within that scope are cancelled as well.

Dispatchers

In Kotlin coroutines, Dispatchers is a class that provides a set of thread pools and thread contexts that are used to determine on which thread a coroutine should be executed.

A coroutine running on a background thread can be used to perform long-running or CPU-intensive tasks without blocking the UI thread, while a coroutine running on the UI thread can be used to update the user interface.

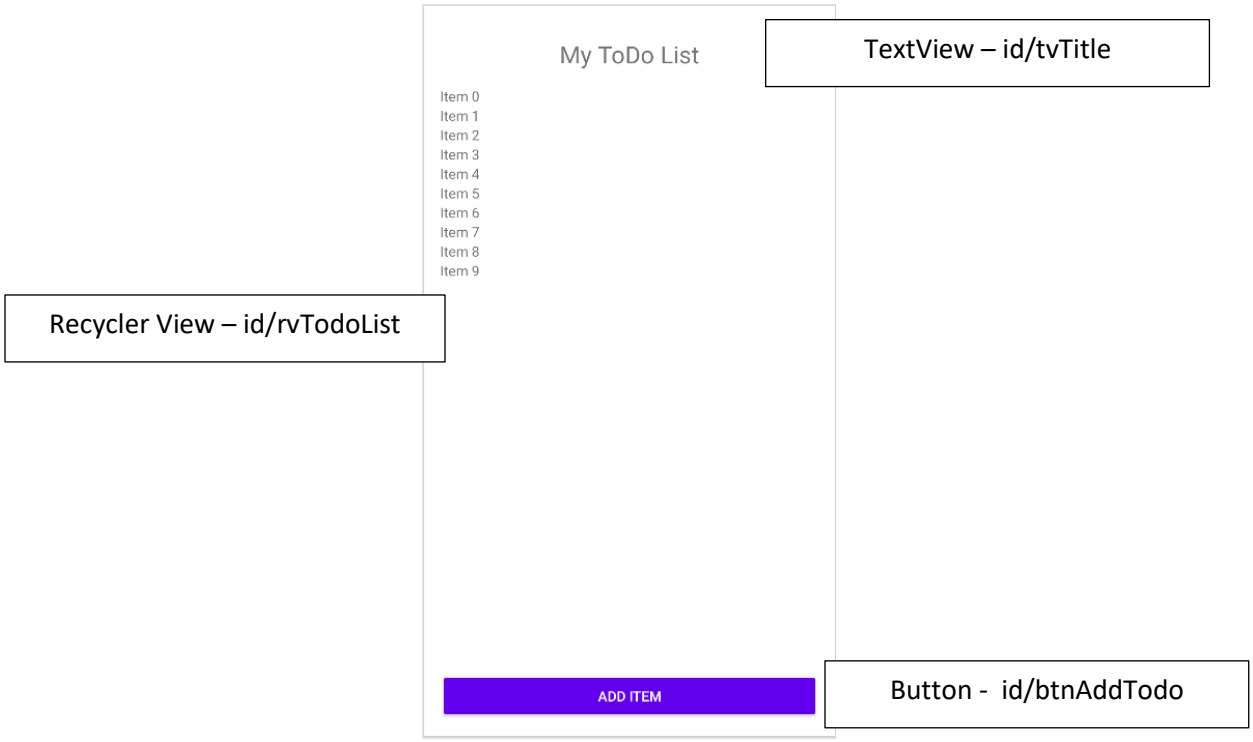
Here are the common dispatchers available in Kotlin coroutines:

- **Dispatchers.Default**: This dispatcher is used for CPU-intensive tasks. It uses a shared thread pool and creates as many threads as there are CPU cores available.
- **Dispatchers.IO:** This dispatcher is used for IO-bound tasks, such as reading from or writing to files or making network requests. It uses a shared thread pool that is optimized for IO operations and can create more threads than the number of available CPU cores.
- **Dispatchers.Main:** This dispatcher is used for performing operations on the UI thread, such as updating the user interface or launching coroutines that will interact with UI elements.
- **Dispatchers.Unconfined:** This dispatcher is used for coroutines that do not have a specific thread context, and can run on any thread, including the UI thread.

Database implementation with Room Persistence Library

Recycler View

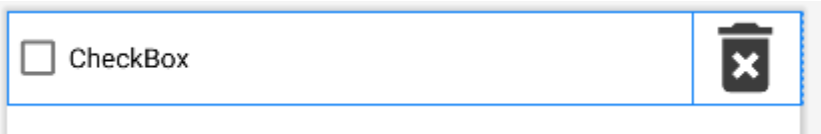
1. Remove all the previous codes and viewsDesign the following UI



2. Create a new package named ‘adapters’ in the main package
3. Create a class named TodoAdapter in it.
4. Extend the class with RecyclerView.Adapter class. And override the following methods

```
class TodoAdapter:RecyclerView.Adapter<ViewHolder>() {
   override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {

   }

   override fun getItemCount(): Int {

   }

   override fun onBindViewHolder(holder: ViewHolder, position: Int) {

   }

}
```

5. Create a new layout resource file in the layout directory named view_item
6. Convert the view to linear layout and design it like below



- Linear layout
    - Layout width – match parent
    - Layout height – 48dp
    - Id - todo_row_item
- Check Box
    - Layout width – wrap_content
    - Layout height – math_parent
    - Layout weight – 8
    - Id- cbTodo
- Image View
    - Layout width – wrap_content
    - Layout height – math_parent
    - Layout weight – 1
    - Id – ivDelete

7. Let's implement the Recycler View Adapter

```
class TodoAdapter:RecyclerView.Adapter<TodoAdapter.ViewHolder>() {
   class ViewHolder(view:View):RecyclerView.ViewHolder(view){
      val cbTodo:CheckBox
      val ivDelete:ImageView

      init {
         cbTodo = view.findViewById(R.id.cbTodo)
         ivDelete = view.findViewById(R.id.ivDelete)
      }
   }
   override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
      val view = LayoutInflater.from(parent.context)
         .inflate(R.layout.view_item,parent,false)
      return ViewHolder(view)
   }
   override fun onBindViewHolder(holder: ViewHolder, position: Int) {
      holder.cbTodo.text = "Sample Test"
   }
   override fun getItemCount(): Int {
      return 1
   }
}
```

8. Implement the main activity code as follows and run the program

```
val recyclerView:RecyclerView = findViewById(R.id.rvTodoList)
val adapter = TodoAdapter()
recyclerView.adapter = adapter
recyclerView.layoutManager = LinearLayoutManager(this)
```

Database implementation with room

1. Add the following dependencies

```
// Room
implementation "androidx.room:room-runtime:2.5.0"
kapt "androidx.room:room-compiler:2.5.0"

// Kotlin Extensions and Coroutines support for Room
implementation "androidx.room:room-ktx:2.5.0"
```

And in plugins add the following. Then click Sync now

```
id 'kotlin-kapt'
```

2. Create a package named database in the main directory.
3. Create a class named TodoDatabase
4. Before implementing it create new package named entities in the database directory
5. In entities, create a data class as follows

```
@Entity
data class Todo(
   var item: String?
) {
   @PrimaryKey(autoGenerate = true)
   var id: Int? = null
}
```

6. After that, create a new package named daos in the database directory

7. In the daos package, create an interface named TodoDao.

```
@Dao
interface TodoDao {
   @Insert
   suspend fun insertTodo(todo: Todo)

   @Delete
   suspend fun delete(todo: Todo)

   @Query("SELECT * From Todo")
   fun getAllUsers(): List<Todo>

}
```

8. Implement the TodoDatabase as follows

```
@Database(entities = [Todo::class], version = 1)
abstract class TodoDatabase:RoomDatabase(){

   abstract fun getTodoDao():TodoDao

   companion object{
      @Volatile
      private var INSTANCE: TodoDatabase? = null

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

      }
   }
}
```

9. Create a new package named repositories in the database directory.
10. Inside that create a class named TodoRepository and implement it as follows.

```
class TodoRepository(
private val db: TodoDatabase
) {
   suspend fun insert(todo: Todo) = db.getTodoDao().insertTodo(todo)
   suspend fun delete(todo: Todo) = db.getTodoDao().delete(todo)
   fun getAllTodos() =db.getTodoDao().getAllTodos()

}
```

11. After that modify the MainActivity onCreate method as follows

```
val repository = TodoRepository(TodoDatabase.getInstance(this))
val recyclerView: RecyclerView = findViewById(R.id.rvTodoList)
val ui = this
val adapter = TodoAdapter()
recyclerView.adapter = adapter
recyclerView.layoutManager = LinearLayoutManager(ui)

CoroutineScope(Dispatchers.IO).launch {
     val data = repository.getAllTodos()
     adapter.setData(data, ui)
}
```

12. Introduce an alert display method to be use add new Todo's to the list.

```
fun displayDialog(repository: TodoRepository, adapter: TodoAdapter) {
      // Create a new instance of AlertDialog.Builder
      val builder = AlertDialog.Builder(this)

      // Set the alert dialog title and message
      builder.setTitle("Enter New Todo item:")
      builder.setMessage("Enter the todo item below:")

      // Create an EditText input field
      val input = EditText(this)
      input.inputType = InputType.TYPE_CLASS_TEXT
      builder.setView(input)

      // Set the positive button action
      builder.setPositiveButton("OK") { dialog, which ->
         // Get the input text and display a Toast message
         val item = input.text.toString()
         CoroutineScope(Dispatchers.IO).launch {
            repository.insert(Todo(item))
            val data = repository.getAllTodos()
            runOnUiThread{
               adapter.setData(data,this@MainActivity)
```

```
            }
        }


    }

    // Set the negative button action
    builder.setNegativeButton("Cancel") { dialog, which ->
        dialog.cancel()
    }

    // Create and show the alert dialog
    val alertDialog = builder.create()
    alertDialog.show()
}
```

13. Implement the button click event for the add todo button

```
val btnAddTodo = findViewById<Button>(R.id.btnAddTodo)
btnAddTodo.setOnClickListener {
    displayDialog(repository)
}
```

14. Now let's modify the Todo Adapter

```
class TodoAdapter() : RecyclerView.Adapter<TodoAdapter.ViewHolder>() {
    lateinit var data: List<Todo>
    lateinit var context: Context

    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val cbTodo: CheckBox
        val ivDelete: ImageView

        init {
            cbTodo = view.findViewById(R.id.cbTodo)
            ivDelete = view.findViewById(R.id.ivDelete)

        }
    }

    fun setData(data: List<Todo>, context: Context) {
        this.data = data
        this.context = context
        notifyDataSetChanged()
    }
```

```
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.view_item, parent, false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.cbTodo.text = data[position].item
        holder.ivDelete.setOnClickListener {

            if(holder.cbTodo.isChecked){
                val repository = TodoRepository(TodoDatabase.getInstance(context))
                holder.ivDelete.setImageResource(R.drawable.clicked_delete_image)
                CoroutineScope(Dispatchers.IO).launch {
                    repository.delete(data[position])
                    val data = repository.getAllTodos()
                    withContext(Dispatchers.Main) {
                        setData(data, context)
                        holder.ivDelete.setImageResource(R.drawable.delete_image)
                    }
                }
            }else{

                Toast.makeText(context,"Cannot delete unmarked Todo items",Toast.LENGTH_LONG)
                    .show()
            }
        }
    }

    override fun getItemCount() = data.size
}
```