ANDROID ARCHITECTURE COMPONENTS

Dawn of a new era in Android Development

By Ayokunle Paul

Hello Android


17

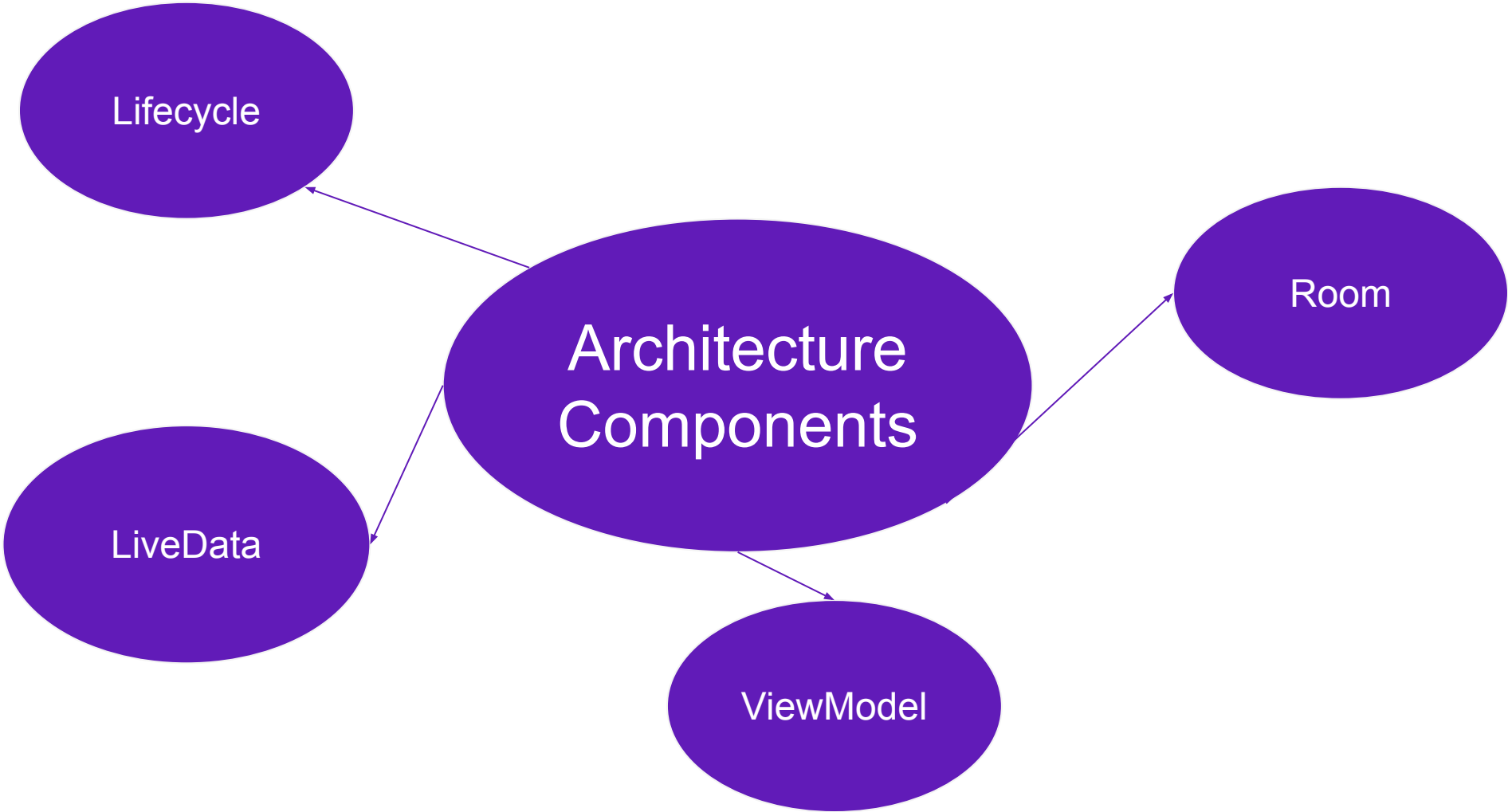
What are Architecture Components???

A new collection of libraries that help you design robust, testable, and maintainable apps.

developer.android.com

HOW?

1. Promotion of the MVVM pattern.
2. Making it easier to conform to the SOC principle.
3. Reducing the risk of resource leaks.
4. Increases apps modularity.


Lifecycle
Architecture Components
Room
LiveData
ViewModel

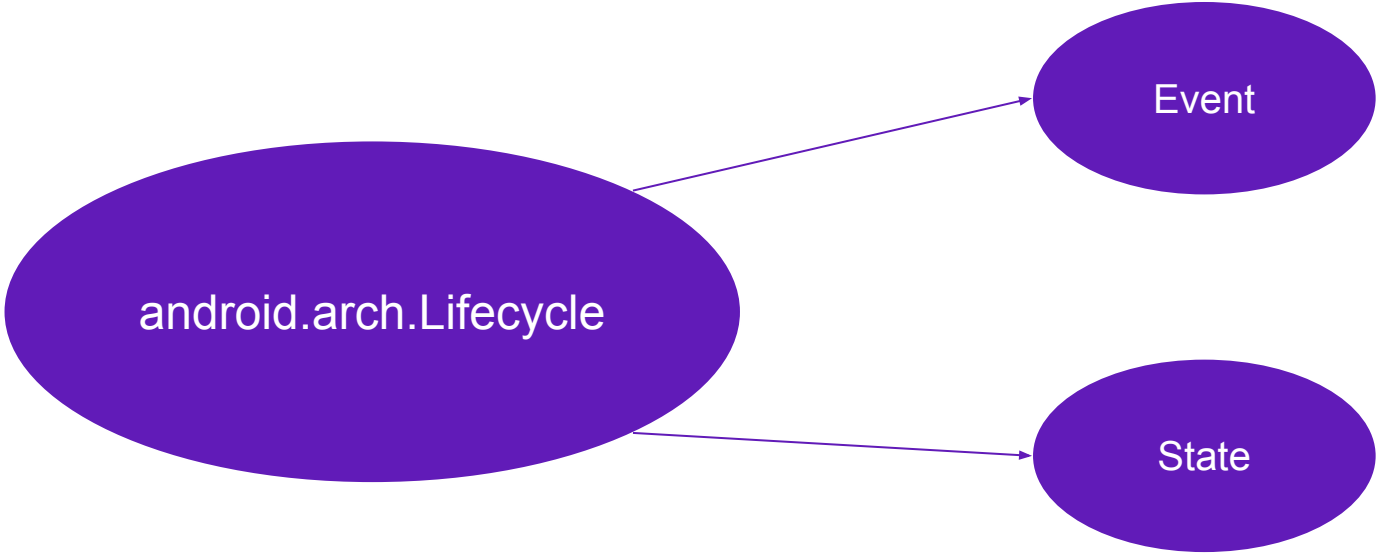
Android OS

System Components

- Activities
- Services
- Broadcast Receivers
- Content Providers

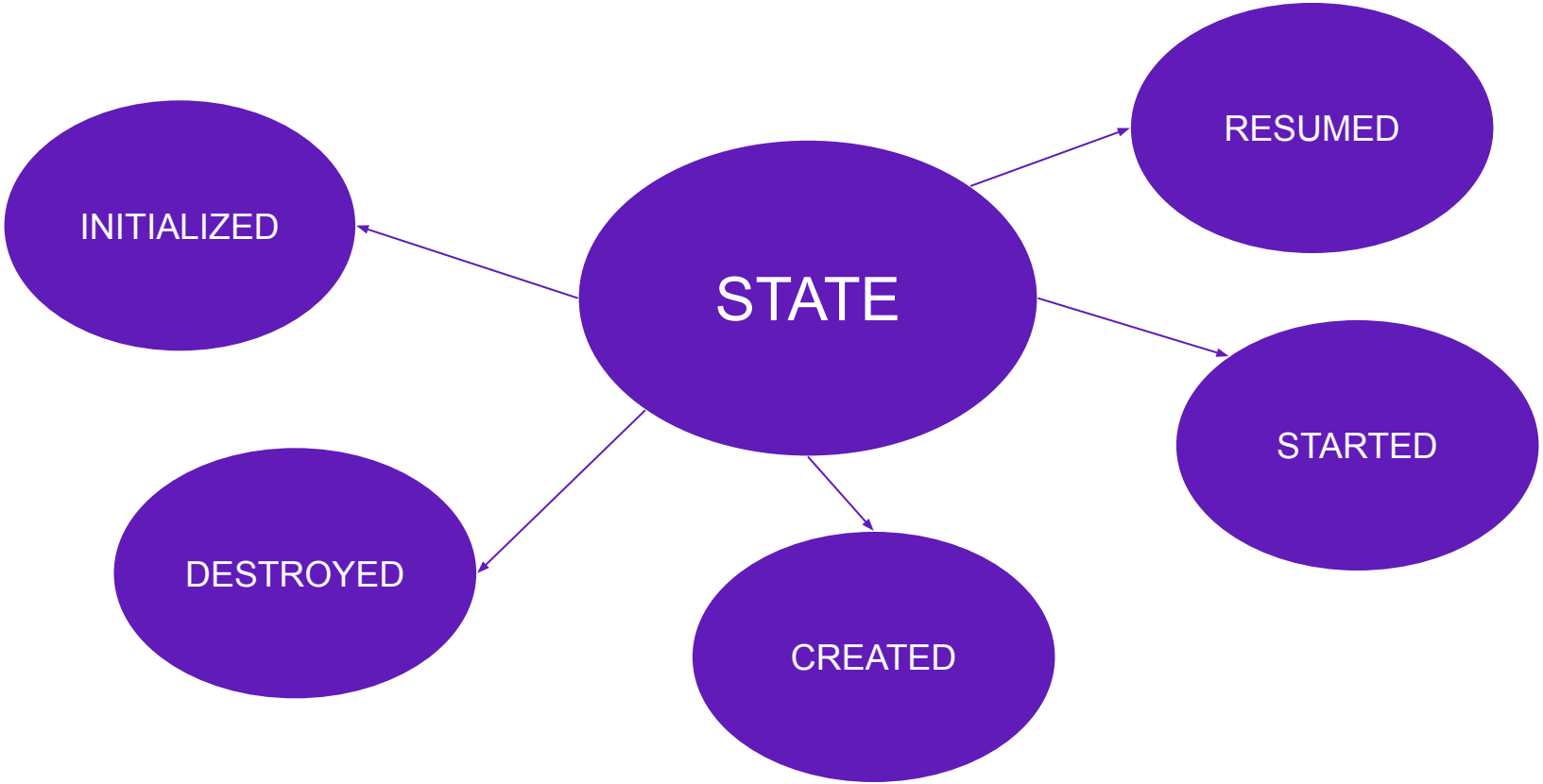
Lifecycle

Lifecycle is a class that holds the information about the lifecycle state of a component (like an activity or a fragment) and allows other objects to observe this state.



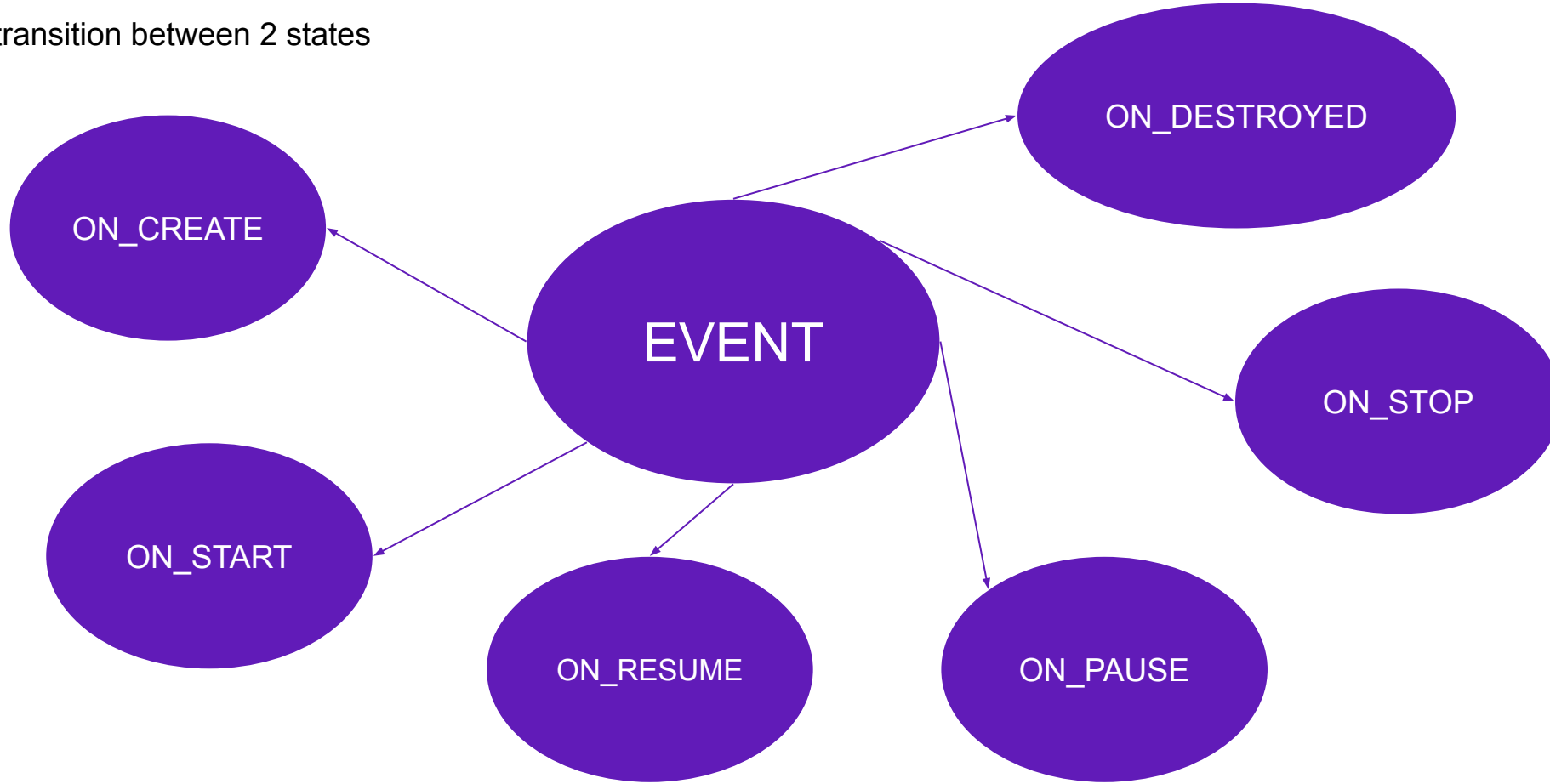
STATE

The current state of the component tracked by the Lifecycle class.



EVENT

The transition between 2 states



IMPLEMENTATION?

```
public class MyObserver implements LifecycleObserver {
    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
    public void connectListener() {
            /.../
}
    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
    public void disconnectListener() {
            /.../
    }
}

myLifecycleOwner.getLifecycle().addObserver(new MyObserver());
```

```
class MyLocationListener implements LifecycleObserver {
    private boolean enabled = false;
    public MyLocationListener(Context context, Lifecycle lifecycle, Callback callback) {
        ...
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_START)
    void start() {
        if (enabled) {
            // connect
        }
    }
```

```
public void enable() {
        enabled = true;
        if (lifecycle.getCurrentState().isAtLeast(STARTED)) {
            // connect if not connected
        }
    }


    @OnLifecycleEvent(Lifecycle.Event.ON_STOP)
    void stop() {
        // disconnect if connected
    }
}
```

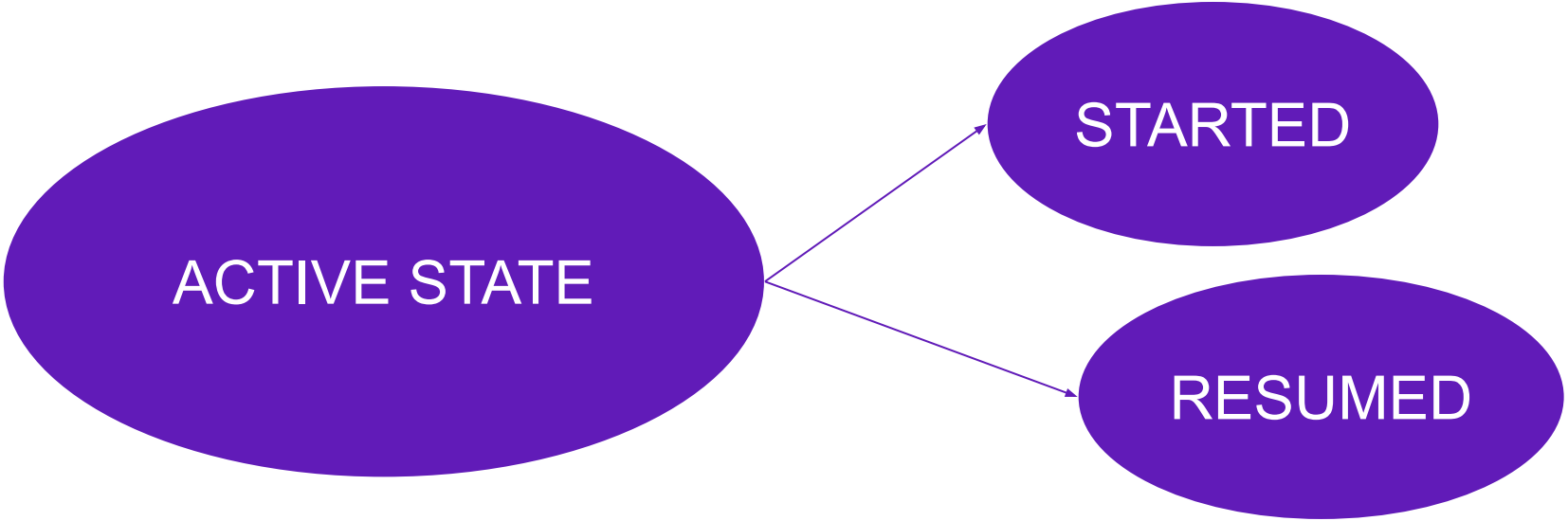
III
(Custom LifecycleOwner)

```java
public class MyActivity extends Activity implements LifecycleOwner {
    private LifecycleRegistry mLifecycleRegistry;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mLifecycleRegistry = new LifecycleRegistry(this);
        mLifecycleRegistry.markState(Lifecycle.State.CREATED);
    }
```

```
    @Override
    public void onStart() {
        super.onStart();
        mLifecycleRegistry.markState(Lifecycle.State.STARTED);
    }
    @NonNull
    @Override
    public Lifecycle getLifecycle() {
        return mLifecycleRegistry;
    }
}
```

LiveData

LiveData is an observable data holder class. Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services. This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.



ADVANTAGES

- ➔ Ensures your UI matches your data state
- ➔ No memory leaks
- ➔ No crashes due to stopped activities
- ➔ No manual lifecycle handling
- ➔ Always up-to-date data
- ➔ Proper configuration changes
- ➔ Sharing resources

IMPLEMENTATION?

```
public class NameViewModel extends ViewModel {

// Create a LiveData with a String

private MutableLiveData<String> mCurrentName;


    public MutableLiveData<String> getCurrentName() {

        if (mCurrentName == null) {

            mCurrentName = new MutableLiveData<String>();

        }

        return mCurrentName;

    }

// Rest of the ViewModel...

}
```

Observing LiveData<T>

```
public class NameActivity extends AppCompatActivity {

    private NameViewModel mModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Other code to setup the activity…
        // Get the ViewModel.
        mModel = ViewModelProviders.of(this).get(NameViewModel.class);
```

```
    // Create the observer which updates the UI.
        final Observer<String> nameObserver = new Observer<String>() {
            @Override
            public void onChanged(@Nullable final String newName) {
                // Update the UI, in this case, a TextView.
                mNameTextView.setText(newName);
            }
        };
        mModel.getCurrentName().observe(this, nameObserver);
    }
}
```

Updating LiveData<T>

```
mButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        String anotherName = "John Doe";
        mModel.getCurrentName().setValue(anotherName);
    }
});
```

ViewModel

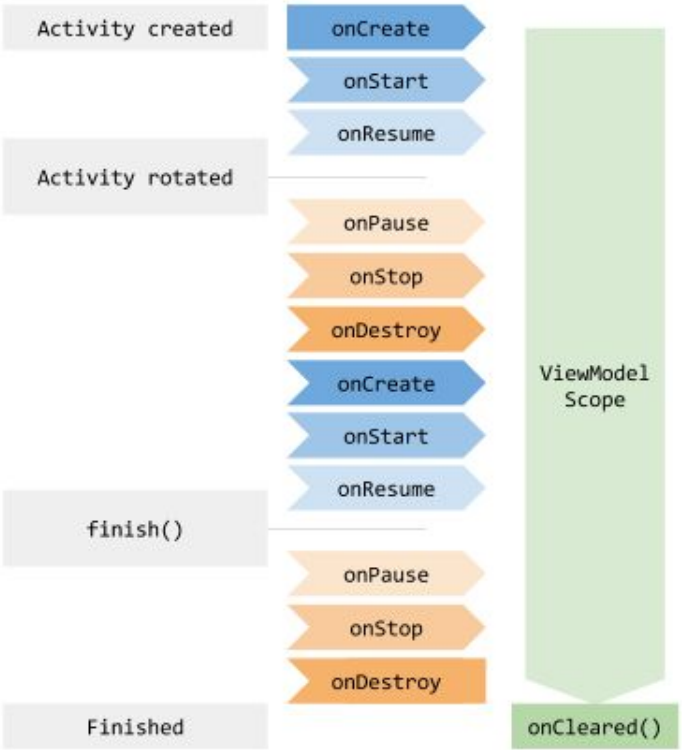
The ViewModel class is designed to store and manage UI-related data in a lifecycle conscious way. The ViewModel class allows data to survive configuration changes such as screen rotations.

IMPLEMENTATION?

```
public class MyViewModel extends ViewModel {
    private MutableLiveData<List<User>> users;
    public LiveData<List<User>> getUsers() {
        if (users == null) {
            users = new MutableLiveData<List<Users>>();
            loadUsers();
        }
        return users;
    }
```

```
    private void loadUsers() {

        // Do an asyncronous operation to fetch users.

    }

}
```

```
public class MyActivity extends AppCompatActivity {

    public void onCreate(Bundle savedInstanceState) {

        // Create a ViewModel the first time the system calls an activity's onCreate() method.

        // Re-created activities receive the same MyViewModel instance created by the first
activity.


        MyViewModel model = ViewModelProviders.of(this).get(MyViewModel.class);

        model.getUsers().observe(this, users -> {

            // update UI

        });

    }

}
```



ViewModel Lifecycle

You can also share data between Fragments

```
public class SharedViewModel extends ViewModel {
    private final MutableLiveData<Item> selected = new MutableLiveData<Item>();

    public void select(Item item) {
        selected.setValue(item);
    }

    public LiveData<Item> getSelected() {
        return selected;
    }
}
```

```
public class MasterFragment extends Fragment {

    private SharedViewModel model;

    public void onCreate(Bundle
savedInstanceState) {

        super.onCreate(savedInstanceState);

        model =
ViewModelProviders.of(getActivity()).get(Shared
ViewModel.class);

        itemSelector.setOnClickListener(item -> {

            model.select(item);

        });

    }

}
```

```
public class DetailFragment extends Fragment {

    public void onCreate(Bundle
savedInstanceState) {

        super.onCreate(savedInstanceState);

        SharedViewModel model =
ViewModelProviders.of(getActivity()).get(Shared
ViewModel.class);

        model.getSelected().observe(this, { item
->

            // Update the UI.

        });

    }

}
```

The MVVM pattern

Android architecture components promotes the MVVM pattern.

- Model
- View
- ViewModel

“Separation of Concern is the design principle that for separating a computer program into each sections, such that each section addresses a separate concern.”

- Wikipedia

Remember, your apps components shouldn't do more than they should.

Thanks!

Contact us:

Obafemi Awolowo University, Ile-Ife

paul_eipeks@gmail.com