# Kotlin Coroutines

## 1 – What is the mean of asynchronous programming?

- The ability of performing the heavy tasks that take more than 5 seconds into the background, this improve the application's performance and prevent the problem of ANR (App not response) that happen when performing a task into the main thread and this task take a long time.

## 2 – What is the different between concurrency and parallelism?

- Say that we have 3 tasks and we want to perform these tasks into the background.
- When using the **parallelism** each task will perform into separated thread, this is very expensive solutions and consume the device resources.
- When using the **concurrency** we can perform the 3 tasks into the same back thread.
- **To achieve concurrent we need three things:**
    1. Stop/resume technique to make this task able to work into the background.
    2. Controller to move from one thread to another.
    3. Background thread to work in it.

## 3 – What are the libraries that achieve the asynchronous programming?

- **Rx-Java** Achieve the asynchronous programming with Parallelism
- **Coroutines** Achieve the asynchronous programming with Concurrency

## 4 – How Coroutines achieve the concurrency?

- Coroutines key **Suspend** this is the (Stop/resume technique) and it responsible for make the function able to work into the background, and it not responsible for move to another thread.
- **The dispatcher** and the **context** these are responsible for move to the background thread and move between the threads.
- In Coroutines we have **scopes** that enable control the task (Coroutines Builder)

- So we Control on the task using Coroutines scopes, and we enable the task for working into the background by Suspend key, and we move between threads using Dispatchers and context.

- Note that we can call **Suspend** functions inside Coroutines or inside other suspend.

## 5 – How to build Coroutines?

- **We need some tools:**
    1. At first we need Scope that Coroutines will work inside it such (Global scope, View model Scope, run blocking)
    2. Coroutines Starting key (Launch and Async/Await)
    3. **Declare the Thread type:**
        - **Dispatcher.Io**: we use it when we receive data from server or database.
        - **Dispatcher. Default**: it declare in case of UN defined the thread type, and it use to perform the very heavy tasks.
        - **Dispatcher. Main**: use in very simple tasks.
        - **Dispatcher. Unconfined**.

## 6 – What are the withContext and runBlocking?

- **WithContext:**
    - It is not a Coroutines builder but it's a suspend function that used to move from the background thread to perform one thing into main thread and return to background thread again.
- **runBlocking**:
    - It is a Coroutines builder that enable us to call and use a suspend function inside the main thread.
    - It is use to block the thread for some time until one task perform, say we have two task and we need to perform the first and delay 2 second and then perform the second, then we separated these two asks by run blocking.

## 7 – What are the Async and await and what is the different between them and the launcher?

- These are Coroutines builders, we use them when we need return value from the Coroutines.

- **Different between them and the launcher:**
    - The launcher return Job, so we don't use it when we need to return vale from Coroutines.
    - The Async/Await return deferred value that wraps around the data type single value.
- **Note That:**
    - We start the Coroutines by Async and return the value from the deferred by Await.

## 8 – What is the Job or what is the structure concurrency?

- The ability of control on the Coroutines builder.
- Coroutines builder that return job, we can control it by cancel it, stop it and delay one Coroutines until other completed.
- **Notes:**
    - If we have a parent Coroutines and inside it two child Coroutines, in case of cancel the parent, then the two Childs will also cancel.
    - If one child canceled the parent and the other child will canceled also.
- **Example:**
    - I have the functions A (), B (), C () and I want C () start after the A (), B () finished and I need A (), B () work as parallel.

```
val parent = GlobalScope.launch {
        val child1 = A ()
        val child2 = B ()
        // we use join to delay C () until A (), B () has been finished.
        JoinAll (child1, child2)
        C ()
}
```

## 9 – What is the Channels and what is the different between it and the deferred?

- Using the channel we can connect and transfer list of data between two Coroutines.
- The data not send directly between two Coroutines, but it found Buffer between them this buffer we define its length during define the channel.
- We use two keywords **Send**, **offer** to send data between them.

- **Send:**

  It send the first item and wait for other Coroutines accept and store it, then send the second item, etc.

  So if the buffer is has no capacity, it will receive at least his capacity and reject others.

- **Offer:**

  It send the list as a single unit all list send at the first time, so if I have 4 items into the list and the buffer capacity accept 3 items, so no one item will accept, because the buffer size less than the list size.

- **different between it and the deferred:**
  - The deferred return a single value.
  - Channel return list of data.
- **Example:**
  - val channel = Channel<String>(3)
  - val list = arrayListOf("a", "b", "c")
  - runBlocking {

```
        Launch {
            list.forEach {
                Channel. Send (it)
                Delay (1000)
            }
        }

        Launch {
            channel.forEach {
                Print (it)
            }
        }

    }
```

## All is done ☺