Android Applications Development

09 – Coroutines

Objectives

- Introduction to Kotlin Coroutine
- Application of Coroutine to Android development
- Introduction on how to connect to the network using http client and perform network operation

Credits

- The following slides are extracted from :
  - https://developer.android.com/training/basics/network-ops/index.html

From the beginning

- GSM
- GPRS
- WAP
- SMS / MMS
- 3G / 3.5G
- GPS / A-GPS
- What are they ?

Connectivity

- Web service
  - Connecting to SQL servers
- Non-relational databases (Firebase)

Connecting …

- Manifest must include the following permissions :

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

- By default, Android 3.0 (API level 11) and higher requires you to perform network operations on a thread other than the main UI thread;
- To avoid creating an unresponsive UI, don't perform network operations on the UI thread.
- if you don't, a NetworkOnMainThreadException is thrown.

KOTLIN

Coroutines

Background Services

- It is crucial that any long running tasks are performed on a separate thread from the main (UI) thread
  - otherwise your application will appear
    - slow
    - unresponsive
  - may be terminated by the operating system

Coroutine

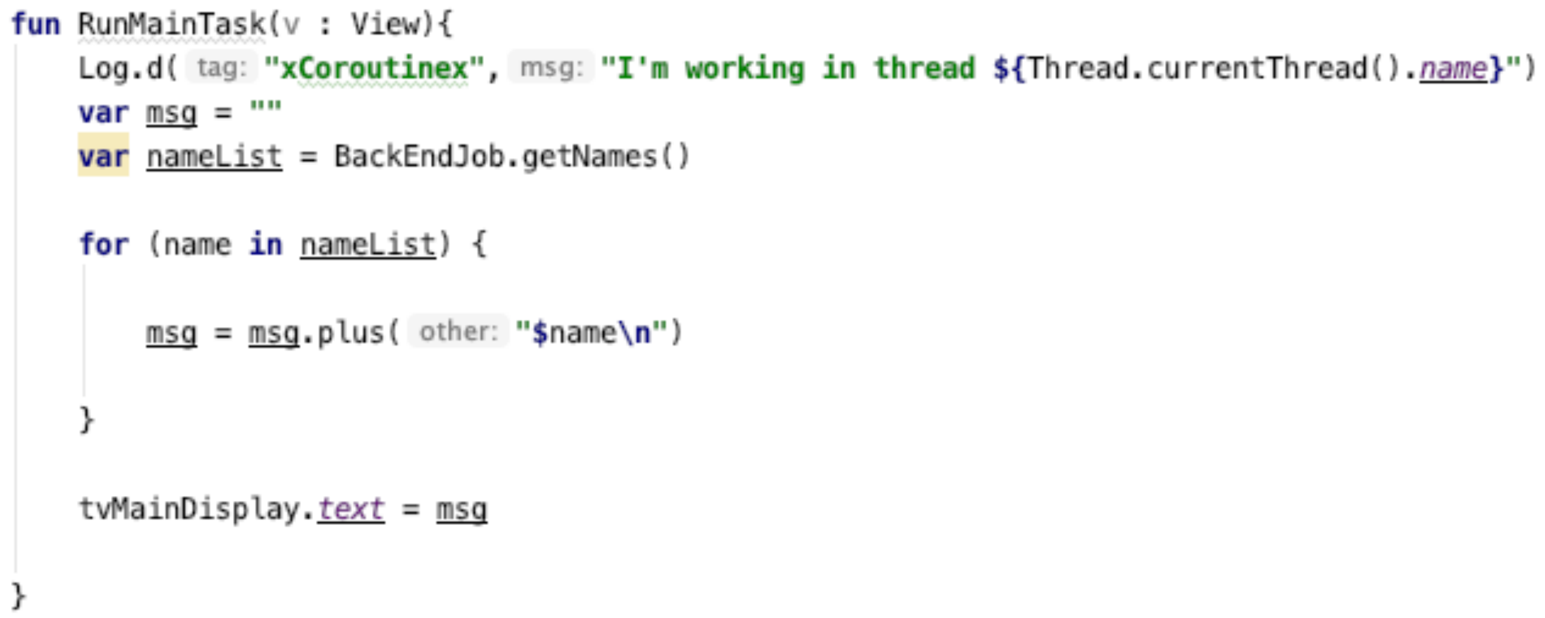
- On Android, every app has a main thread that is in charge of handling UI (like drawing views) and coordinating user interactions.
  - If there is too much work happening on this thread, the app appears to hang or slow down, leading to an undesirable user experience. Any long running task should be done without blocking the main thread, so your app doesn’t display frozen animations, or respond slowly to touch events.

Coroutine

- Kotlin coroutines introduces a new style of concurrency that can be used on Android to simplify async code.
- Coroutines solves
  - Long running tasks
    - Tasks that take too long to block the main thread.
  - Main-Safety
    - Allows you to ensure that an suspend function can be called from the main thread

Long running task

BackEndJob.getNames() //a task that takes 5 secs



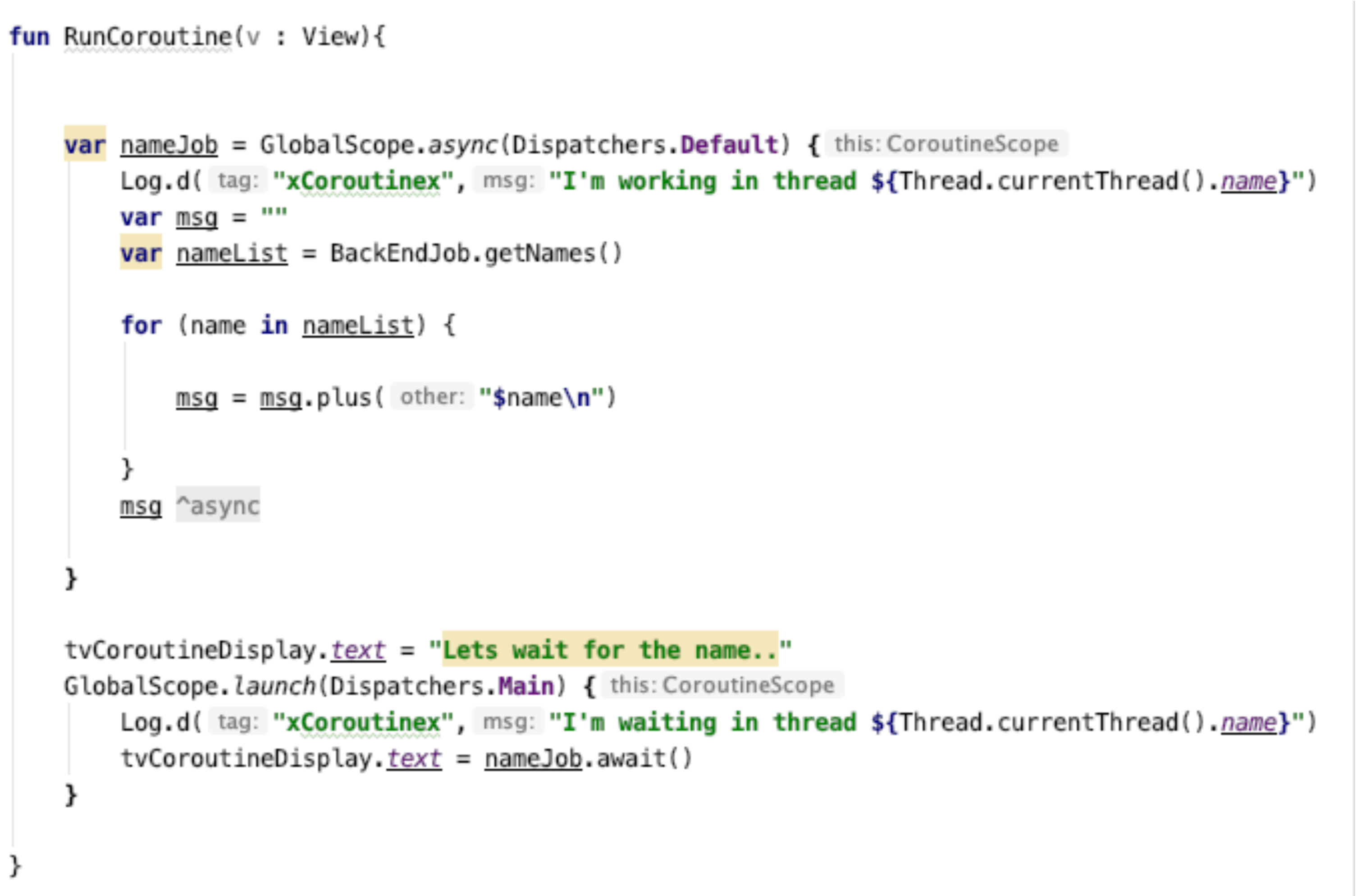
```
fun RunMainTask(v : View){
    Log.d( tag: "xCoroutinex", msg: "I'm working in thread ${Thread.currentThread().name}")
    var msg = ""
    var nameList = BackEndJob.getNames()

    for (name in nameList) {

        msg = msg.plus( other: "$name\n")

    }

    tvMainDisplay.text = msg

}
```

Result

- RunMainTask causes non responsiveness
- After invoking “BackEndJob.getNames()”, the main task stays within the method to be working.
- The main task will not be able to work on other tasks.
  - E.g. tapping on other buttons.

Long running task

BackEndJob.getNames() //a task that takes 5 secs

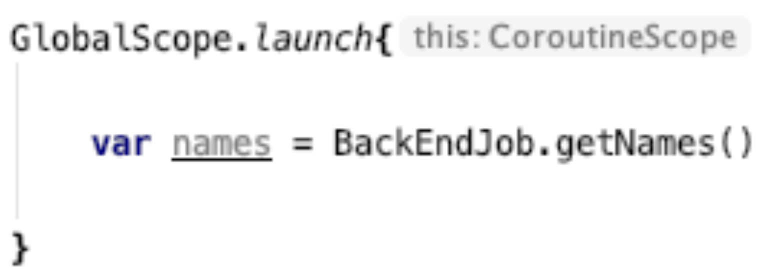


```
fun RunCoroutine(v : View){


    var nameJob = GlobalScope.async(Dispatchers.Default) { this: CoroutineScope
        Log.d( tag: "xCoroutinex", msg: "I'm working in thread ${Thread.currentThread().name}")
        var msg = ""
        var nameList = BackEndJob.getNames()

        for (name in nameList) {

            msg = msg.plus( other: "$name\n")

        }
        msg ^async

    }

    tvCoroutineDisplay.text = "Lets wait for the name.."
    GlobalScope.launch(Dispatchers.Main) { this: CoroutineScope
        Log.d( tag: "xCoroutinex", msg: "I'm waiting in thread ${Thread.currentThread().name}")
        tvCoroutineDisplay.text = nameJob.await()
    }

}
```

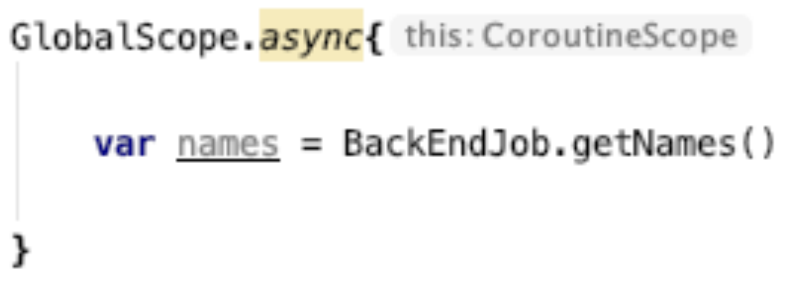
Result

- RunCoroutine does not cause non responsiveness
- After invoking coroutine, the main task goes on to other codes.

How it works



```
GlobalScope.launch{ this: CoroutineScope

    var names = BackEndJob.getNames()

}
```



```
GlobalScope.async{ this: CoroutineScope

    var names = BackEndJob.getNames()

}
```

- Global scope is used to launch top-level coroutines which are operating on the whole application lifetime and are not cancelled prematurely.

There are two ways to start coroutines, and they have different uses:

- launch builder will start a new coroutine that is “fire and forget” — that means it won’t return the result to the caller.
- async builder will start a new coroutine, and it allows you to return a result with a suspend function called await.

Scope, Job and Dispatcher

- Scope
  - A CoroutineScope keeps tracks of any coroutine it creates using launch or async.
- Job
  - A Job is a handle to a coroutine. Each coroutine that you create with launch or async returns a Job instance that uniquely identifies the coroutine and manages its lifecycle.
- Dispatcher
  - Kotlin coroutines use dispatchers to determine which threads are used for coroutine execution. All coroutines must run in a dispatcher.

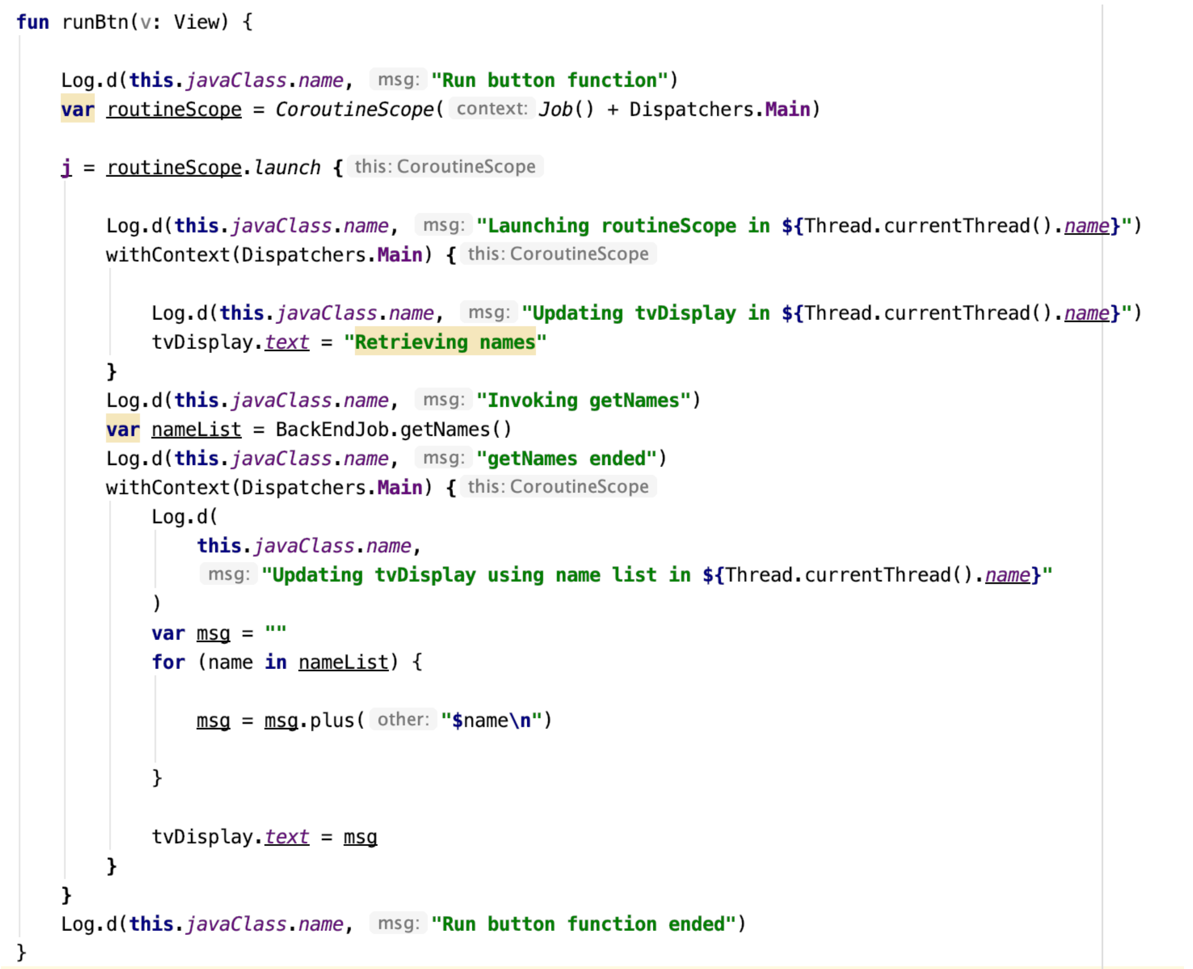
Dispatchers

- Dispatchers.Main
  - Use this dispatcher to run a coroutine on the main Android thread.
  - This should be used only for interacting with the UI and performing quick work.
- Dispatchers.IO
  - This dispatcher is optimized to perform disk or network I/O outside of the main thread.
- Dispatchers.Default
  - This dispatcher is optimized to perform CPU-intensive work outside of the main thread.

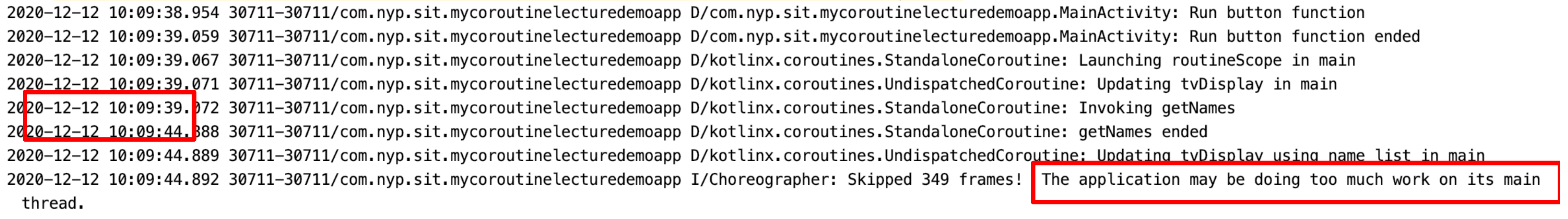
Structured concurrency

- Coroutine provides structured concurrency in Kotlin
- Structured concurrency needs to accomplish three things
  - Cancel work
  - Keep track of work while it’s running
  - Signal errors when a coroutine fails
- To accomplish this structured concurrency gives us some guarantees about our code. Here are the guarantees of structured concurrency.
  - When a **scope cancels**, all of its **coroutines cancel**.
  - When a **suspend fun returns**, all of its **work is done**.
  - When a **coroutine errors**, its **caller or scope is notified**.

How it works



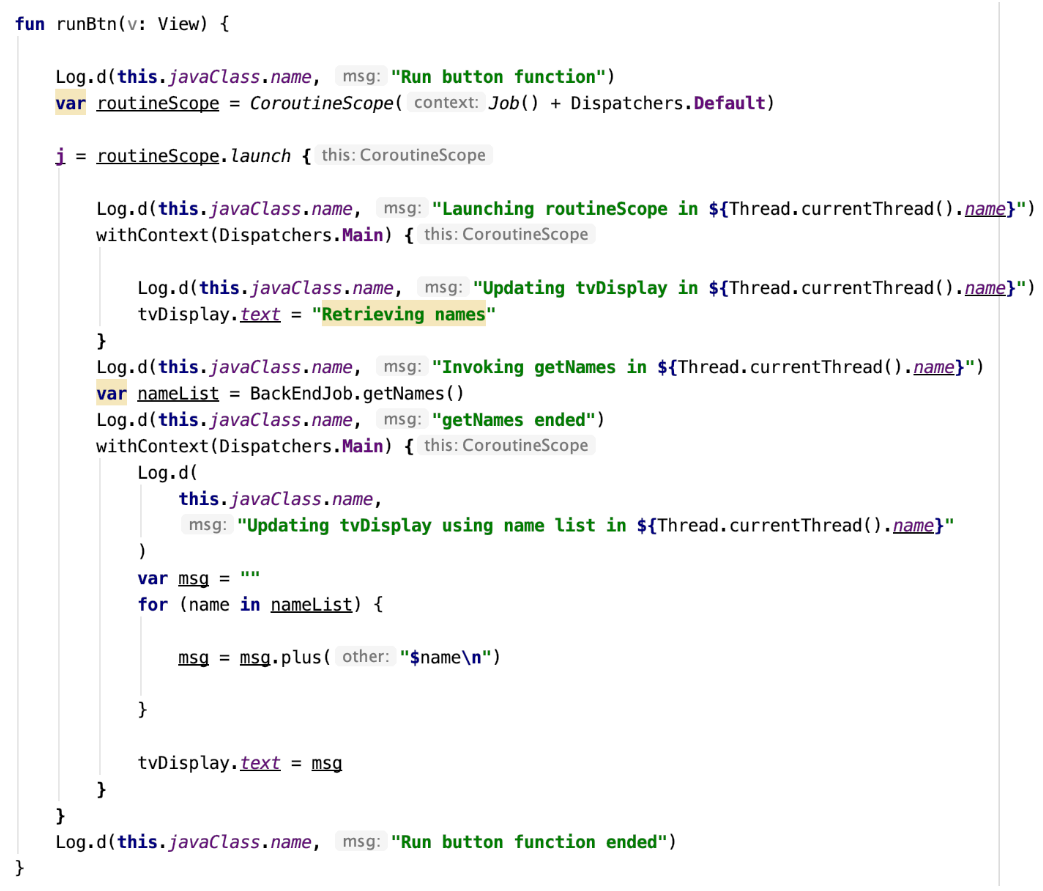
```
fun runBtn(v: View) {

    Log.d(this.javaClass.name,  msg: "Run button function")
    var routineScope = CoroutineScope( context: Job() + Dispatchers.Main)

    j = routineScope.launch { this: CoroutineScope

        Log.d(this.javaClass.name,  msg: "Launching routineScope in ${Thread.currentThread().name}")
        withContext(Dispatchers.Main) { this: CoroutineScope

            Log.d(this.javaClass.name,  msg: "Updating tvDisplay in ${Thread.currentThread().name}")
            tvDisplay.text = "Retrieving names"
        }
        Log.d(this.javaClass.name,  msg: "Invoking getNames")
        var nameList = BackEndJob.getNames()
        Log.d(this.javaClass.name,  msg: "getNames ended")
        withContext(Dispatchers.Main) { this: CoroutineScope
            Log.d(
                this.javaClass.name,
                 msg: "Updating tvDisplay using name list in ${Thread.currentThread().name}"
            )
            var msg = ""
            for (name in nameList) {

                msg = msg.plus( other: "$name\n")

            }

            tvDisplay.text = msg
        }
    }
    Log.d(this.javaClass.name,  msg: "Run button function ended")
}
```



```
2020-12-12 10:09:38.954 30711-30711/com.nyp.sit.mycoroutinelecturedemoapp D/com.nyp.sit.mycoroutinelecturedemoapp.MainActivity: Run button function
2020-12-12 10:09:39.059 30711-30711/com.nyp.sit.mycoroutinelecturedemoapp D/com.nyp.sit.mycoroutinelecturedemoapp.MainActivity: Run button function ended
2020-12-12 10:09:39.067 30711-30711/com.nyp.sit.mycoroutinelecturedemoapp D/kotlinx.coroutines.StandaloneCoroutine: Launching routineScope in main
2020-12-12 10:09:39.071 30711-30711/com.nyp.sit.mycoroutinelecturedemoapp D/kotlinx.coroutines.UndispatchedCoroutine: Updating tvDisplay in main
2020-12-12 10:09:39.072 30711-30711/com.nyp.sit.mycoroutinelecturedemoapp D/kotlinx.coroutines.StandaloneCoroutine: Invoking getNames
2020-12-12 10:09:44.888 30711-30711/com.nyp.sit.mycoroutinelecturedemoapp D/kotlinx.coroutines.StandaloneCoroutine: getNames ended
2020-12-12 10:09:44.889 30711-30711/com.nyp.sit.mycoroutinelecturedemoapp D/kotlinx.coroutines.UndispatchedCoroutine: Updating tvDisplay using name list in main
2020-12-12 10:09:44.892 30711-30711/com.nyp.sit.mycoroutinelecturedemoapp I/Choreographer: Skipped 349 frames!  The application may be doing too much work on its main
  thread.
```

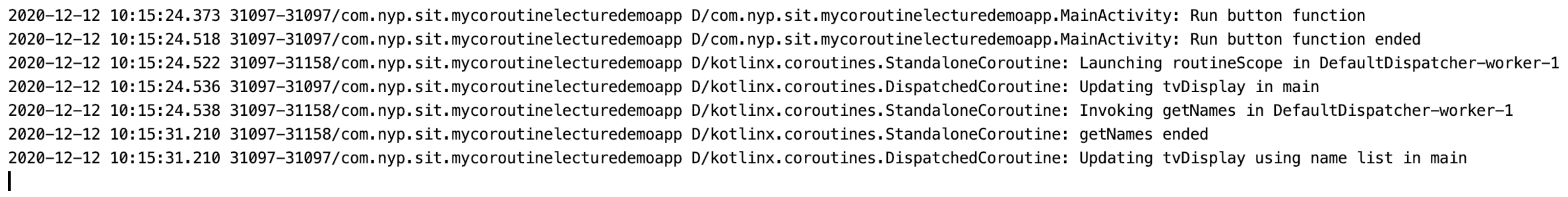
1. runBtn is invoked
2. Initialise routineScope
3. Launch routineScope
4. runBtn ends
5. routineScope runs
6. tvDisplay updated using Main dispatcher
7. Retrieve name
8. Update tvDisplay with name list using Main dispatcher

How it works



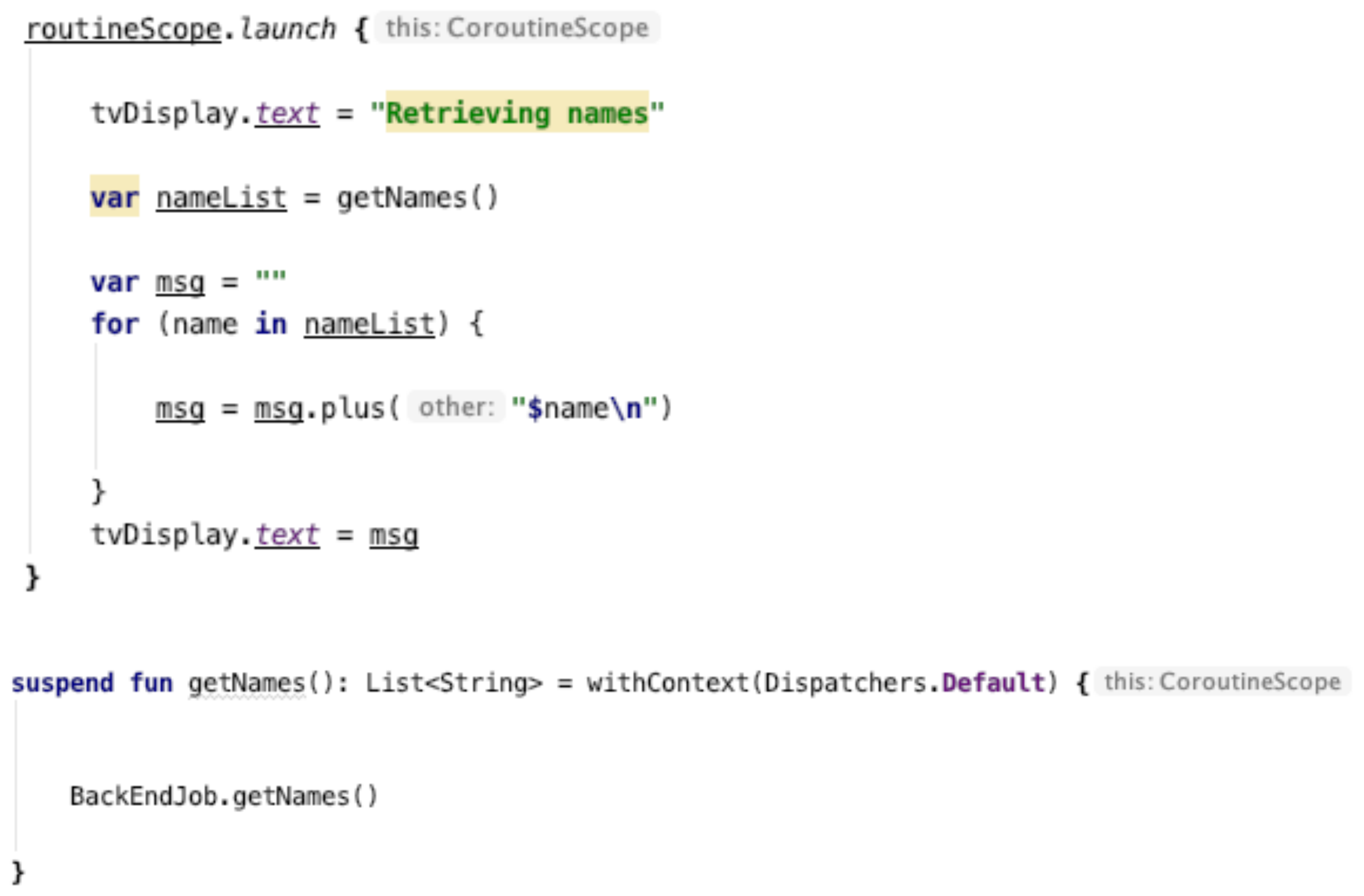
```
fun runBtn(v: View) {

    Log.d(this.javaClass.name,  msg: "Run button function")
    var routineScope = CoroutineScope( context: Job() + Dispatchers.Default)

    j = routineScope.launch { this: CoroutineScope

        Log.d(this.javaClass.name,  msg: "Launching routineScope in ${Thread.currentThread().name}")
        withContext(Dispatchers.Main) { this: CoroutineScope

            Log.d(this.javaClass.name,  msg: "Updating tvDisplay in ${Thread.currentThread().name}")
            tvDisplay.text = "Retrieving names"
        }
        Log.d(this.javaClass.name,  msg: "Invoking getNames in ${Thread.currentThread().name}")
        var nameList = BackEndJob.getNames()
        Log.d(this.javaClass.name,  msg: "getNames ended")
        withContext(Dispatchers.Main) { this: CoroutineScope
            Log.d(
                this.javaClass.name,
                 msg: "Updating tvDisplay using name list in ${Thread.currentThread().name}"
            )
            var msg = ""
            for (name in nameList) {

                msg = msg.plus( other: "$name\n")

            }

            tvDisplay.text = msg
        }
    }
    Log.d(this.javaClass.name,  msg: "Run button function ended")
}
```

1. runBtn is invoked
2. Initialise routineScope
3. Launch routineScope
4. runBtn ends
5. routineScope runs
6. tvDisplay updated using Main dispatcher
7. Retrieve name
8. Update tvDisplay with name list using Main dispatcher



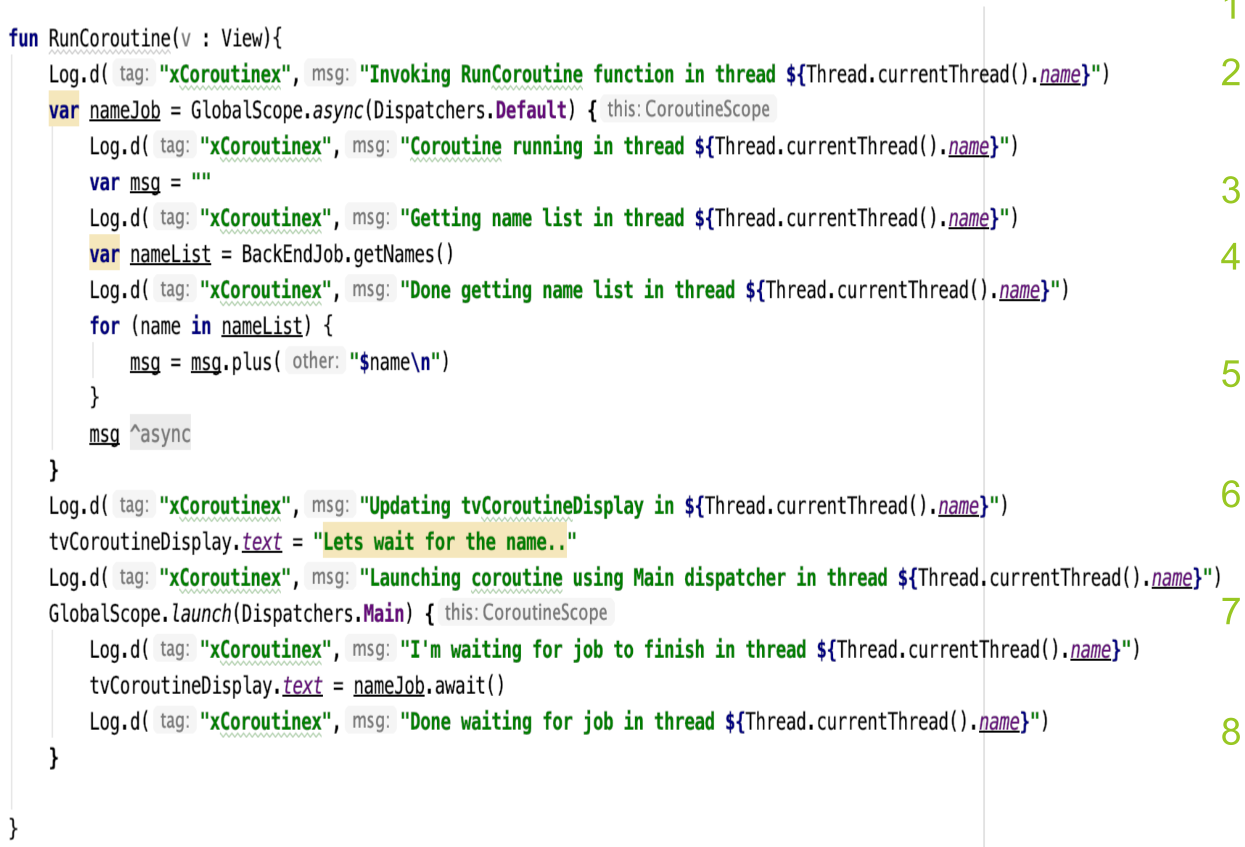
```
2020-12-12 10:15:24.373 31097-31097/com.nyp.sit.mycoroutinelecturedemoapp D/com.nyp.sit.mycoroutinelecturedemoapp.MainActivity: Run button function
2020-12-12 10:15:24.518 31097-31097/com.nyp.sit.mycoroutinelecturedemoapp D/com.nyp.sit.mycoroutinelecturedemoapp.MainActivity: Run button function ended
2020-12-12 10:15:24.522 31097-31158/com.nyp.sit.mycoroutinelecturedemoapp D/kotlinx.coroutines.StandaloneCoroutine: Launching routineScope in DefaultDispatcher-worker-1
2020-12-12 10:15:24.536 31097-31097/com.nyp.sit.mycoroutinelecturedemoapp D/kotlinx.coroutines.DispatchedCoroutine: Updating tvDisplay in main
2020-12-12 10:15:24.538 31097-31158/com.nyp.sit.mycoroutinelecturedemoapp D/kotlinx.coroutines.StandaloneCoroutine: Invoking getNames in DefaultDispatcher-worker-1
2020-12-12 10:15:31.210 31097-31158/com.nyp.sit.mycoroutinelecturedemoapp D/kotlinx.coroutines.StandaloneCoroutine: getNames ended
2020-12-12 10:15:31.210 31097-31097/com.nyp.sit.mycoroutinelecturedemoapp D/kotlinx.coroutines.DispatchedCoroutine: Updating tvDisplay using name list in main
```

How it works – suspend function



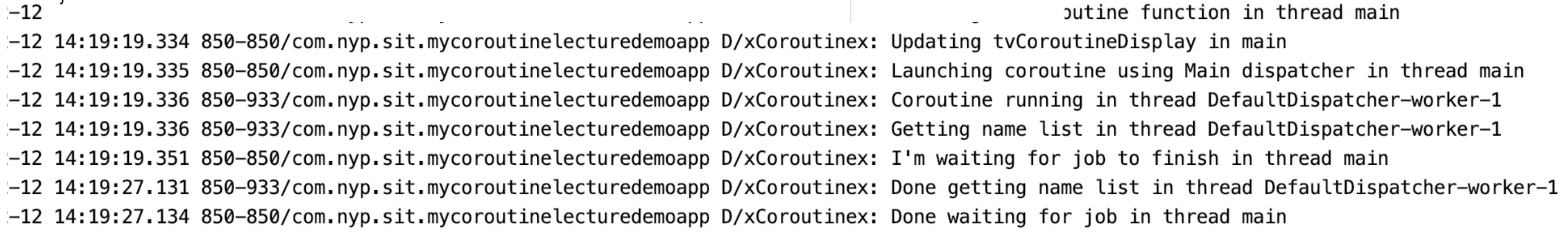
```
routineScope.launch { this: CoroutineScope

    tvDisplay.text = "Retrieving names"

    var nameList = getNames()

    var msg = ""
    for (name in nameList) {

        msg = msg.plus( other: "$name\n")

    }
    tvDisplay.text = msg
}


suspend fun getNames(): List<String> = withContext(Dispatchers.Default) { this: CoroutineScope


    BackEndJob.getNames()

}
```

How it works – Async



```
fun RunCoroutine(v : View){
    Log.d( tag: "xCoroutinex", msg: "Invoking RunCoroutine function in thread ${Thread.currentThread().name}")
    var nameJob = GlobalScope.async(Dispatchers.Default) { this: CoroutineScope
        Log.d( tag: "xCoroutinex", msg: "Coroutine running in thread ${Thread.currentThread().name}")
        var msg = ""
        Log.d( tag: "xCoroutinex", msg: "Getting name list in thread ${Thread.currentThread().name}")
        var nameList = BackEndJob.getNames()
        Log.d( tag: "xCoroutinex", msg: "Done getting name list in thread ${Thread.currentThread().name}")
        for (name in nameList) {
            msg = msg.plus( other: "$name\n")
        }
        msg ^async
    }
    Log.d( tag: "xCoroutinex", msg: "Updating tvCoroutineDisplay in ${Thread.currentThread().name}")
    tvCoroutineDisplay.text = "Lets wait for the name.."
    Log.d( tag: "xCoroutinex", msg: "Launching coroutine using Main dispatcher in thread ${Thread.currentThread().name}")
    GlobalScope.launch(Dispatchers.Main) { this: CoroutineScope
        Log.d( tag: "xCoroutinex", msg: "I'm waiting for job to finish in thread ${Thread.currentThread().name}")
        tvCoroutineDisplay.text = nameJob.await()
        Log.d( tag: "xCoroutinex", msg: "Done waiting for job in thread ${Thread.currentThread().name}")
    }

}
```

1. RunCoroutine is invoked
2. Initialise Async coroutine using Default dispatcher
3. Updating textView text
4. Launch coroutine using Main dispatcher
5. nameJob coroutine starts to retrieve name list
6. Main dispatcher coroutine waits for nameJob to finish
7. nameJob coroutine finish retrieving for name list
8. Main dispatcher coroutine updates text view.



```
-12                                                                          outine function in thread main
-12 14:19:19.334 850-850/com.nyp.sit.mycoroutinelecturedemoapp D/xCoroutinex: Updating tvCoroutineDisplay in main
-12 14:19:19.335 850-850/com.nyp.sit.mycoroutinelecturedemoapp D/xCoroutinex: Launching coroutine using Main dispatcher in thread main
-12 14:19:19.336 850-933/com.nyp.sit.mycoroutinelecturedemoapp D/xCoroutinex: Coroutine running in thread DefaultDispatcher-worker-1
-12 14:19:19.336 850-933/com.nyp.sit.mycoroutinelecturedemoapp D/xCoroutinex: Getting name list in thread DefaultDispatcher-worker-1
-12 14:19:19.351 850-850/com.nyp.sit.mycoroutinelecturedemoapp D/xCoroutinex: I'm waiting for job to finish in thread main
-12 14:19:27.131 850-933/com.nyp.sit.mycoroutinelecturedemoapp D/xCoroutinex: Done getting name list in thread DefaultDispatcher-worker-1
-12 14:19:27.134 850-850/com.nyp.sit.mycoroutinelecturedemoapp D/xCoroutinex: Done waiting for job in thread main
```

NETWORKING

Communicating

Building Url

```
val FORECASE_OPEN_WEATHER_URL= "http://api.openweathermap.org/data/2.5/forecast"
val QUERY_PARAM = "q"
val FORMAT_PARAM = "mode"
val UNITS_PARAM = "units"
val DAYS_PARAM = "cnt"
val API_KEY="APPID"


fun buildUrl(locationQuery: String): URL {
   // COMPLETED (1) Fix this method to return the URL used to query Open Weather Map's API

   val builtUri = Uri.parse(FORECASE_OPEN_WEATHER_URL).buildUpon()
         .appendQueryParameter(QUERY_PARAM, locationQuery)
         .appendQueryParameter(FORMAT_PARAM, format)
         .appendQueryParameter(UNITS_PARAM, units)
         .appendQueryParameter(DAYS_PARAM, Integer.toString(numDays))
         .appendQueryParameter(API_KEY,"xxxxxxxx")
         .build()

   var url: URL? = null
   try {
      url = URL(builtUri.toString())
   } catch (e: MalformedURLException) {
      e.printStackTrace()
   }

   Log.v(TAG, "Built URI " + url!!)

   return url
}
```

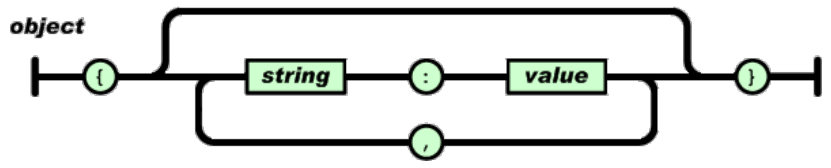
Values :
http://api.openweathermap.org/data/2.5/forecast?q=Singapore&mode=json&units=metric&cnt=1&APPID=0e0d7490f406a855e2bc
bfdf5b4cc0e8

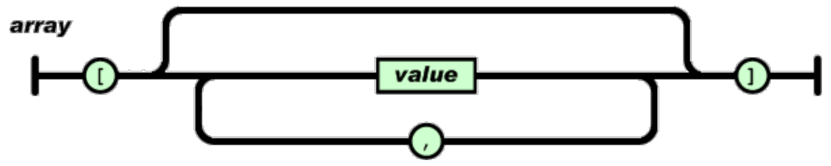
Fetching data

```
fun getResponseFromHttpUrl(url: URL): String? {
    val urlConnection = url.openConnection() as HttpURLConnection
    try {
        val `in` = urlConnection.getInputStream()

        val scanner = Scanner(`in`)
        scanner.useDelimiter("\\A")


        val hasInput = scanner.hasNext()
        return if (hasInput) {
            scanner.next()
        } else {
            null
        }
    }
    catch (ex : Exception)
    {

        Log.d("App",ex.toString())


    }
    finally {
        urlConnection.disconnect()
    }

    return null
}
```

Json

- JavaScript Object Notation
- Lightweight data-intercahnge format
- Built on two structure
  - Collection of name/value pairs
  - Ordered list of values





Example

Array

Objects

```
{
"employees": [
  { "firstName":"John" , "lastName":"Doe" },
  { "firstName":"Anna" , "lastName":"Smith" },
  { "firstName":"Peter" , "lastName":"Jones" }
]
}
```

Example

```
[
        {
                "symbol": "ABC",
                "price": 42.226532717702945,
                "change": -0.3541920256958341
        },
]
```

Android . . .

- Makes use of JSONArray and JSONObject object.

JSONArray jArr = **JSONArray(responseBody);**

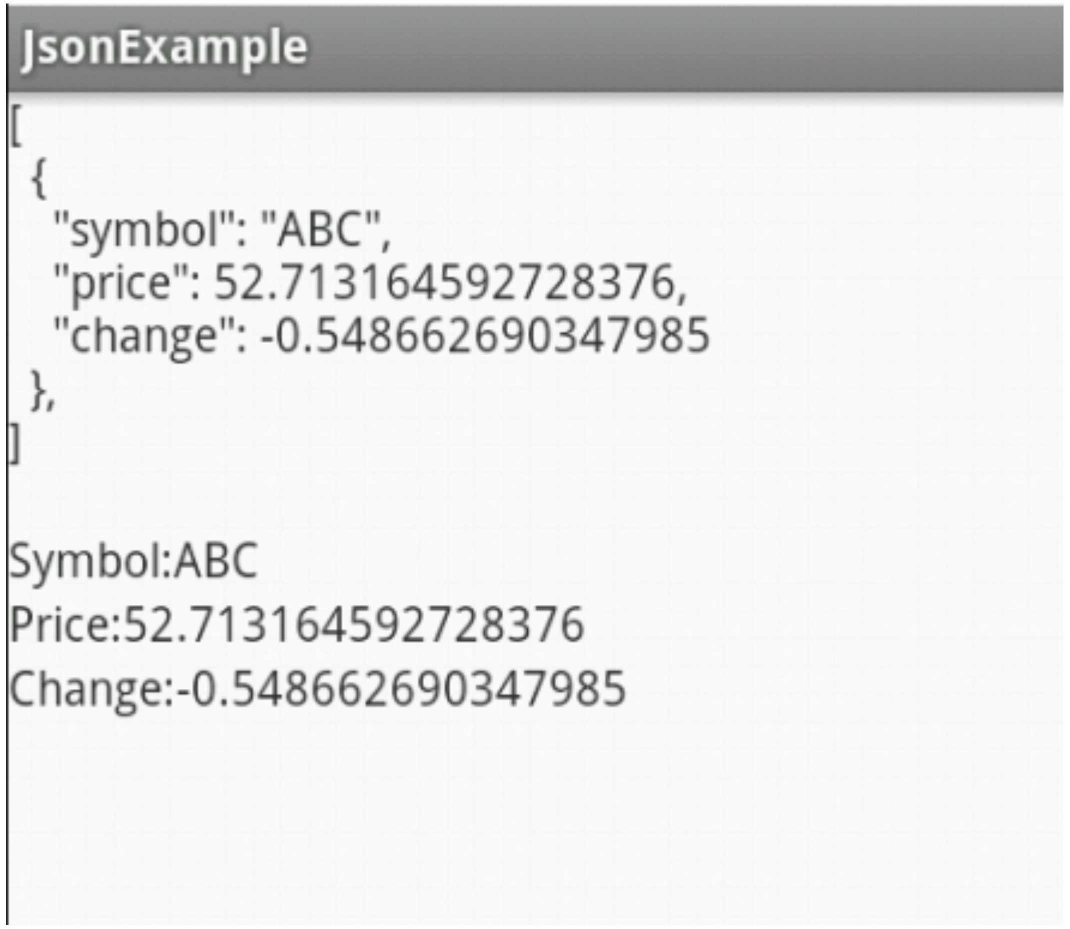
JSONObject jObj = jArr.**getJSONObject**(0);

String Symbol ="Symbol:"+ jObj.getString("symbol");

String price = "Price:" + jObj.getString("price");

String change = "Change:" + jObj.getString("change");

Example



```
[
 {
  "symbol": "ABC",
  "price": 52.713164592728376,
  "change": -0.548662690347985
 },
]

Symbol:ABC
Price:52.713164592728376
Change:-0.548662690347985
```

Example 2 (Json)

```
{"cod":"200","message":0.0057,"cnt":1,

"list":[{"dt":1548039600,
"main":{"temp":27.68,"temp_min":27.68,"temp_max":28.46,"pressure":1025.06,"sea
_level":1025.76,"grnd_level":1025.06,"humidity":91,"temp_kf":-0.78},

"weather":[{"id":801,"main":"Clouds","description":"few clouds","icon":"02d"}],

"clouds":{"all":12},
"wind":{"speed":5.83,"deg":336.503},
"sys":{"pod":"d"},
"dt_txt":"2019-01-21 03:00:00"}],

"city":{"id":1880252,"name":"Singapore",
"coord":{"lat":1.2905,"lon":103.852},"country":"MY","population":3547809}
}
```

Example 2 (Json)

```
val OWM_LIST = "list"

val OWM_PRESSURE = "pressure"
val OWM_HUMIDITY = "humidity"
val OWM_WINDSPEED = "speed"
val OWM_WIND_DIRECTION = "deg"

val OWM_TEMPERATURE = "main"

val OWM_MAX = "temp_max"
val OWM_MIN = "temp_min"

val OWM_WEATHER = "weather"
val OWM_DESCRIPTION = "main"
val OWM_WEATHER_ID = "id"
val OWM_MESSAGE_CODE = "cod"
```

Example 2 (Json)

```
val forecastJson = JSONObject(forecastJsonStr)

val weatherArray =
forecastJson.getJSONArray(OWM_LIST)

for (i in 0 until weatherArray.length()) {

….
}
```

Breakdown ..

```
val OWM_LIST = "list”
…
val weatherArray = forecastJson.getJSONArray(OWM_LIST)
```

Value of weatherArray =>

```
"list":[{"dt":1548039600,
"main":{"temp":27.68,"temp_min":27.68,"temp_max":28.46,"pressure":1025
.06,"sea_level":1025.76,"grnd_level":1025.06,"humidity":91,"temp_kf":-
0.78},
"weather":[{"id":801,"main":"Clouds","description":"few
clouds","icon":"02d"}],
"clouds":{"all":12},
"wind":{"speed":5.83,"deg":336.503},
"sys":{"pod":"d"},
"dt_txt":"2019-01-21 03:00:00"}]
```

Example 2 (Json)

```
val forecastJson = JSONObject(forecastJsonStr)

val weatherArray =
forecastJson.getJSONArray(OWM_LIST)

for (i in 0 until weatherArray.length()) {

….
}
```

Example 2

```
/* Get the JSON object representing the day */
val dayForecast = weatherArray.getJSONObject(i)
```

"list":[{"dt":1548039600,
"main":{"temp":27.68,"temp_min":27.68,"temp_max":28.46,"pressure":1025.06,"sea_level":1025.76,"grnd_level":1025.06,"humidity":91,"temp_kf":-0.78},
"weather":[{"id":801,"main":"Clouds","description":"few clouds","icon":"02d"}],
"clouds":{"all":12},
"wind":{"speed":5.83,"deg":336.503},
"sys":{"pod":"d"},
"dt_txt":"2019-01-21 03:00:00"}]

Breakdown

```
val weatherObject = dayForecast.getJSONArray(OWM_WEATHER).getJSONObject(0)
description = weatherObject.getString(OWM_DESCRIPTION)
weatherId = weatherObject.getInt(OWM_WEATHER_ID)
```

"list":[{"dt":1548039600,
"main":{"temp":27.68,"temp_min":27.68,"temp_max":28.46,"pressure":1025.06,"sea_level":1025.76,"grnd_level":1025.06,"humidity":91,"temp_kf":-0.78},
"weather":[{"id":801,"main":"Clouds","description":"few clouds","icon":"02d"}],
"clouds":{"all":12},
"wind":{"speed":5.83,"deg":336.503},
"sys":{"pod":"d"},
"dt_txt":"2019-01-21 03:00:00"}]

Values :

```
weatherObject = {"id":801,"main":"Clouds","description":"few clouds","icon":"02d"}
description = “few clouds”
weatherId = “801”
```

Breakdown

```
temperatureObject = dayForecast.getJSONObject("main")
high = temperatureObject.getDouble(OWM_MAX)
low = temperatureObject.getDouble(OWM_MIN)
pressure = temperatureObject.getDouble(OWM_PRESSURE).toFloat()
humidity = temperatureObject.getInt(OWM_HUMIDITY).toFloat()
```

```
"list":[{"dt":1548039600,
"main":{"temp":27.68,"temp_min":27.68,"temp_max":28.46,"pressure":1025.06,"sea_level":102
5.76,"grnd_level":1025.06,"humidity":91,"temp_kf":-0.78},
"weather":[{"id":801,"main":"Clouds","description":"few clouds","icon":"02d"}],
"clouds":{"all":12},
"wind":{"speed":5.83,"deg":336.503},
"sys":{"pod":"d"},
"dt_txt":"2019-01-21 03:00:00"}]
```

Values:

```
temperatureObject =
"temp":27.68,"temp_min":27.68,"temp_max":28.46,"pressure":1025.06,"sea_level":1025.76,"gr
nd_level":1025.06,"humidity":91,"temp_kf":-0.78
high = 28.46
low = 27.68
pressure = 1025.06
humidity = 91
```

Breakdown

```
windObject = dayForecast.getJSONObject("wind")
windSpeed = windObject.getDouble(OWM_WINDSPEED).toFloat()
windDirection = windObject.getDouble(OWM_WIND_DIRECTION).toFloat()
```

```
"list":[{"dt":1548039600,
"main":{"temp":27.68,"temp_min":27.68,"temp_max":28.46,"pressure":1025.06,"sea
_level":1025.76,"grnd_level":1025.06,"humidity":91,"temp_kf":-0.78},
"weather":[{"id":801,"main":"Clouds","description":"few clouds","icon":"02d"}],
"clouds":{"all":12},
"wind":{"speed":5.83,"deg":336.503},
"sys":{"pod":"d"},
"dt_txt":"2019-01-21 03:00:00"}]
```

Values: windObject = {"speed":5.83,"deg":336.503}
windSpeed = 5.83
windDirection = 336.503

Example 2

```
val dayForecast = weatherArray.getJSONObject(i)

val weatherObject =
dayForecast.getJSONArray(OWM_WEATHER).getJSONObject(0)
description = weatherObject.getString(OWM_DESCRIPTION)
weatherId = weatherObject.getInt(OWM_WEATHER_ID)

val temperatureObject = dayForecast.getJSONObject("main")
high = temperatureObject.getDouble(OWM_MAX)
low = temperatureObject.getDouble(OWM_MIN)
pressure = temperatureObject.getDouble(OWM_PRESSURE).toFloat()
humidity = temperatureObject.getInt(OWM_HUMIDITY).toFloat()

val windObject = dayForecast.getJSONObject("wind")
windSpeed = windObject.getDouble(OWM_WINDSPEED).toFloat()
windDirection = windObject.getDouble(OWM_WIND_DIRECTION).toFloat()

//Add to list / array
parsedWeatherData.add(WeatherEntry(date,weatherId,high,low,humidity,pressure,
windSpeed,windDirection))
```