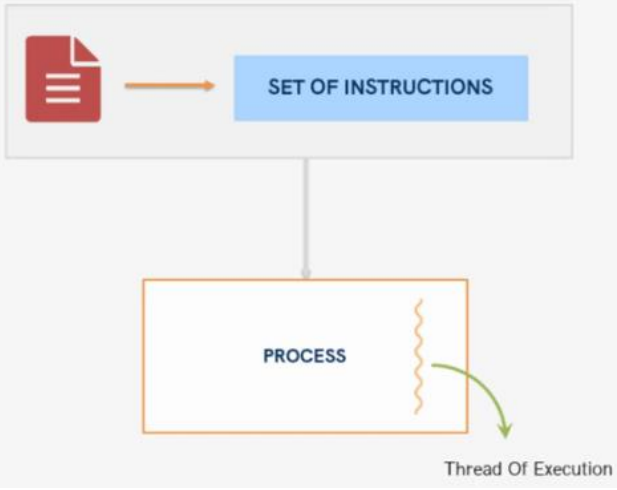
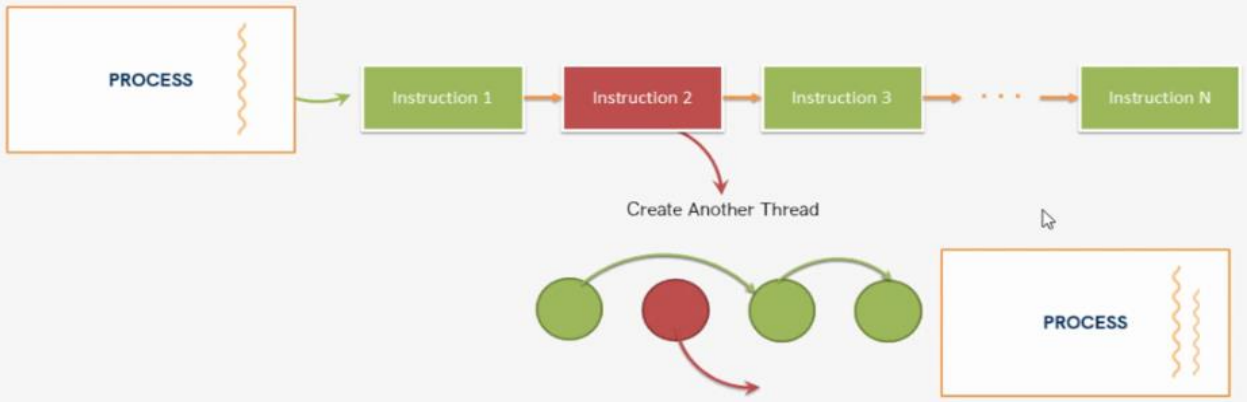
Kotlin Coroutine

Basic concepts:

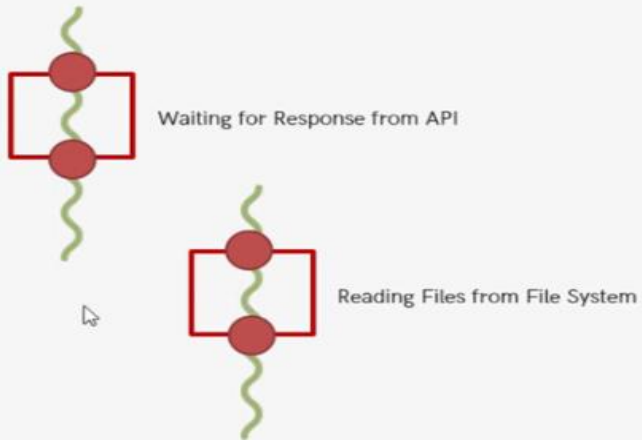
- Program
- Process (Actual instance of your program)
  - Process ID
  - State
  - Memory
  - Handles for Networking, File System etc.
  - Thread (i.e. Thread Of Execution)



- Sequential Execution

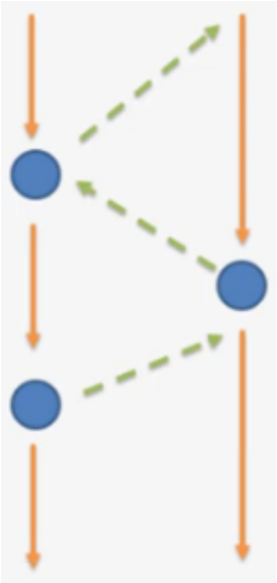


- Can we just re-use the thread when it is waiting for some response or IO operation?

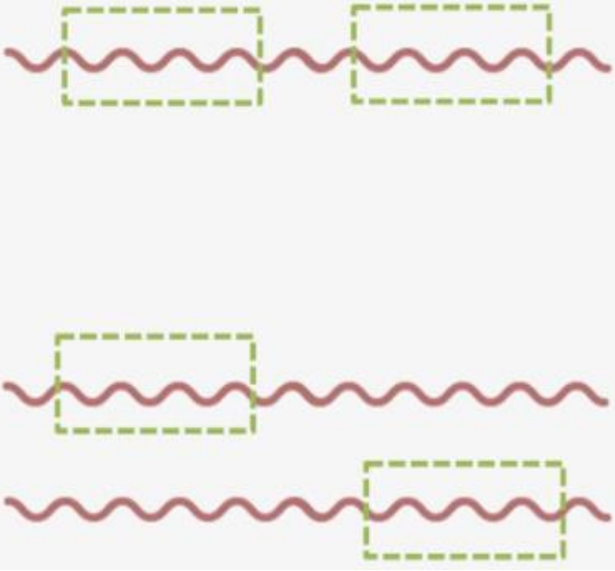


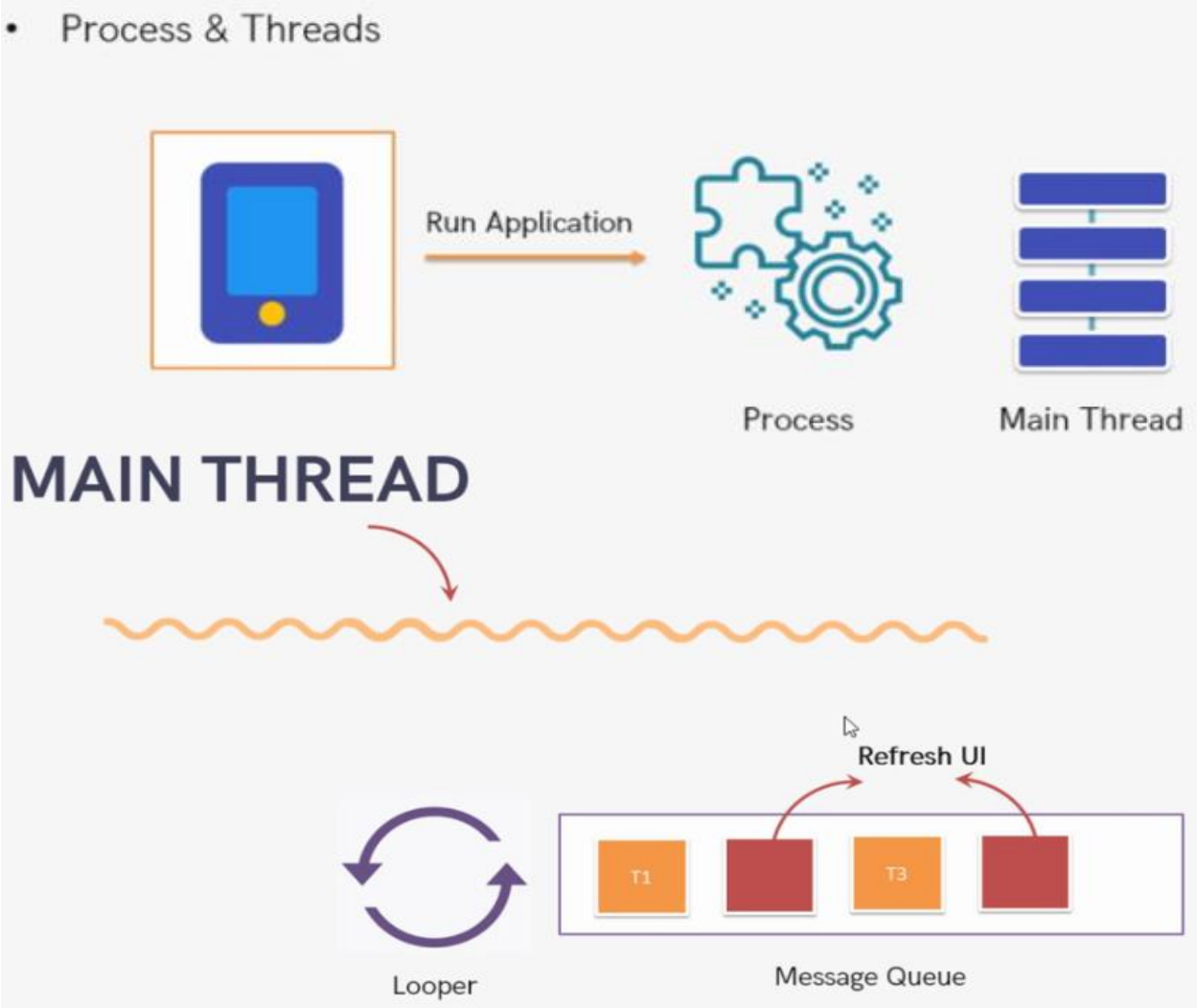
COROUTINES

- Executed inside a thread
- One thread can have many coroutines
- Cheap



Thread Pool





Problem: T3 will be hang.

Solution: Background threads

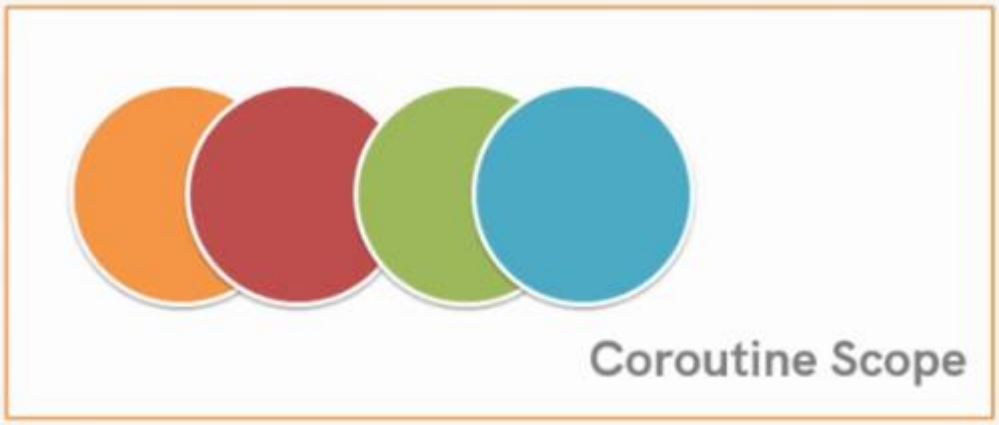
Thread limitations: (1)Memory space (2)Context switching

KOTLIN COROUTINES

- What's the solution in Java? – **No Solution**
- What's the solution in Kotlin? **Coroutines**
- Coroutines are just like threads (**lightweight threads**) but not threads.
- Coroutines run on top of Threads.

COROUTINES

- Coroutine Scope - Lifetime
- Coroutine Context - Threads

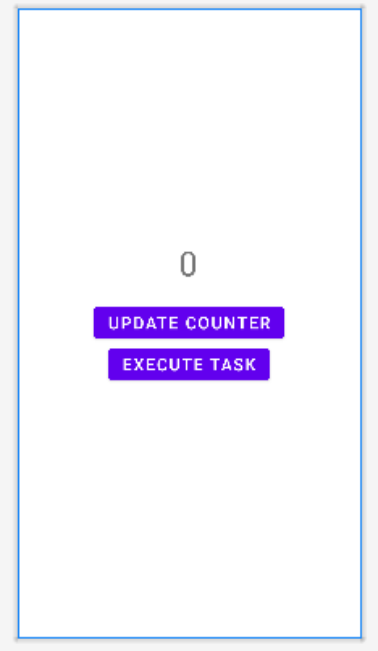


DISPATCHERS

- Coroutines run on top of threads.
- Dispatchers is a way to define threads on which Coroutines are executed.
- Predefined Dispatchers –
    - Dispatchers.IO
    - Dispatchers.Main
    - Dispatchers.Default

activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">
   <TextView
      android:id="@+id/counter"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="0"
      android:textSize="40sp"
      app:layout_constraintBottom_toTopOf="@+id/button"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintHorizontal_bias="0.498"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent"
      app:layout_constraintVertical_bias="0.934" />
   <Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="updateCounter"
      android:text="Update counter"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintHorizontal_bias="0.5"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent"
      app:layout_constraintVertical_bias="0.5" />
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="doAction"
      android:text="Execute task"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent"
      app:layout_constraintVertical_bias="0.571"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```



MainActivity.kt:

```
package com.ghani.kotlincoroutine

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.TextView
import kotlinx.coroutines.*
import kotlin.concurrent.thread

class MainActivity : AppCompatActivity() {

    lateinit var counterText:TextView
    private val TAG:String = "KOTLINFUN"

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

        counterText = findViewById(R.id.counter)
        Log.d(TAG,"${Thread.currentThread().name}")

    }
```

```
    fun updateCounter(view: View) {
        Log.d(TAG,"${Thread.currentThread().name}")
        counterText.text = "${counterText.text.toString().toInt()+1}"
    }

    private fun ExecuteLongRunningTask(){
        for (i in 1..1000000000L){

        }
    }

    /*
    //flow-1
    fun doAction(view: View) {
        ExecuteLongRunningTask()
    }
    */

    /*
    //flow-2
    fun doAction(view: View) {
        thread(start=true){
            ExecuteLongRunningTask()
        }
    }
    */

    //flow-3
    fun doAction(view: View) {
        CoroutineScope(Dispatchers.IO).launch {
            Log.d(TAG,"1 - ${Thread.currentThread().name}")
        }
        MainScope().launch(Dispatchers.Default) {
            Log.d(TAG,"2 - ${Thread.currentThread().name}")
        }
        GlobalScope.launch(Dispatchers.Main) {
            Log.d(TAG,"1 - ${Thread.currentThread().name}")
        }
    }


}
```

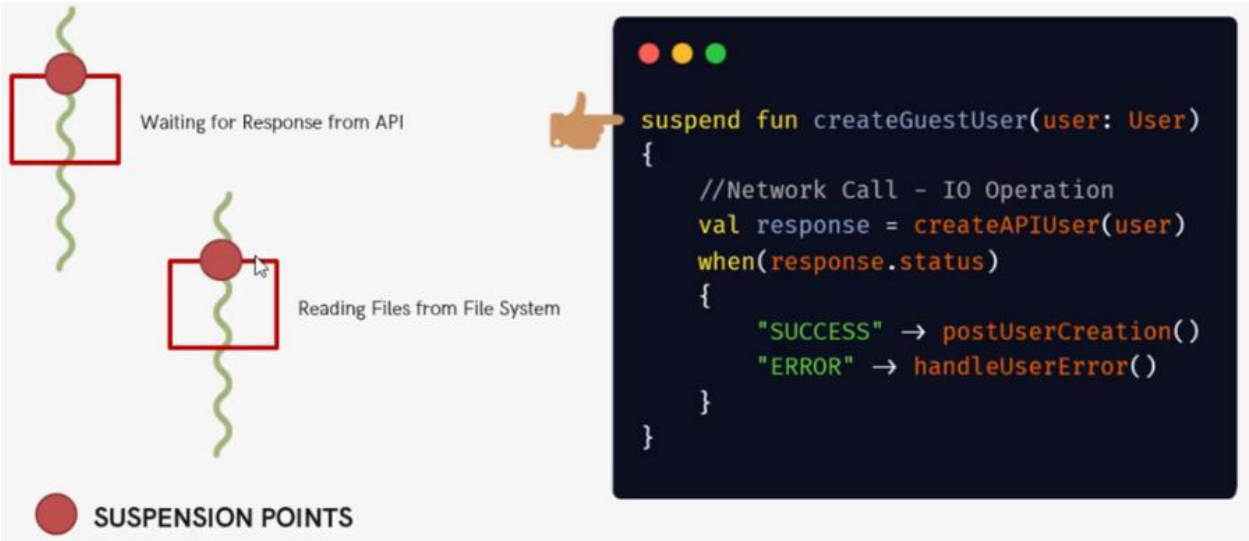
Threads & Coroutine:

```
fun executeTask(view: View)
{
    thread(start = true) {
        executeLongRunningTask()
    }
}
```

```
fun executeTask(view: View)
{
    CoroutineScope(Dispatchers.IO).launch {
        executeLongRunningTask()
    }
}
```

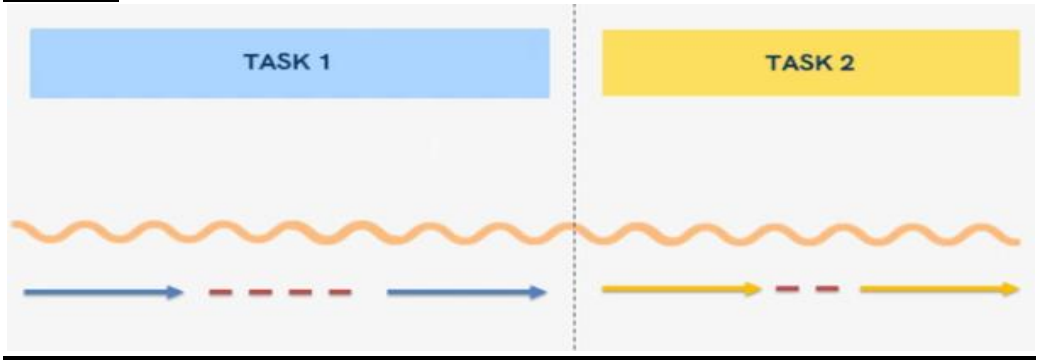
- Coroutines helps to implement functionality that can be suspended & later resumed at specified points without blocking the thread

Suspending function:

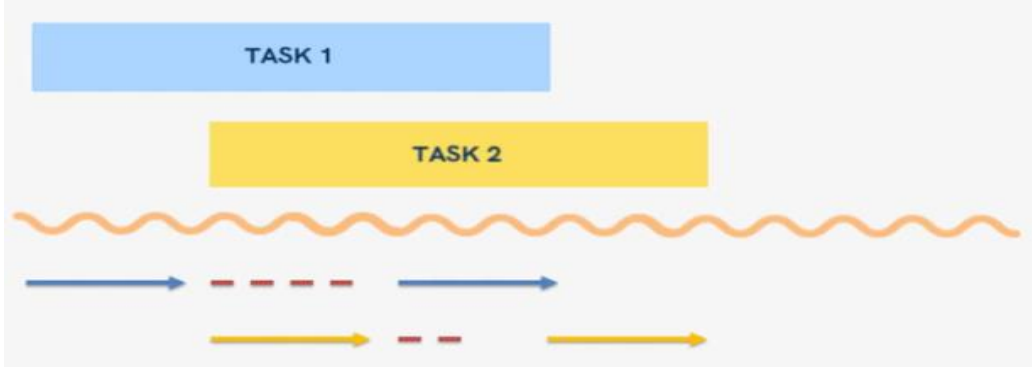


- Functions with suspend modifier.
- Helps coroutine to suspend the computation at a particular point.
- Suspending functions must be called from either **Coroutines** or **Other Suspending Function**

Threads:



Coroutine:



MainActivity.kt

```
package com.ghani.kotlincoroutine2
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.*

class MainActivity : AppCompatActivity() {
   private val TAG:String = "KotlinCoroutine"

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

      CoroutineScope(Dispatchers.Main).launch {
         task1()
      }
      CoroutineScope(Dispatchers.Main).launch {
         task2()
      }
   }
```

```
    suspend fun task1(){
        Log.d(TAG,"Task1 Starting")
        yield()
        //delay(10000)
        Log.d(TAG,"Task1 Ending")
    }
    suspend fun task2(){
        Log.d(TAG,"Task2 Starting")
        yield()
        //delay(10000)
        Log.d(TAG,"Task2 Ending")
    }
}
```

Coroutine Builders:

- Coroutine Builders - Functions that help in creating coroutines.
- We have already seen **launch** function.

- Use Launch - when you do not care about the result. (Fire & Forget)
- Use Async - when you expect result/output from your coroutine
- Although both can be used to achieve the same functionality but it is better to use things that are meant for it.

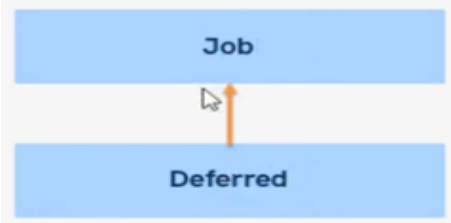
MainActivity.kt:

```
package com.ghani.kotlincoroutine3

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.*

class MainActivity : AppCompatActivity() {
   private val TAG = "Kotlin Coroutine"

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

      CoroutineScope(Dispatchers.IO).launch {
         printFollowers()
      }
   }

   private suspend fun getFbFollowers():Int {
      delay(1000)
      return 54
   }

   private suspend fun getInstaFollowers():Int {
      delay(1000)
      return 113
   }

   /*
   //flow-1
   private suspend fun printFollowers() {
      var fbFollowers = 0
      CoroutineScope(Dispatchers.IO).launch {
         fbFollowers =  getFbFollowers()
      }
      Log.d(TAG,fbFollowers.toString())
   }
   */
```

```
/*
//flow-2
private suspend fun printFollowers() {
   var fbFollowers = 0
   val job = CoroutineScope(Dispatchers.IO).launch {
     fbFollowers =  getFbFollowers()
   }
   job.join()
   Log.d(TAG,fbFollowers.toString())
}
*/

/*
//flow-3
private suspend fun printFollowers() {
   val job = CoroutineScope(Dispatchers.IO).async {
      getFbFollowers()
   }
   Log.d(TAG,job.await().toString())
}
*/

/*
//flow-4
private suspend fun printFollowers() {
   var fbFollowers = 0
   var instaFollowers = 0
   val job = CoroutineScope(Dispatchers.IO).launch {
     fbFollowers =  getFbFollowers()
   }
   val job2 = CoroutineScope(Dispatchers.IO).launch {
      instaFollowers =  getInstaFollowers()
   }
   job.join()
   job2.join()
   Log.d(TAG,"FB-$fbFollowers, Insta-$instaFollowers")
}
*/
```

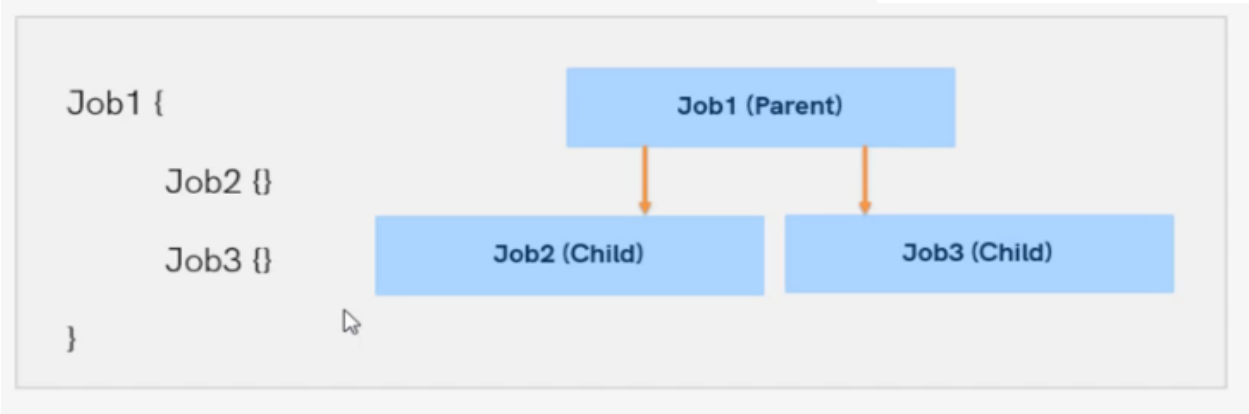
```
    /*
    //flow-5
    private suspend fun printFollowers() {
        val fb= CoroutineScope(Dispatchers.IO).async {
            getFbFollowers()
        }
        val insta = CoroutineScope(Dispatchers.IO).async {
            getInstaFollowers()
        }
        Log.d(TAG,"FB-${fb.await()}, Insta-${insta.await()}")
    }
    */

    /*
    //flow-6
    private suspend fun printFollowers() {
        CoroutineScope(Dispatchers.IO).launch {
            var fb = getFbFollowers()
            var insta =  getInstaFollowers()
            Log.d(TAG,"FB-$fb, Insta-$insta")
        }
    }
    */

    //flow-7
    private suspend fun printFollowers() {
        CoroutineScope(Dispatchers.IO).launch {
            var fb = async {getFbFollowers()}
            var insta = async {getInstaFollowers()}
            Log.d(TAG,"FB-${fb.await()}, Insta-${insta.await()}")
        }
    }

}
```

```
var job = CoroutineScope(Dispatchers.IO).launch {

}

var deferred = CoroutineScope(Dispatchers.IO).async {

}
```




```
var job = launch(Dispatchers.IO)
{
    var fb = async { getFBFollowers() }
    var insta = async { getInstaFollowers() }

    Log.d(TAG, "${ fb.await() } + ${ insta.await() }")
}
```

Job Hierarcy:



MainActivity.kt:

```
package com.ghani.kotlincoroutine4
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.*

class MainActivity : AppCompatActivity() {
   private val TAG = "Kotlin Coroutine"
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      GlobalScope.launch(Dispatchers.Main) {
         execute()
      }
   }
   /*
   //Flow-1(child job catch properties of parent job automatically)
   private suspend fun execute() {
      val parent_job = GlobalScope.launch(Dispatchers.Main) {
         Log.d(TAG, "Parent- $coroutineContext")
         val child_job1 = launch {
            Log.d(TAG, "Child1- $coroutineContext")
         }
         val child_job2 = launch(Dispatchers.IO) {
            Log.d(TAG, "Child2- $coroutineContext")
         }
      }
   }
   */

   /*
   //Flow-2(Parent job cancelled will caused child job cancelled automatically)
   private suspend fun execute() {
      val parent_job = GlobalScope.launch(Dispatchers.Main) {
         Log.d(TAG, "Parent job started")
         val child_job = launch {
            try {
               Log.d(TAG, "Child job started")
               delay(5000)
               Log.d(TAG, "Child job ended")
            }catch (e:CancellationException){
               Log.d(TAG, "Child job cancelled")
            }
```

```
            }
            delay(3000)
            child_job.cancel()
            Log.d(TAG, "Parent job ended")
        }
        //delay(1000)
        //parent_job.cancel()
        parent_job.join()
        Log.d(TAG, "Parent job completed")
    }
    */

    //Flow-3
    private suspend fun execute() {
        val parent_job = CoroutineScope(Dispatchers.IO) .launch{
            for (i in 1..1000){
                /*
                //Flow-3.1(Coroutine is cancelled but thread still busy on executeLongRunningTask() )
                executeLongRunningTask()
                Log.d(TAG, i.toString())
                */

                //Flow-3.2(Coroutine and thread both are cancelled)
                if (isActive){
                    executeLongRunningTask()
                    Log.d(TAG, i.toString())
                }
            }
        }
        delay(100)
        Log.d(TAG, "Cancelling job")
        parent_job.cancel()
        parent_job.join()
        Log.d(TAG, "Parent job completed")
    }
    private  fun executeLongRunningTask() {
        for (i in 1..10000000){
        }
    }

}
```

MainActivity.kt:

```
package com.ghani.kotlincoroutine5

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.*

class MainActivity : AppCompatActivity() {
   private val TAG = "Kotlin Coroutine"

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

      GlobalScope.launch(Dispatchers.Main) {
         execute()
      }
   }


   /*
   //Flow-1.1(Executing order -> Before-After-Inside)
   private suspend fun execute() {
      Log.d(TAG, "Before")
      GlobalScope.launch { //Non-blocking
         delay(1000)
         Log.d(TAG, "Inside")
      }
      Log.d(TAG, "After")
   }
   */

   /*
   //Flow-1.2(Executing order -> Before-Inside-After)
   private suspend fun execute() {
      Log.d(TAG, "Before")
      withContext(Dispatchers.IO){  //Blocking
         delay(1000)
         Log.d(TAG, "Inside")
      }
      Log.d(TAG, "After")
   }
   */
```

```
    /*
    //Flow-2.1 (Only print -> Hello,because thread finish before coroutine)
    private suspend fun execute() {
        GlobalScope.launch {
            delay(100000)
            Log.d(TAG, "World")
        }
        Log.d(TAG, "Hello")
    }
     */
    /*
    //Flow-2.2 (Print -> Hello world,because of sleep, coroutine complete before thread finish)
    private suspend fun execute() {
        GlobalScope.launch {
            delay(100000)
            Log.d(TAG, "World")
        }
        Log.d(TAG, "Hello")
        Thread.sleep(150000)
    }
   */
     //Flow-2.3 (Print -> Hello world,because of runBlocking, coroutine complete before thread
finish)
    private suspend fun execute() {
        runBlocking{
            launch {
                delay(100000)
                Log.d(TAG, "World")
            }
            Log.d(TAG, "Hello")
        }
    }
    /*
    //Flow-2.4 (Print -> Hello world,because of runBlocking, coroutine complete before thread finish)
    //It can also write in intellij IDEA as below (After setup gradle & dependencies):
    fun main() =runBlocking{
        launch {
            delay(100000)
            println("World")
        }
        println("Hello")
    }
    */
}
```

View Model Scope:

- Coroutine scope attached with your View Models.
- Coroutines in this scope will be cancelled automatically when viewmodel is cleared. We don't need to manually cancel the coroutines.

LifeCycleScope:

- Coroutine scope attached with lifecycle (Activity or Fragments)
- Coroutines in this scope will be cancelled automatically when lifecycle is destroyed. We don't need to manually cancel the coroutines.

---

build.gradle(Module):

```
dependencies {
   //Adding by me
   implementation"org.jetbrains.kotlinx:kotlinx-coroutines-core:1.5.1"
   implementation"org.jetbrains.kotlinx:kotlinx-coroutines-android:1.5.1"
   def lifecycle_version = "2.5.1"
   implementation"androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"
   implementation"androidx.lifecycle:lifecycle-runtime-ktx:$lifecycle_version"
}
```

AnotherActivy.kt:

```
package com.ghani.kotlincoroutine6
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class AnotherActivy : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_another_activy)
   }
}
```

MainViewModel.kt:

```
package com.ghani.kotlincoroutine6
import android.util.Log
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
```

```
class MainViewModel: ViewModel(){
   private val TAG:String = "KotlinCoroutine"

   init {
      viewModelScope.launch{
         while (true){
            delay(500)
            Log.d(TAG,"Hello from cheeze code!!")
         }
      }
   }
   override fun onCleared() {   //it works after view model destroyed
      super.onCleared()
      Log.d(TAG,"ViewModel destroyed!!")
   }
}
```

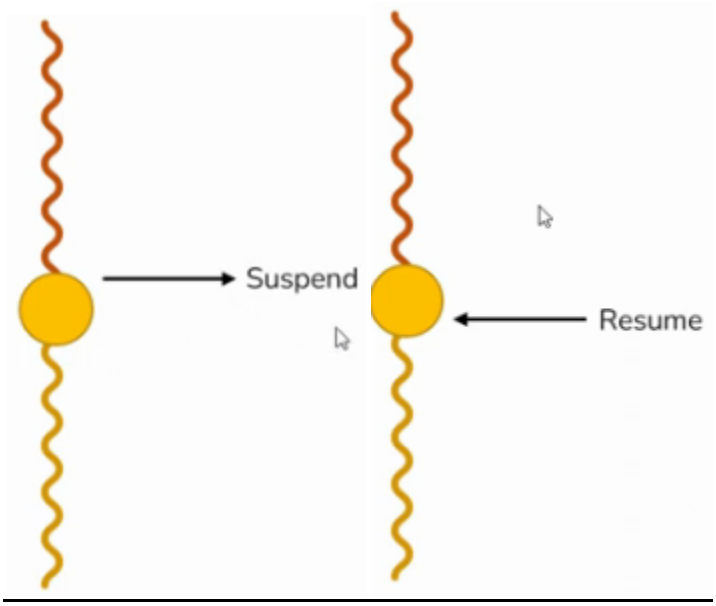
MainActivity.kt:

```
package com.ghani.kotlincoroutine6
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.lifecycleScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch

class MainActivity : AppCompatActivity() {
   lateinit var viewModel: MainViewModel

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      viewModel = ViewModelProvider(this).get(MainViewModel::class.java)

      lifecycleScope.launch{
         delay(2000)
         val intent = Intent(this@MainActivity,AnotherActivy::class.java)
         startActivity(intent)
         finish()
      }
   }
}
```

Kotlin Flows

Coroutines & suspend:

- Coroutines helps to implement asynchronous, non blocking code.
- For this we use - Suspend Functions.



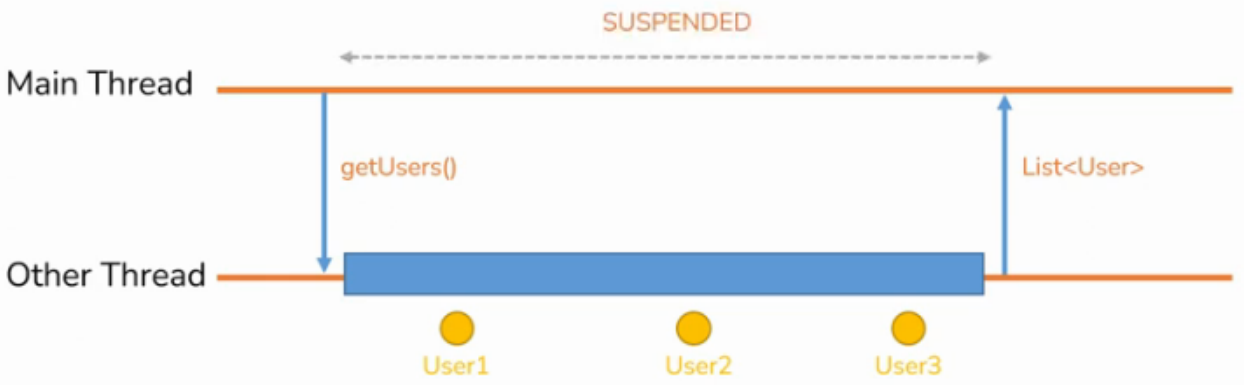
- Either you fire and forget using launch or wait for data (i.e. single object) using async.
- Suspend functions only return a single object.
- Suspend functions work great for things like –
  - Storing some value in database
  - Network calls
  - Doing task that returns single value
- But there are scenarios where you have streams of data –
  - Video Streaming
  - FM radio
  - Mobile sending audio signals to Bluetooth speakers
- For these scenarios, we need streams.

Channels & Flows:

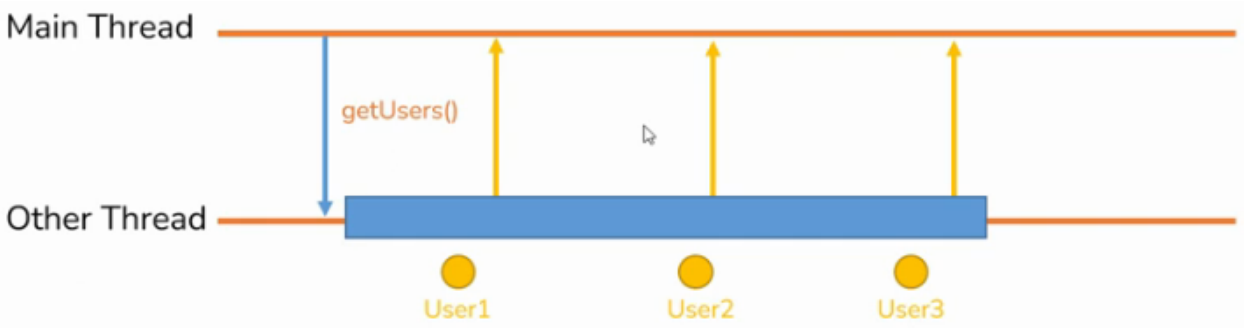
- Kotlin has asynchronous stream support using Channels & Flows.
- Channels (Send & Receive)
- Flows (Emit & Collect)

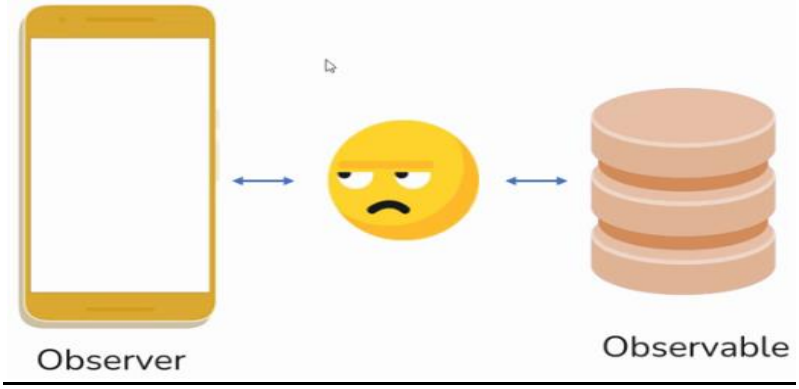
- Channels are Hot.
- Flows are mostly Cold.

Coroutines:



Streams:

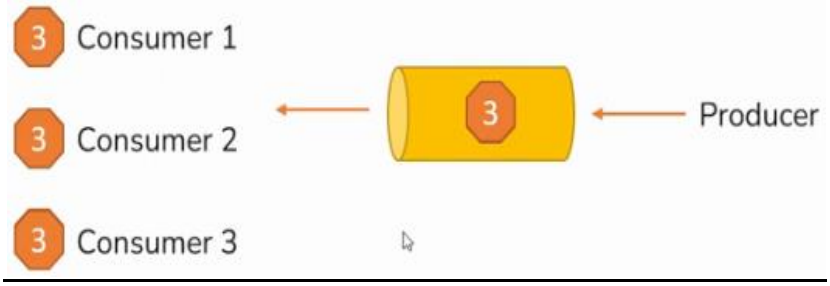




Cold & Hot stream:

- Cold Streams are preferred over Hot Streams.
- Resource Wastage
- Manual Close

StateFlow:



LiveData vs StateFlow:

- Transformations on Main Thread
- Operators
- LifeCycle Dependent

Program1: Coroutine shows all the items in same time after complete whole execution.

MainActivity.kt:

```
package com.ghani.kotlinflows

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch

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

        CoroutineScope(Dispatchers.Main).launch {
            getUserNames().forEach{
                Log.d("KotinFlows",it)
            }
        }
    }
    private suspend fun getUserNames(): List<String> {
        val l = mutableListOf<String>()
        l.add(getUser("1"))
        l.add(getUser("2"))
        l.add(getUser("3"))
        return l
    }
    private suspend fun getUser(id: String): String {
        delay(3000)  //Assume network call
        return "User$id"
    }
}
```

Output:

```
KotlinCoroutine         com.ghani.kotlinflows               D  User1
KotlinCoroutine         com.ghani.kotlinflows               D  User2
KotlinCoroutine         com.ghani.kotlinflows               D  User3
```

Program2: Channel shows items step by step when execution running.

MainActivity.kt:

```
package com.ghani.kotlinflows2

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.channels.Channel
import kotlinx.coroutines.launch

class MainActivity : AppCompatActivity() {
    val channel = Channel<Int>()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        producer()
        consumer()
    }
    fun producer(){
        CoroutineScope(Dispatchers.Main).launch {
            channel.send(1)
            channel.send(2)
            channel.send(3)
        }
    }
    fun consumer(){
        CoroutineScope(Dispatchers.Main).launch {
            Log.d("KotlinChannels-1",channel.receive().toString())
            Log.d("KotlinChannels-2",channel.receive().toString())
            Log.d("KotlinChannels-3",channel.receive().toString())
        }
    }
}
```

Output:

```
KotlinChannels-1        com.ghani.kotlinflows2          D  1
KotlinChannels-2        com.ghani.kotlinflows2          D  2
KotlinChannels-3        com.ghani.kotlinflows2          D  3
```

Program3:

Part-1: val job has properties cancel() that cancelling the flows.

Part-2: Both flows start from beginning, delay does not matter.

Part-3: Processing emit by various properties.

Part-4: Processing emit by (first(),toList())

Part-5: Processing emit by various Non-terminal properties.

MainActivity.kt:

```
package com.ghani.kotlinflows3

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.*

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

      /*
      //Part-1
      val job = GlobalScope.launch {
         val data: Flow<Int> = producer()
         data.collect{
            Log.d("KotlinFlows",it.toString())
         }
      }
      GlobalScope.launch {
         delay(9000)
         job.cancel()
      }
      */
```

```
/*
//Part-2
GlobalScope.launch {
  val data: Flow<Int> = producer()
  data.collect{
     Log.d("KotlinFlows-1",it.toString())
  }
}
GlobalScope.launch {
   val data: Flow<Int> = producer()
   delay(4000)
   data.collect{
      Log.d("KotlinFlows-2",it.toString())
   }
}
*/

/*
//Part-3
GlobalScope.launch(Dispatchers.Main) {
   producer()
      .onStart {
         emit(-10)
         Log.d("KotlinFlows", "Starting out")
      }
      .onCompletion {
         emit(60)
         Log.d("KotlinFlows", "Completed")
      }
      .onEach {
         //May be, this function call first, even its call before (.onStart) function
         Log.d("KotlinFlows", "About to emit $it ")
      }
      .collect {
         Log.d("KotlinFlows", "${it.toString()}")
      }
}*/
```

```
        /*
        //Part-4
        GlobalScope.launch {
            //Terminal operators (first(),toList()) contain suspend keyword
            val result1 = producer().first()
            val result2 = producer().toList()
                Log.d("KotlinFlows",result1.toString())
                Log.d("KotlinFlows",result2.toString())
        }
        */


        //Part-5
        GlobalScope.launch(Dispatchers.Main) {
            //Non-terminal operators (map,filter) & terminal operator(collect)
            //Without terminal operator,flow not start
            producer()
                .map {
                    it * 2
                }
                .filter {
                    it < 8
                }
                .collect {
                    Log.d("KotlinFlows", it.toString())
                }
        }
    }

    fun producer() = flow<Int> {
        val l = listOf<Int>(1, 2, 3, 4, 5)
        l.forEach {
            delay(2000)
            emit(it)
        }
    }
}
```

Output:

Part-1:

```
KotlinFlows          com.ghani.kotlinflows3          D  1
KotlinFlows          com.ghani.kotlinflows3          D  2
KotlinFlows          com.ghani.kotlinflows3          D  3
KotlinFlows          com.ghani.kotlinflows3          D  4
```

Part-2:

```
KotlinFlows-1        com.ghani.kotlinflows3          D  1
KotlinFlows-1        com.ghani.kotlinflows3          D  2
KotlinFlows-2        com.ghani.kotlinflows3          D  1
KotlinFlows-1        com.ghani.kotlinflows3          D  3
KotlinFlows-2        com.ghani.kotlinflows3          D  2
KotlinFlows-1        com.ghani.kotlinflows3          D  4
KotlinFlows-2        com.ghani.kotlinflows3          D  3
KotlinFlows-1        com.ghani.kotlinflows3          D  5
KotlinFlows-2        com.ghani.kotlinflows3          D  4
KotlinFlows-2        com.ghani.kotlinflows3          D  5
```

Part-3:

```
KotlinFlows          com.ghani.kotlinflows3          D  About to emit -10
KotlinFlows          com.ghani.kotlinflows3          D  -10
KotlinFlows          com.ghani.kotlinflows3          D  Starting out
```

```
KotlinFlows          com.ghani.kotlinflows3          D  About to emit 1
KotlinFlows          com.ghani.kotlinflows3          D  1
KotlinFlows          com.ghani.kotlinflows3          D  About to emit 2
KotlinFlows          com.ghani.kotlinflows3          D  2
KotlinFlows          com.ghani.kotlinflows3          D  About to emit 3
KotlinFlows          com.ghani.kotlinflows3          D  3
KotlinFlows          com.ghani.kotlinflows3          D  About to emit 4
KotlinFlows          com.ghani.kotlinflows3          D  4
KotlinFlows          com.ghani.kotlinflows3          D  About to emit 5
KotlinFlows          com.ghani.kotlinflows3          D  5
KotlinFlows          com.ghani.kotlinflows3          D  About to emit 60
KotlinFlows          com.ghani.kotlinflows3          D  60
KotlinFlows          com.ghani.kotlinflows3          D  Completed
```

Part-4:

```
KotlinFlows          com.ghani.kotlinflows3          D  1
KotlinFlows          com.ghani.kotlinflows3          D  [1, 2, 3, 4, 5]
```

Part-5:

```
KotlinFlows          com.ghani.kotlinflows3          D  2
KotlinFlows          com.ghani.kotlinflows3          D  4
KotlinFlows          com.ghani.kotlinflows3          D  6
```

Program4: Use of measureTimeMillis {} & buffer() function.

MainActivity.kt:

```
package com.ghani.kotlinflows4

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.*
import kotlinx.coroutines.launch
import kotlin.system.measureTimeMillis

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

        GlobalScope.launch(Dispatchers.Main) {
            val time = measureTimeMillis {
                producer()
                    .buffer(3) // it takes 3 items for saving the time
                    .collect {
                        delay(1500)
                        Log.d("KotlinFlows", it.toString())
                    }
            }
            Log.d("KotlinFlows",time.toString())
        }
    }

    /*
    private fun producer() = flow<Int> {
        val l = listOf<Int>(1, 2, 3, 4, 5)
        l.forEach {
            delay(1000)
            emit(it)
        }
    }
    */
```

```
  //same to above
  private fun producer(): Flow<Int> {
     return flow<Int> {
        val l = listOf<Int>(1, 2, 3, 4, 5)
        l.forEach {
           delay(1000)
           emit(it)   //it work only within flow
        }
     }
  }
}
```

Output:

```
KotlinFlows          com.ghani.kotlinflows4          D  1
KotlinFlows          com.ghani.kotlinflows4          D  2
KotlinFlows          com.ghani.kotlinflows4          D  3
KotlinFlows          com.ghani.kotlinflows4          D  4
KotlinFlows          com.ghani.kotlinflows4          D  5
KotlinFlows          com.ghani.kotlinflows4          D  9510
```

Program5: Use of .map{} & .filter() function.

MainActivity.kt:

```
package com.ghani.kotlinflows5

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.asFlow
import kotlinx.coroutines.flow.filter
import kotlinx.coroutines.flow.map
import kotlinx.coroutines.launch

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

```
        GlobalScope.launch(Dispatchers.Main){
            getNotes()
                .map{
                    FormattedNote(it.isActive,it.title.uppercase(),it.description)
                }
                .filter{
                    it.isActive
                }
                .collect{
                    Log.d("KotlinFlows",it.toString())
                }
        }
    }
}

private fun getNotes(): Flow<Note>{
        val l = listOf(
            Note(1,true,"First","First description"),
            Note(2,true,"Second","Second description"),
            Note(3,false,"Third","Third description")
        )
    return l.asFlow()
}

data class Note(val id:Int,val isActive:Boolean,val title:String,val description:String)
data class FormattedNote(val isActive:Boolean,val title:String,val description:String)
```

Output:

```
KotlinFlows    com.ghani.kotlinflows5    D  FormattedNote(isActive=true, title=FIRST, description=First description)
KotlinFlows    com.ghani.kotlinflows5    D  FormattedNote(isActive=true, title=SECOND, description=Second description)
```

Program6: Use of .flowOn() function that work for all the functions that are above from this function.

MainActivity.kt:

```
package com.ghani.kotlinflows6

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
```

```
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.*
import kotlinx.coroutines.launch

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

      GlobalScope.launch(Dispatchers.Main) {
            producer()
               .map{
                  delay(1000)
                  it * 2
                  Log.d("KotlinFlows-2", "Map thread - ${Thread.currentThread().name}")
               }
               .filter{
                  delay(1000)
                  Log.d("KotlinFlows-3", "Filter thread - ${Thread.currentThread().name}")
                  it < 8
               }
               .flowOn(Dispatchers.IO)
               .collect {
                  Log.d("KotlinFlows-4", "Collector thread - ${Thread.currentThread().name}")
               }
      }
   }

   private fun producer(): Flow<Int> {
      return flow<Int> {
         val l = listOf<Int>(1, 2, 3,4,5)
         l.forEach {
            delay(1000)
            Log.d("KotlinFlows-1", "Emitter thread - ${Thread.currentThread().name}")
            emit(it)
         }
      }
   }
}
```

Output:

```
KotlinFlows-1    com.ghani.kotlinflows6    D  Emitter thread - DefaultDispatcher-worker-1
KotlinFlows-2    com.ghani.kotlinflows6    D  Map thread - DefaultDispatcher-worker-1
KotlinFlows-3    com.ghani.kotlinflows6    D  Filter thread - DefaultDispatcher-worker-1
KotlinFlows-4    com.ghani.kotlinflows6    D  Collector thread - main
KotlinFlows-1    com.ghani.kotlinflows6    D  Emitter thread - DefaultDispatcher-worker-1
KotlinFlows-2    com.ghani.kotlinflows6    D  Map thread - DefaultDispatcher-worker-1
KotlinFlows-3    com.ghani.kotlinflows6    D  Filter thread - DefaultDispatcher-worker-3
KotlinFlows-4    com.ghani.kotlinflows6    D  Collector thread - main
KotlinFlows-1    com.ghani.kotlinflows6    D  Emitter thread - DefaultDispatcher-worker-3
KotlinFlows-2    com.ghani.kotlinflows6    D  Map thread - DefaultDispatcher-worker-3
KotlinFlows-3    com.ghani.kotlinflows6    D  Filter thread - DefaultDispatcher-worker-3
KotlinFlows-4    com.ghani.kotlinflows6    D  Collector thread - main
KotlinFlows-1    com.ghani.kotlinflows6    D  Emitter thread - DefaultDispatcher-worker-3
KotlinFlows-2    com.ghani.kotlinflows6    D  Map thread - DefaultDispatcher-worker-3
KotlinFlows-3    com.ghani.kotlinflows6    D  Filter thread - DefaultDispatcher-worker-3
KotlinFlows-4    com.ghani.kotlinflows6    D  Collector thread - main
KotlinFlows-1    com.ghani.kotlinflows6    D  Emitter thread - DefaultDispatcher-worker-3
KotlinFlows-2    com.ghani.kotlinflows6    D  Map thread - DefaultDispatcher-worker-1
KotlinFlows-3    com.ghani.kotlinflows6    D  Filter thread - DefaultDispatcher-worker-3
KotlinFlows-4    com.ghani.kotlinflows6    D  Collector thread - main
```

Program7: Error handling.

MainActivity.kt:

```
package com.ghani.kotlinflows7

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.*
import kotlinx.coroutines.launch

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

```
    /*
    //Part-1(Error in Emitter that handling in Collector)
    GlobalScope.launch(Dispatchers.Main) {
        try{
            producer()
                .collect {
                    Log.d("KotlinFlows-2", "Collector thread - ${Thread.currentThread().name}")
                }
        }
        catch (e:Exception){
            Log.d("KotlinFlows-3", e.message.toString())
        }
    }
}

private fun producer(): Flow<Int> {
    return flow<Int> {
        val l = listOf<Int>(1, 2, 3, 4, 5)
        l.forEach {
            delay(1000)
            Log.d("KotlinFlows-1", "Emitter thread - ${Thread.currentThread().name}")
            emit(it)  //item(it) emit from here to .collect
            throw Exception("Error in Emitter")
        }
    }
}
*/

    /*
    //Part-2(Error in Collector that handling in Collector)
    GlobalScope.launch(Dispatchers.Main) {
        try{
            producer()
                .collect {
                    Log.d("KotlinFlows-2", "Collector thread - ${Thread.currentThread().name}")
                    throw Exception("Error in Collector")
                }
        }
        catch (e:Exception){
            Log.d("KotlinFlows-3", e.message.toString())
        }
    }
}
```

```
    private fun producer(): Flow<Int> {
        return flow<Int> {
            val l = listOf<Int>(1, 2, 3, 4, 5)
            l.forEach {
                delay(1000)
                Log.d("KotlinFlows-1", "Emitter thread - ${Thread.currentThread().name}")
                emit(it)
            }
        }
    }
    */

        //Part-3(Error in Emitter that handling in Emitter)
        GlobalScope.launch(Dispatchers.Main) {
            try{
                producer()
                    .collect {
                        Log.d("KotlinFlows-3", "Collector thread - ${Thread.currentThread().name}")
                    }
            }
            catch (e:Exception){
                Log.d("KotlinFlows-4", e.message.toString())
            }
        }
    }

    private fun producer(): Flow<Int> {
        return flow<Int> {
            val l = listOf<Int>(1, 2, 3, 4, 5)
            l.forEach {
                delay(1000)
                Log.d("KotlinFlows-1", "Emitter thread - ${Thread.currentThread().name}")
                emit(it)
                throw Exception("Error in Emitter")
            }
        }.catch{
            Log.d("KotlinFlows-2", "Emitter catch - ${it.message}")
            emit(-1)
        }
    }
}
```

Output:

Part-1:

```
KotlinFlows-1          com.ghani.kotlinflows7          D  Emitter thread - main
KotlinFlows-2          com.ghani.kotlinflows7          D  Collector thread - main
KotlinFlows-3          com.ghani.kotlinflows7          D  Error in Emitter
```

Part-2:

```
KotlinFlows-1          com.ghani.kotlinflows7          D  Emitter thread - main
KotlinFlows-2          com.ghani.kotlinflows7          D  Collector thread - main
KotlinFlows-3          com.ghani.kotlinflows7          D  Error in Collector
```

Part-3:

```
KotlinFlows-1          com.ghani.kotlinflows7          D  Emitter thread - main
KotlinFlows-3          com.ghani.kotlinflows7          D  Collector thread - main
KotlinFlows-2          com.ghani.kotlinflows7          D  Emitter catch - Error in Emitter
KotlinFlows-3          com.ghani.kotlinflows7          D  Collector thread - main
```

Program8:

Part-1:(Collector from this Flow, collect items from the beginning).

Part-2:(Collector from this SharedFlow, collect items from recent flow).

MainActivity.kt:

```
package com.ghani.kotlinflows8

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.flow
import kotlinx.coroutines.launch

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

```
        GlobalScope.launch(Dispatchers.Main) {
            val result = producer()
            result.collect{
                Log.d("KotlinFlows-1",it.toString())
            }
        }
        GlobalScope.launch(Dispatchers.Main)  {
            val result = producer()
            delay(6000)
            result.collect{
                Log.d("KotlinFlows-2",it.toString())
            }
        }
    }

    /*
    //Part-1(Collector from this Flow, collect items from the beginning)
    private fun producer(): Flow<Int> {
        return flow<Int> {
            val l = listOf<Int>(1, 2, 3, 4, 5)
            l.forEach {
                delay(2000)
                emit(it)
            }
        }
    }
    */

    //Part-2(Collector from this SharedFlow, collect items from recent flow)
    private fun producer(): Flow<Int> {
        val mutableSharedFlow = MutableSharedFlow<Int>(1)
        //(..) -> stored recently last item/items
        GlobalScope.launch {
            val l = listOf<Int>(1, 2, 3, 4, 5)
            l.forEach {
                mutableSharedFlow.emit(it)
                delay(2000)
            }
        }
        return mutableSharedFlow
    }
}
```

Output:

Part-1:

```
KotlinFlows-1          com.ghani.kotlinflows8          D  1
KotlinFlows-1          com.ghani.kotlinflows8          D  2
KotlinFlows-1          com.ghani.kotlinflows8          D  3
KotlinFlows-2          com.ghani.kotlinflows8          D  1
KotlinFlows-1          com.ghani.kotlinflows8          D  4
KotlinFlows-2          com.ghani.kotlinflows8          D  2
KotlinFlows-1          com.ghani.kotlinflows8          D  5
KotlinFlows-2          com.ghani.kotlinflows8          D  3
KotlinFlows-2          com.ghani.kotlinflows8          D  4
KotlinFlows-2          com.ghani.kotlinflows8          D  5
```

Part-2:

```
KotlinFlows-1          com.ghani.kotlinflows8          D  1
KotlinFlows-1          com.ghani.kotlinflows8          D  2
KotlinFlows-1          com.ghani.kotlinflows8          D  3
KotlinFlows-2          com.ghani.kotlinflows8          D  4
KotlinFlows-1          com.ghani.kotlinflows8          D  4
KotlinFlows-2          com.ghani.kotlinflows8          D  5
KotlinFlows-1          com.ghani.kotlinflows8          D  5
```

Program9: MutableSharedFlow vs MutableStateFlow

MainActivity.kt:

```
package com.ghani.kotlinflows9

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.launch
```

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

        GlobalScope.launch(Dispatchers.Main)  {
            val result = producer()
            delay(6000)
            result.collect{
                Log.d("KotlinFlows-2",it.toString())
            }
        }
    }

    /*
    //Part-1
    private fun producer(): Flow<Int> {
        val mutableSharedFlow = MutableSharedFlow<Int>()
        GlobalScope.launch {
            val l = listOf<Int>(1, 2, 3, 4, 5)
            l.forEach {
                mutableSharedFlow.emit(it)
                Log.d("KotlinFlows-1",it.toString())
                delay(1000)
            }
        }
        return mutableSharedFlow
    }
    */

    //Part-2(MutableStateFlow stored last item)
    private fun producer(): Flow<Int> {
        val mutableStateFlow = MutableStateFlow<Int>(10)  //initial stored value(10) giving by me
        GlobalScope.launch {
            val l = listOf<Int>(1, 2, 3, 4, 5)
            l.forEach {
                mutableStateFlow.emit(it)
                Log.d("KotlinFlows-1",it.toString())
                delay(1000)
            }
        }
        return mutableStateFlow
    }
}
```

Output:

Part-1:

```
KotlinFlows-1          com.ghani.kotlinflows9          D  1
KotlinFlows-1          com.ghani.kotlinflows9          D  2
KotlinFlows-1          com.ghani.kotlinflows9          D  3
KotlinFlows-1          com.ghani.kotlinflows9          D  4
KotlinFlows-1          com.ghani.kotlinflows9          D  5
```

Part-2:

```
KotlinFlows-1          com.ghani.kotlinflows9          D  1
-------------------------------------------------------------
KotlinFlows-1          com.ghani.kotlinflows9          D  2
KotlinFlows-1          com.ghani.kotlinflows9          D  3
KotlinFlows-1          com.ghani.kotlinflows9          D  4
KotlinFlows-1          com.ghani.kotlinflows9          D  5
KotlinFlows-2          com.ghani.kotlinflows9          D  5
```

Program10: return `StateFlow and use it properties (.value).`

MainActivity.kt:

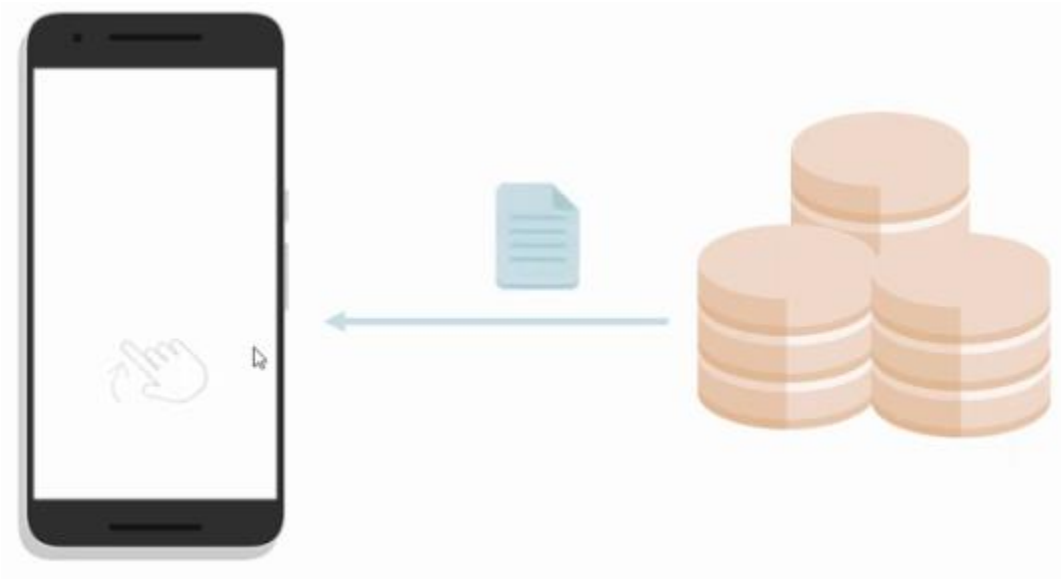
```
package com.ghani.kotlinflows10

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch

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

```
        GlobalScope.launch(Dispatchers.Main)  {
            val result = producer()
            delay(6000)
            Log.d("KotlinFlows-2",result.value.toString())
        }
    }

    private fun producer(): StateFlow<Int> {
        val mutableStateFlow = MutableStateFlow<Int>(10)   //Stored last item
        GlobalScope.launch {
            val l = listOf<Int>(1, 2, 3, 4, 5)
            l.forEach {
                mutableStateFlow.emit(it)
                Log.d("KotlinFlows-1",it.toString())
                delay(1000)
            }
        }
        return mutableStateFlow
    }
}
```

Output:

KotlinFlows-1	com.ghani.kotlinflows9	D	1
KotlinFlows-1	com.ghani.kotlinflows9	D	2
KotlinFlows-1	com.ghani.kotlinflows9	D	3
KotlinFlows-1	com.ghani.kotlinflows9	D	4
KotlinFlows-1	com.ghani.kotlinflows9	D	5
KotlinFlows-2	com.ghani.kotlinflows9	D	5

Paging 3



Paging 3 :

-Paging Source

-Pager

-Paging Adapter

val anchorPosition: Int?

//Most recently accessed index in the list.

fun closestPageToPosition(anchorPosition: Int)

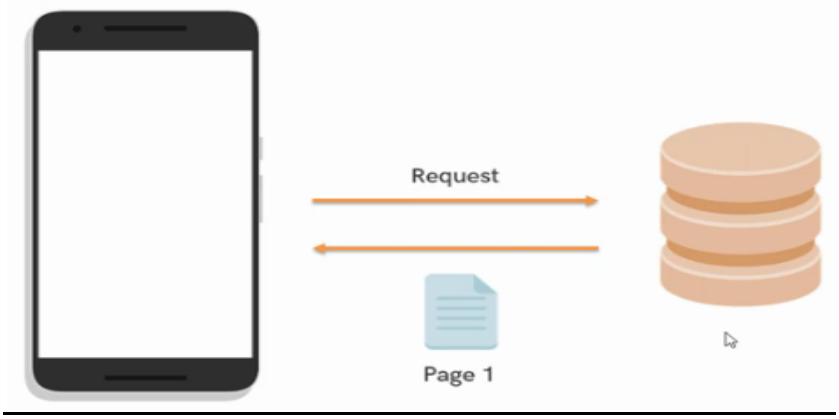
/*

This function can be called with anchorPosition to fetch the loaded page that is closest to the last accessed index in the list.

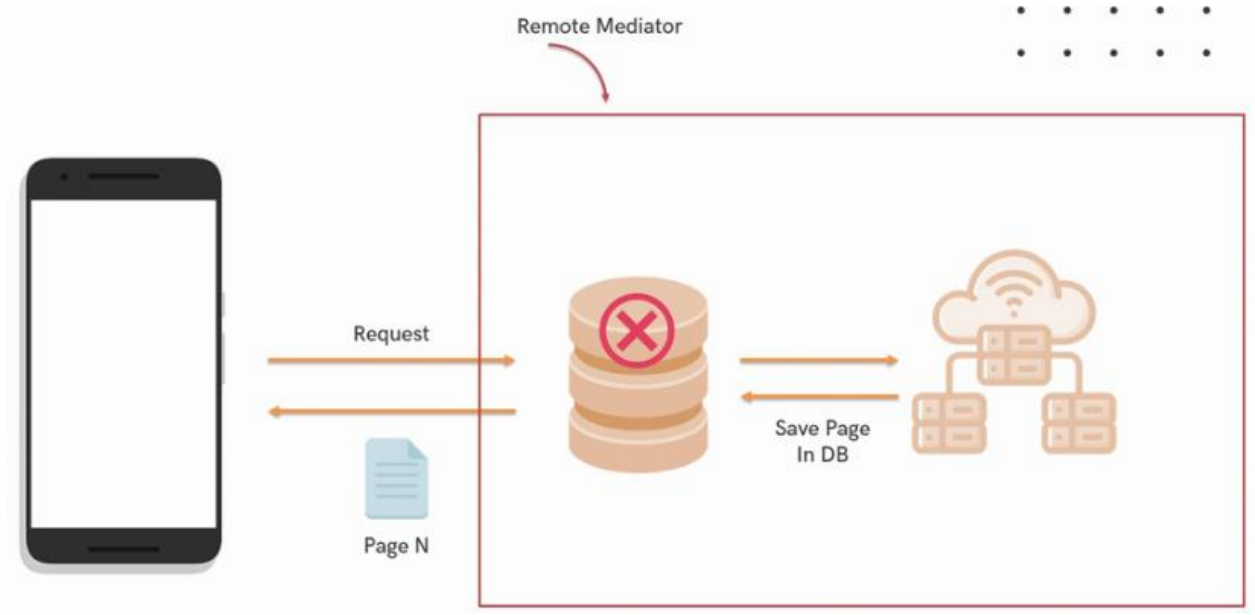
*/

Paging with API:

Paging with DB:



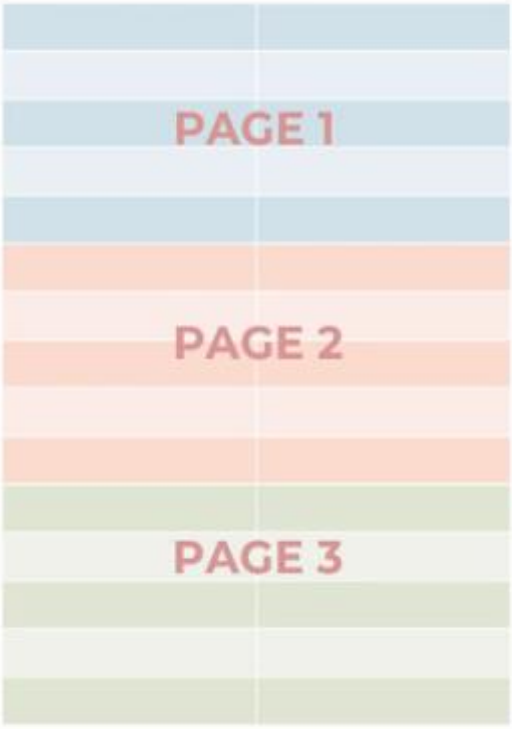
Remote Mediator:



PAGING 3

- Write logic to fetch the page
- Save the Quotes in DB (Create room implementation)
- Another table is needed to maintain the keys
- This table is used to calculate the next key or prev key.

Quote Table



RemoteKeys

Id	Prev Key	Next Key
ase12s	null	2
qwer11	null	2
xqd413	null	2
trt123a	null	2
tr1114	1	3
ssre22	1	3
rwee12	1	3
w4343a	1	3
a12343	1	3

PAGING 3

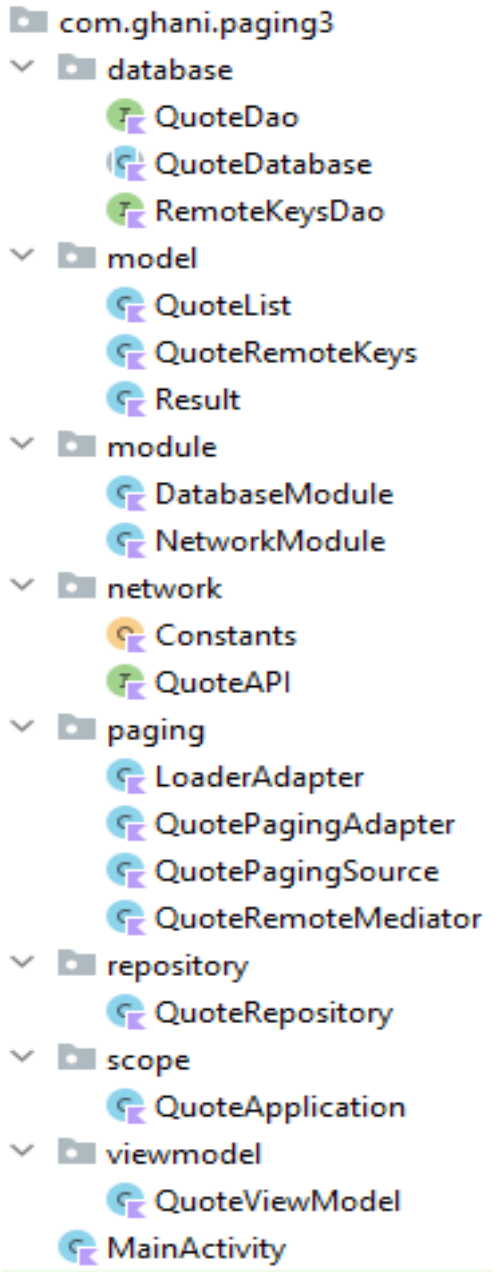
- First Time Fetch or Refresh
- Prepend
- Append

Flow-1:

QuoteList-Result-Constant-QuoteAPI-NetworkModule-QuotePagingService-QuoteRepository-QuoteViewModel-QuotePagingAdapter-LoaderAdapter-QuoteApplication-MainActivity.

Flow-2:

QuoteList - Result - Constant - QuoteAPI - NetworkModule -QuoteRemoteKeys-QuoteDatabase-RemotKeysDao-QuoteDATABASE-DatabaseModule-QouteRepositoryModule-QuoteReposity-QouteViewModel-QoutePagingAdapter-LoaderAdapter- QuoteApplication-MainActivity.



build.gradle(Project):

```
plugins {
   //Adding by me
   id("com.google.dagger.hilt.android") version "2.44" apply false
}
```

build.gradle(Module):

```
plugins {
   //Adding by me
   id 'kotlin-kapt'
   id'com.google.dagger.hilt.android'
}

dependencies {
   //Adding by me
   def hilt_version = "2.44"
   implementation"com.google.dagger:hilt-android:$hilt_version"
   kapt"com.google.dagger:hilt-android-compiler:$hilt_version"

   def lifecycle_version = "2.5.1"
   implementation"androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"
   implementation"androidx.lifecycle:lifecycle-livedata-ktx:$lifecycle_version"

   def retrofit_version = "2.9.0"
   implementation "com.squareup.retrofit2:retrofit:$retrofit_version"
   implementation "com.squareup.retrofit2:converter-gson:$retrofit_version"

   def room_version = "2.5.1"
   implementation"androidx.room:room-runtime:$room_version"
   implementation"androidx.room:room-ktx:$room_version"
   implementation"androidx.room:room-paging:$room_version"
   kapt"androidx.room:room-compiler:$room_version"

   def coroutines_version = "1.5.1"
   implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:$coroutines_version")
   implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:$coroutines_version")

   def paging_version = "3.1.1"
   implementation("androidx.paging:paging-runtime:$paging_version")
}
```

AndroidManifest.xml:

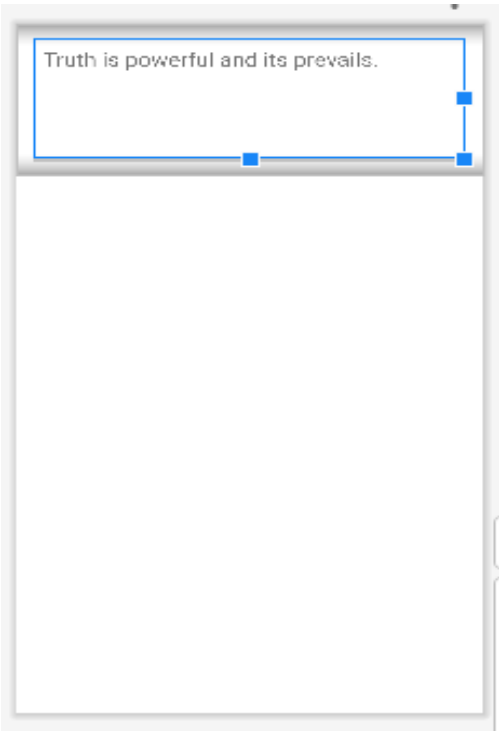
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools">

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

   <application
      android:usesCleartextTraffic="true"
      android:name=".scope.QuoteApplication"
   </application>

</manifest>
```

loader_item.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <ProgressBar
      android:id="@+id/progress_bar"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

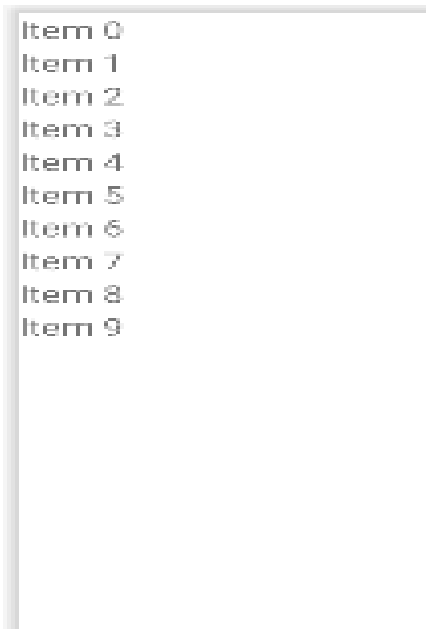
quote_item.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:padding="16dp"
   android:layout_width="match_parent"
   android:layout_height="160dp">

   <TextView
      android:id="@+id/quote"
      android:padding="8dp"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="Truth is powerful and its prevails."
      android:textSize="20sp"/>

   <View
      android:layout_width="match_parent"
      android:layout_height="2dp"
      android:background="#CCC"/>

</LinearLayout>
```



activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

   <androidx.recyclerview.widget.RecyclerView
      android:id="@+id/quoteList"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintTop_toTopOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```



QuoteList.kt:

```
package com.ghani.paging3.model

data class QuoteList(
   val count: Int,
   val lastItemIndex: Int,
   val page: Int,
   val results: List<Result>,
   val totalCount: Int,
   val totalPages: Int
)
```

Result.kt:

```
package com.ghani.paging3.model

import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "Quote")
data class Result(
   @PrimaryKey(autoGenerate = false)
   val _id: String,

   val author: String,
   val authorSlug: String,
   val content: String,
   val dateAdded: String,
   val dateModified: String,
   val length: Int,
)
```

Constants.kt:

```
package com.ghani.paging3.network

object Constants {
   const val BASE_url = "http://quotable.io/"
}
```

QuoteAPI.kt:

```
package com.ghani.paging3.network
import com.ghani.paging3.model.QuoteList
import retrofit2.http.GET
import retrofit2.http.Query

interface QuoteAPI {

   @GET("/quotes")
   suspend fun getQuotes(@Query("page") page:Int):QuoteList

}
```

NetworkModule.kt:

```
package com.ghani.paging3.module
import com.ghani.paging3.network.QuoteAPI
import com.ghani.paging3.network.Constants
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import javax.inject.Singleton

@InstallIn(SingletonComponent::class)
@Module
class NetworkModule {

   @Singleton
   @Provides
   fun providesRetrofit():Retrofit{
      return Retrofit.Builder().baseUrl(Constants.BASE_url)
         .addConverterFactory(GsonConverterFactory.create())
         .build()
   }
   @Singleton
   @Provides
   fun getQuoteAPI(retrofit: Retrofit): QuoteAPI {
      return retrofit.create(QuoteAPI::class.java)
   }

}
```

QuoteRemoteKeys.kt:

```
package com.ghani.paging3.model

import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity
data class QuoteRemoteKeys(
    @PrimaryKey
    val id: String,

    val prevPage: Int?,
    val nextPage: Int?
)
```

QuoteDao.kt:

```
package com.ghani.paging3.database

import androidx.paging.PagingSource
import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import com.ghani.paging3.model.Result

@Dao
interface QuoteDao {

    @Query("SELECT * FROM quote")
    fun getQuotes(): PagingSource<Int, Result>    //Return as pages form.

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun addQuotes(quotes: List<Result>)

    @Query("DELETE FROM quote")
    suspend fun deleteQuotes()

}
```

RemoteKeysDao.kt:

```
package com.ghani.paging3.database

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import com.ghani.paging3.model.QuoteRemoteKeys

@Dao
interface RemoteKeysDao {

   @Query("SELECT * FROM QuoteRemoteKeys WHERE id = :id")
   suspend fun getRemoteKeys(id:String): QuoteRemoteKeys

   @Insert(onConflict = OnConflictStrategy.REPLACE)
   suspend fun addAllRemoteKeys(remoteKeys: List<QuoteRemoteKeys>)

   @Query("DELETE FROM QuoteRemoteKeys")
   suspend fun deleteAllRemoteKeys()

}
```

QuoteDatabase.kt:

```
package com.ghani.paging3.database

import androidx.room.Database
import androidx.room.RoomDatabase
import com.ghani.paging3.model.QuoteRemoteKeys
import com.ghani.paging3.model.Result

@Database(entities = [Result::class, QuoteRemoteKeys::class],version = 1)
abstract class QuoteDatabase:RoomDatabase() {

   abstract fun quoteDao():QuoteDao
   abstract fun remoteKeysDao():RemoteKeysDao

}
```

DatabaseModule.kt:

```
package com.ghani.paging3.module
import android.content.Context
import androidx.room.Room
import com.ghani.paging3.database.QuoteDatabase
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

@InstallIn(SingletonComponent::class)
@Module
class DatabaseModule {
   @Provides
   @Singleton
   fun provideDatabase(@ApplicationContext context: Context): QuoteDatabase {
     return Room.databaseBuilder(context, QuoteDatabase::class.java,"quoteDB").build()
   }
}
```

QuotePagingSource.kt:

```
package com.ghani.paging3.paging
import androidx.paging.PagingSource
import androidx.paging.PagingState
import com.ghani.paging3.model.Result
import com.ghani.paging3.network.QuoteAPI

class QuotePagingSource(private val quoteAPI: QuoteAPI): PagingSource<Int, Result>() {
   //This fun load data according to page number:
   override suspend fun load(params: LoadParams<Int>): LoadResult<Int, Result> {
      return try {
         val position = params.key ?: 1  //if params.key = null, then it take 1
         val response = quoteAPI.getQuotes(position)
         LoadResult.Page(
            data = response.results,
            prevKey = if (position == 1) null else position - 1,
            nextKey = if (position == response.totalPages) null else position + 1
         )
      } catch (e: Exception) {
         LoadResult.Error(e)
      }
   }
```

```
    //This fun find the page number:
    override fun getRefreshKey(state: PagingState<Int, Result>): Int? {
        return state.anchorPosition?.let {anchorPosition ->
            val anchorPage = state.closestPageToPosition(anchorPosition)
            anchorPage?.prevKey?.plus(1) ?: anchorPage?.nextKey?.minus(1)
        }

        /*
        //Below code is written to understand the above code:
        if (state.anchorPosition != null) {
            val anchorPage = state.closestPageToPosition(state.anchorPosition!!)
            if (anchorPage?.prevKey != null) {
                return anchorPage.prevKey!!.plus(1)
            }
            else if (anchorPage?.nextKey != null) {
                return anchorPage.nextKey!!.minus(1)
            }
        }
        else {
            return null  //cause, params.key = null
        }
        */

    }
}
```

QuoteRemoteMediator.kt:

```
package com.ghani.`paging3`.paging

import androidx.paging.ExperimentalPagingApi
import androidx.paging.LoadType
import androidx.paging.PagingState
import androidx.paging.RemoteMediator
import androidx.room.withTransaction
import com.ghani.paging3.database.QuoteDatabase
import com.ghani.paging3.model.QuoteRemoteKeys
import com.ghani.paging3.network.QuoteAPI
import com.ghani.paging3.model.Result

@ExperimentalPagingApi
class QuoteRemoteMediator(
    private val quoteApi: QuoteAPI,
    private val quoteDatabase: QuoteDatabase,
) : RemoteMediator<Int, Result>() {
```

```
    val quoteDao = quoteDatabase.quoteDao()
    val quoteRemoteKeysDao = quoteDatabase.remoteKeysDao()

    override suspend fun load(loadType: LoadType, state: PagingState<Int, Result>):
MediatorResult {
        return try {

            val currentPage = when (loadType) {
                LoadType.REFRESH -> {
                    val remoteKeys = getRemoteKeyClosetToCurrentPosition(state)
                    remoteKeys?.nextPage?.minus(1) ?: 1
                }
                LoadType.PREPEND -> {
                    val remoteKeys = getRemoteKeyForFirstItem(state)
                    val prevPage = remoteKeys?.prevPage
                        ?: return MediatorResult.Success(
                            endOfPaginationReached = remoteKeys != null
                        )
                    prevPage
                }
                LoadType.APPEND -> {
                    val remoteKeys = getRemoteKeyForLastItem(state)
                    val nextPage = remoteKeys?.nextPage
                        ?: return MediatorResult.Success(
                            endOfPaginationReached = remoteKeys != null
                        )
                    nextPage
                }
            }

            val response = quoteApi.getQuotes(currentPage)
            val endOfPaginationReached = response.totalPages == currentPage

            val prevPage = if (currentPage == 1) null else currentPage - 1
            val nextPage = if (endOfPaginationReached) null else currentPage + 1

            quoteDatabase.withTransaction {

                if (loadType == LoadType.REFRESH) {
                    quoteDao.deleteQuotes()
                    quoteRemoteKeysDao.deleteAllRemoteKeys()
                }
```

```
            quoteDao.addQuotes(response.results)
            val keys = response.results.map { quote ->
                QuoteRemoteKeys(
                    id = quote._id,
                    prevPage = prevPage,
                    nextPage = nextPage
                )
            }
            quoteRemoteKeysDao.addAllRemoteKeys(keys)
        }
        MediatorResult.Success(endOfPaginationReached)
    } catch (e: Exception) {
        MediatorResult.Error(e)
    }
  }

  private suspend fun getRemoteKeyClosetToCurrentPosition(state: PagingState<Int, Result>):
QuoteRemoteKeys? {
    return state.anchorPosition?.let { position ->
        state.closestItemToPosition(position)?._id?.let { id ->
            quoteRemoteKeysDao.getRemoteKeys(id = id)
        }
    }
  }

  private suspend fun getRemoteKeyForFirstItem(state: PagingState<Int, Result>):
QuoteRemoteKeys? {
    return state.pages.firstOrNull{it.data.isNotEmpty()}?.data?.firstOrNull()?.let { quote ->
        quoteRemoteKeysDao.getRemoteKeys(id = quote._id)
    }
  }

  private suspend fun getRemoteKeyForLastItem(state: PagingState<Int, Result>):
QuoteRemoteKeys? {
    return state.pages.lastOrNull{it.data.isNotEmpty()}?.data?.lastOrNull()?.let { quote ->
        quoteRemoteKeysDao.getRemoteKeys(id = quote._id)
    }
  }

}
```

QuoteRepository.kt:

```
package com.ghani.paging3.repository

import androidx.paging.ExperimentalPagingApi
import androidx.paging.Pager
import androidx.paging.liveData
import androidx.paging.PagingConfig
import com.ghani.paging3.database.QuoteDatabase
import com.ghani.paging3.paging.QuoteRemoteMediator
import com.ghani.paging3.network.QuoteAPI
import com.ghani.paging3.paging.QuotePagingSource
import javax.inject.Inject


/*
//Flow-1(Data fetch from API)
class QuoteRepository @Inject constructor(private val quoteAPI: QuoteAPI) {

   fun getQuotes() = Pager(
      config = PagingConfig(pageSize = 20, maxSize = 100),
      pagingSourceFactory = { QuotePagingSource(quoteAPI) }
   ).liveData
}
*/


//Flow-2(Data fetch from DB)
@ExperimentalPagingApi  //We it write here,because its written in QuoteRemoteMediator
class QuoteRepository @Inject constructor(private val quoteAPI: QuoteAPI, private val
quoteDatabase: QuoteDatabase) {

   fun getQuotes() = Pager(
      config = PagingConfig(pageSize = 20, maxSize = 100),
      remoteMediator = QuoteRemoteMediator(quoteAPI,quoteDatabase),
      pagingSourceFactory = {quoteDatabase.quoteDao().getQuotes()}
   ).liveData


}
```

QuoteViewModel.kt:

```
package com.ghani.paging3.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.paging.ExperimentalPagingApi
import androidx.paging.cachedIn
import com.ghani.paging3.repository.QuoteRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import javax.inject.Inject

@ExperimentalPagingApi  //We it write here,because its written in QuoteRemoteMediator
@HiltViewModel
class QuoteViewModel @Inject constructor(private val repository: QuoteRepository) :
ViewModel() {
   val list = repository.getQuotes().cachedIn(viewModelScope)
}
```

QuotePagingAdapter.kt:

```
package com.ghani.paging3.paging

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import com.ghani.paging3.model.Result
import androidx.paging.PagingDataAdapter
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.RecyclerView
import com.ghani.paging3.R

class QuotePagingAdapter:PagingDataAdapter<Result,QuotePagingAdapter.QuoteViewHolder>
(COMPARATOR) {

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


   override fun onBindViewHolder(holder: QuoteViewHolder, position: Int) {
      val item = getItem(position)
      if (item != null){
         holder.quote.text = item.content
      }
   }
```

```
    class QuoteViewHolder(itemView: View):RecyclerView.ViewHolder(itemView){
        val quote = itemView.findViewById<TextView>(R.id.quote)
    }

    companion object{
        private val COMPARATOR = object : DiffUtil.ItemCallback<Result>(){
            override fun areItemsTheSame(oldItem: Result, newItem: Result): Boolean {
                return oldItem._id == newItem._id
            }
            override fun areContentsTheSame(oldItem: Result, newItem: Result): Boolean {
                return oldItem == newItem
            }
        }
    }

}
```

LoaderAdapter.kt:

```
package com.ghani.paging3.paging

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ProgressBar
import androidx.core.view.isVisible
import androidx.paging.LoadState
import androidx.paging.LoadStateAdapter
import androidx.recyclerview.widget.RecyclerView
import com.ghani.`paging3`.R

class LoaderAdapter():LoadStateAdapter<LoaderAdapter.LoaderViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, loadState: LoadState):
LoaderViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.loader_item,parent,false)
        return LoaderViewHolder(view)
    }

    override fun onBindViewHolder(holder: LoaderViewHolder, loadState: LoadState) {
        holder.bind(loadState)
    }
```

```
    class LoaderViewHolder(itemView: View):RecyclerView.ViewHolder(itemView) {
        val progressBar = itemView.findViewById<ProgressBar>(R.id.progress_bar)

        fun bind(loadState: LoadState){
            progressBar.isVisible = loadState is  LoadState.Loading
        }
    }

}
```

QuoteApplication.kt:

```
package com.ghani.paging3.scope

import android.app.Application
import dagger.hilt.android.HiltAndroidApp

@HiltAndroidApp
class QuoteApplication:Application() {
}
```

MainActivity.kt:

```
package com.ghani.paging3

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import androidx.paging.ExperimentalPagingApi
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.ghani.paging3.paging.LoaderAdapter
import com.ghani.paging3.paging.QuotePagingAdapter
import com.ghani.paging3.viewmodel.QuoteViewModel
import dagger.hilt.android.AndroidEntryPoint

@ExperimentalPagingApi  //We it write here,because its written in QuoteRemoteMediator
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    lateinit var quoteViewModel: QuoteViewModel
    lateinit var recyclerView: RecyclerView
    lateinit var adapter: QuotePagingAdapter
```

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

        quoteViewModel = ViewModelProvider(this).get(QuoteViewModel::class.java)
        recyclerView = findViewById(R.id.quoteList)
        adapter = QuotePagingAdapter()

        recyclerView.layoutManager = LinearLayoutManager(this)
        recyclerView.setHasFixedSize(true)
        recyclerView.adapter = adapter.withLoadStateHeaderAndFooter(
            header = LoaderAdapter(),
            footer = LoaderAdapter()
        )

        quoteViewModel.list.observe(this, Observer {
            adapter.submitData(lifecycle, it)
        })
    }
}
```