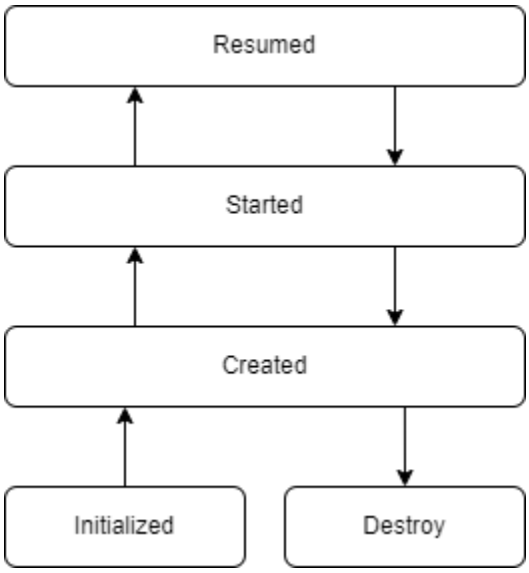
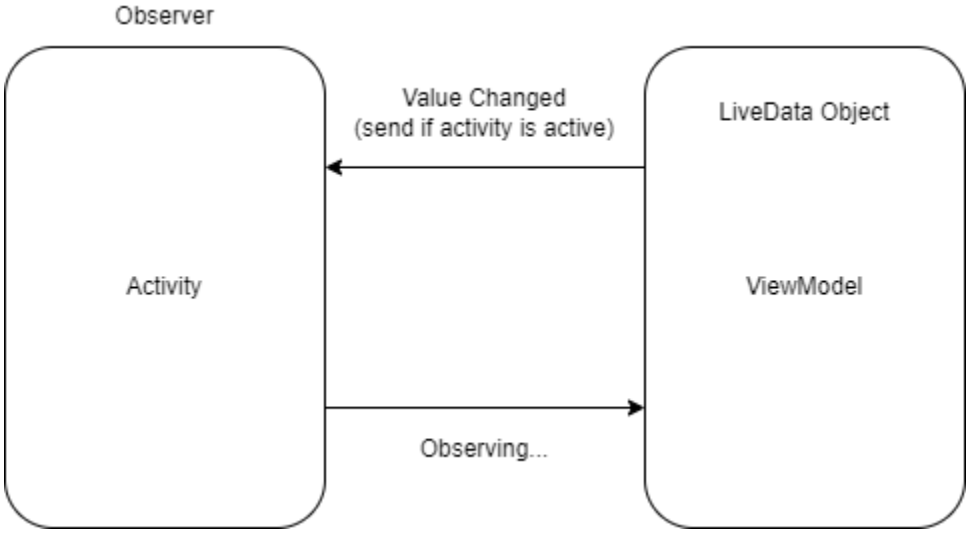
Rz Rasel
(Writting Completed)

LiveData Android আর্কিটেকচারের একটি কম্পোনেন্ট। ViewModel যেমন Android আর্কিটেকচারের একটি কম্পোনেন্ট, ঠিক তেমনি LiveData Android আর্কিটেকচারের একটি কম্পোনেন্ট। এখানে Model View View Model Android আর্কিটেকচারের সাথে LiveData এর ব্যবহার দেখবো।

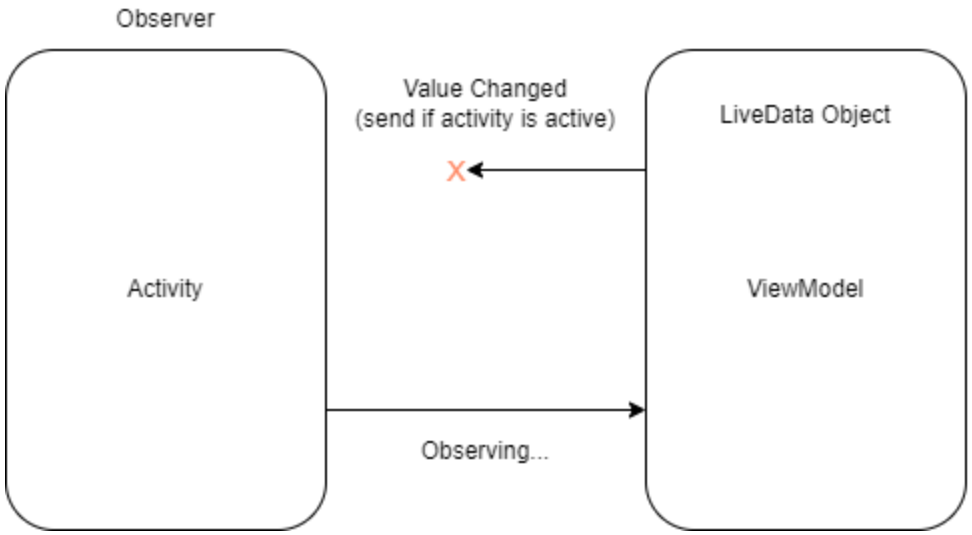
তাহলে LiveData কি, LiveData একটি observable ডেটা হোল্ডার ক্লাস। যা অন্যান্য রেগুলার observable এর মতো নয়। LiveData লাইফসাইকেল সম্পর্কে সচেতন বা লাইফসাইকেলের সাথে সামঞ্জস্য রেখে কাজ করে। অর্থাৎ app এর অন্যান্য কম্পোনেন্ট যেমন activity, fragment বা service র লাইফসাইকেলের সাথে সামঞ্জস্য রেখে কাজ করে।



LiveData শুধুমাত্র app কম্পোনেন্ট একটিভ স্টেটে থাকলেই ডাটা আপডেট করে।



যদি activity paused বা destroyed স্টেটে চলে যায় তাহলে LiveData অবজেক্ট activity কে আপডেট দেয়া বন্ধ করে দেয়। যদি activity, paused বা destroyed স্টেটে চলে যায় তাহলে LiveData অবজেক্ট activity কে আপডেট দেয়া বন্ধ করে দেয়। এবং LiveData, activity আবার এক্টিভ স্টেটে না আসা পর্যন্ত অপেক্ষা করে।



LiveData
Data that is aware of the LifeCycle of it's observer.

সাধারণত LiveData শুধুমাত্র তখনই আপডেট দেয় যখন ডেটা পরিবর্তন হয় এবং observers active থাকে। observer inactive state এ থাকা অবস্থায় কোনো আপডেট হলে, observer inactive state থেকে active state এলে আপডেট চেঞ্জ পায়।

app➜build.gradle

```
implementation "androidx.lifecycle:lifecycle-extensions:2.2.0"
//noinspection LifecycleAnnotationProcessorWithJava8
annotationProcessor "androidx.lifecycle:lifecycle-compiler:2.4.0"
// ViewModel
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.0"

buildFeatures {
    viewBinding true
}
```

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">

    <LinearLayout
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="30dp"
        android:gravity="center"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <TextView
            android:id="@+id/textViewNumber"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="15dp"
            android:text="Hello World!" />
    </LinearLayout>

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

MainActivity class

```
package com.rzrasel.rzrasetutorial
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.rzrasel.rzrasetutorial.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
    private lateinit var viewBinding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        viewBinding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(viewBinding.root)
    }
}
```

viewBinding একটি databinding অবজেক্ট, viewBinding অপশন এনাবল করলে লেআউট ফাইলের নামানুসারে সিস্টেম অটোমেটিক ক্লাসফাইল তৈরি করে। layoutInflater এর মাধ্যমে layout bind করে, ফলে viewBinding অবজেক্টের মাধ্যমে layout এর view এক্সেস করা যায়।

MainActivityViewModel নামের একটি ক্লাস তৈরি করি, যে ক্লাস ViewModel ক্লাসকে এক্সটেন্ড করে।

MainActivityViewModel class

```
package com.rzrasel.rzrasetutorial
import androidx.lifecycle.ViewModel

class MainActivityViewModel : ViewModel() {
}
```

MainActivityViewModel ক্লাস ViewModel ক্লাসকে এক্সটেন্ড করা হয়েছে।

MainActivity class

এখন MainActivity ক্লাসে ViewModel প্রোভাইডারের মাধ্যমে MainActivityViewModel ক্লাসকে ইনটিরগেট করি।

```
package com.rzrasel.rzrasetutorial
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.ViewModelProvider
import com.rzrasel.rzrasetutorial.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
    private lateinit var viewBinding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        viewBinding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(viewBinding.root)
        val viewModel =
ViewModelProvider(this)[MainActivityViewModel::class.java]
    }
}
```

viewModel এ ViewModel Provider এর মাধ্যমে MainActivityViewModel ক্লাসকে assigne করা হয়েছে।

MainActivityViewModel class

```
package com.rzrasel.rzrasetutorial
import android.os.CountDownTimer
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

class MainActivityViewModel : ViewModel() {
    private lateinit var timer: CountDownTimer
    private val _second = MutableLiveData<Int>()
```

```
}
```

এখানে timer হচ্ছে CountDownTimer এর একটি অবজেক্ট। এবং _second একটি MutableLiveData অবজেক্ট। LiveData এবং MutableLiveData এর ভিতর পার্থক্য হচ্ছে MutableLiveData একাধিকবার assigne করা যায়।

```
class MainActivityViewModel : ViewModel() {
    private lateinit var timer: CountDownTimer
    private val _second = MutableLiveData<Int>()

    fun startTimer() {
        timer = object : CountDownTimer(10000, 1000) {
            override fun onTick(p0: Long) {
            }

            override fun onFinish() {
            }
        }
    }
}
```

startTimer ফাংশনের ভিতরে timer CountDownTimer অবজেক্টকে assigne করা হয়েছে। CountDownTimer দুইটা প্যারামিটার নিয়ে থাকে, প্রথমটি কতক্ষন ধরে টাইমারটি চলবে বা delay time, আর দ্বিতীয় প্যারামিটারটি কতক্ষন পরপর চলবে বা interval, আর CountDownTimer এর দুইটি মেথড ওভাররাইড করতে হয় - একটি onTick ও onFinish মেথড। এখানে onTick মেথডটি প্রতিসেকেন্ডে কল হবে, আর onFinish মেথডটি CountDownTimer শেষ হলে কল হবে।

```
timer = object : CountDownTimer(10000, 1000) {
    override fun onTick(p0: Long) {
        val timeLeft = p0 / 1000 // Convert millisecond to second
        _second.value = timeLeft.toInt()
    }

    override fun onFinish() {
    }
}
```

CountDownTimer এর onTick মেথডে timeLeft ভেরিএবলে মিলিসেকেন্ডকে সেকেন্ডে কনভার্ট করে রাখা হয়েছে। এবং পরবর্তীতে তা MutableLiveData তে assigne করা হয়েছে। এখানে onTick মেথডটি প্রতিসেকেন্ডে কল হবে এবং MutableLiveData ও প্রতিসেকেন্ডে আপডেট হবে।

```
timer = object : CountDownTimer(10000, 1000) {
    override fun onTick(p0: Long) {
        val timeLeft = p0 / 1000 // Convert millisecond to second
        _second.value = timeLeft.toInt()
    }
```

```
    override fun onFinish() {
    }
}.start()
```
এবং সবশেষে CountDownTimer এর start ফাংশনটি কল করা হয়েছে।

```
fun stopTimer() {
    timer.cancel()
}
```
MainActivityViewModel ক্লাসে stopTimer নামের আরেকটি মেথড তৈরি করি। এখানে timer কে বন্ধ বা cancel করতে পারবে।

```
fun second(): LiveData<Int> {
    return _second
}
```
`_second` প্রাইভেট `MutableLiveData` হওয়ায় বাইরে থেকে এক্সেস করা যাবে না তাই পাব্লিকলি ডাটা চেঞ্জ অবসার্ভ করার জন্য পাব্লিক `second LiveData` তৈরি করা হলো।

MainActivity class

```
viewModel.startTimer()
viewModel.second().observe(this, Observer {
    viewBinding.textViewNumber.text = it.toString()
})
```

startTimer() মেথডকে কল করে টাইমারকে স্টার্ট করা হলো এবং second LiveData কে অবসার্ভ করবে। যখনি LiveData চেঞ্জ হবে তখনি LiveData অবসার্ভারের মাধ্যমে টেক্সটফিল্ডের ডাটা চেঞ্জ হয়ে যাবে। কোডটি রান করলে দেখা যাবে যখনি LiveData চেঞ্জ হবে তখনি টেক্সট ফিল্ডের ভেলু চেঞ্জ হয়ে যাবে।

MainActivity class

```
class MainActivity : AppCompatActivity() {
    private lateinit var viewBinding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        viewBinding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(viewBinding.root)
                                                val       viewModel       =
ViewModelProvider(this)[MainActivityViewModel::class.java]
        viewModel.startTimer()
        viewModel.second().observe(this, Observer {
            viewBinding.textViewNumber.text = it.toString()
        })
    }
```

```
}
```

MainActivityViewModel class

```
class MainActivityViewModel : ViewModel() {
    private lateinit var timer: CountDownTimer
    private val _second = MutableLiveData<Int>()
    fun second(): LiveData<Int> {
        return _second
    }

    fun startTimer() {
        timer = object : CountDownTimer(10000, 1000) {
            override fun onTick(p0: Long) {
                    val timeLeft = p0 / 1000 // Convert millisecond to
second
                _second.value = timeLeft.toInt()
            }

            override fun onFinish() {
            }
        }.start()
    }

    fun stopTimer() {
        timer.cancel()
    }
}
```

এখন এই এক্সাম্পলের কোড আরো কিছুটা ইমপুরভ করে দেখা যাক।

MainActivityViewModel class

```
class MainActivityTempViewModel : ViewModel() {
    ...
    ...
    var timerValue = MutableLiveData<Long>()
    ...
    ...
    var finished = MutableLiveData<Boolean>()

    ...
    ...
```

```
    fun startTimer() {
        timer = object : CountDownTimer(timerValue.value!!.toLong(),
1000) {
            ...
            ...
            override fun onFinish() {
                finished.value = true
            }
        }.start()
    }

    ...
    ...
}
```

ViewModel তে finished নামে বুলিয়ান টাইপের একটা MutableLiveData অবজেক্ট তৈরি করি। এবং CountDownTimer এর onFinish মেথড থেকে এই বুলিয়ান টাইপের MutableLiveData অবজেক্ট এর ভ্যালু true সেট করি।

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">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="30dp"
        android:gravity="center"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">
```

```
        <TextView
            android:id="@+id/textViewNumber"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="15dp"
            android:text="Hello World!" />

        <EditText
            android:id="@+id/textInputNumber"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:inputType="number" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal">

            <Button
                android:id="@+id/btnStart"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginRight="10dp"
                android:layout_weight="1"
                android:text="Start" />

            <Button
                android:id="@+id/btnStop"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginLeft="10dp"
                android:layout_weight="1"
                android:text="Stop" />
        </LinearLayout>
    </LinearLayout>

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

MainActivity class

```
class MainActivity : AppCompatActivity() {
```

```
    private lateinit var viewBinding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        viewBinding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(viewBinding.root)
        ...
        ...
        viewModel.finished.observe(this, Observer {
            if (it) {
                Toast.makeText(this, "Finished",
Toast.LENGTH_LONG).show()
            }
        })

        viewBinding.btnStart.setOnClickListener(object :
View.OnClickListener{
            override fun onClick(p0: View?) {
                if(viewBinding.textInputNumber.text.isEmpty() ||
viewBinding.textInputNumber.text.length < 4) {
                    Toast.makeText(this@MainActivity, "Invalid number",
Toast.LENGTH_LONG).show()
                } else {
                    viewModel.timerValue.value =
viewBinding.textInputNumber.text.toString().toLong()
                    viewModel.startTimer()
                }
            }
        })

        viewBinding.btnStop.setOnClickListener {
            viewBinding.textViewNumber.text = "0"
            viewModel.stopTimer()
        }
    }
}
```

LiveData UI ডাটা স্টেট নিশ্চিত করে। LiveData observer pattern follow করে। LiveData লাইফ সাইকেলে ডাটা চেঞ্জ হলে observer অবজেক্টকে নোটিফাই করে। আর এই নোটিফিকেশন follow করে এপ্লিকেশনের ডাটা চেঞ্জ হয়। LiveData কোনো memory leak নেই। LiveData র লাইফসাইকেল সম্পূর্ণ হলে, LiveData অবজেক্ট নিজে থেকেই ক্লিন হয় বা মেমোরি ফ্রি করে দেয়। LiveData সব সময় up to date, যদি এমন হয় LiveData কোন কারণে ইনএক্টিভ হয়ে যায়; পরে আবার LiveData একটিভ স্টেটে আসে তাহলে LiveData লেটেস্ট ডাটাই দেবে।

Reference
~~LiveData Explained - Android Architecture Component | Tutorial (Stevdza-San)~~
https://youtu.be/suC0OM5gGAA

https://youtu.be/1Tn7TuHUl4Y

ViewModel Explained - Android Architecture Component | Tutorial

Introduction to MVVM on Android - Tutorial - Learn Android Architecture Patterns
https://youtu.be/_T4zjIEkGOM
Android MVVM Kotlin Tutorial - LiveData + ViewModel (Android Architecture Components)
https://youtu.be/d7UxPYxgBoA

Model View View-Model (MVVM): Getting Started
https://youtu.be/ijXjCtCXcN4
ViewModel Explained - Android Architecture Component | Tutorial
https://youtu.be/orH4K6qBzvE

Rz Rasel