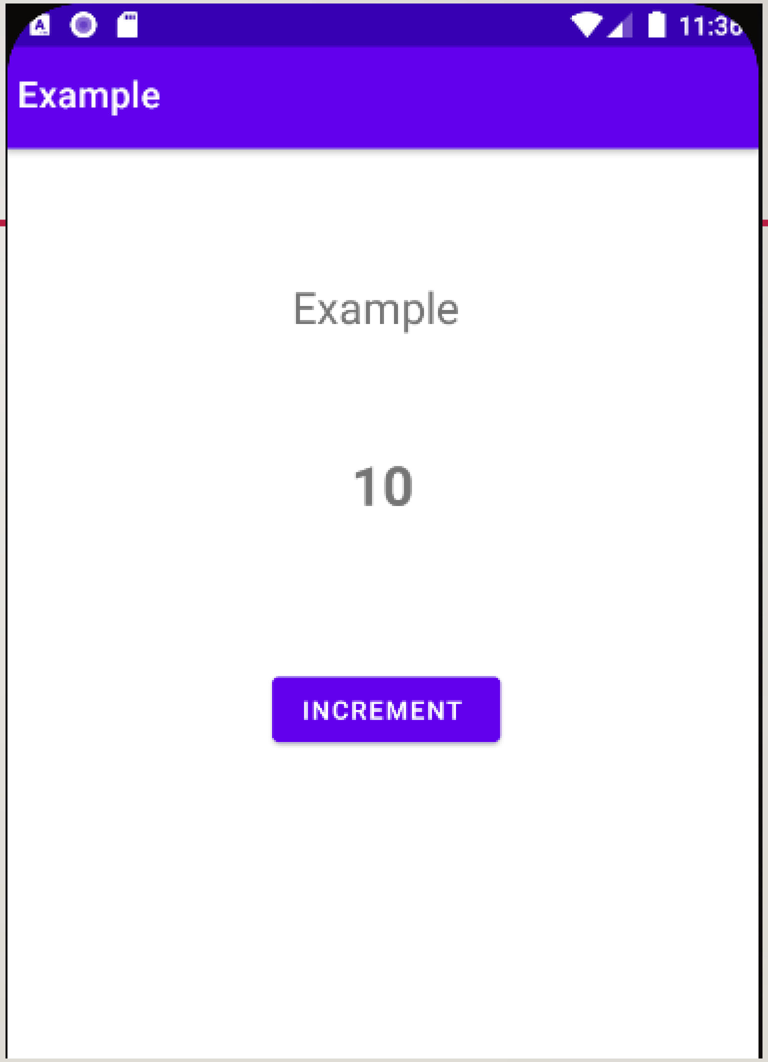
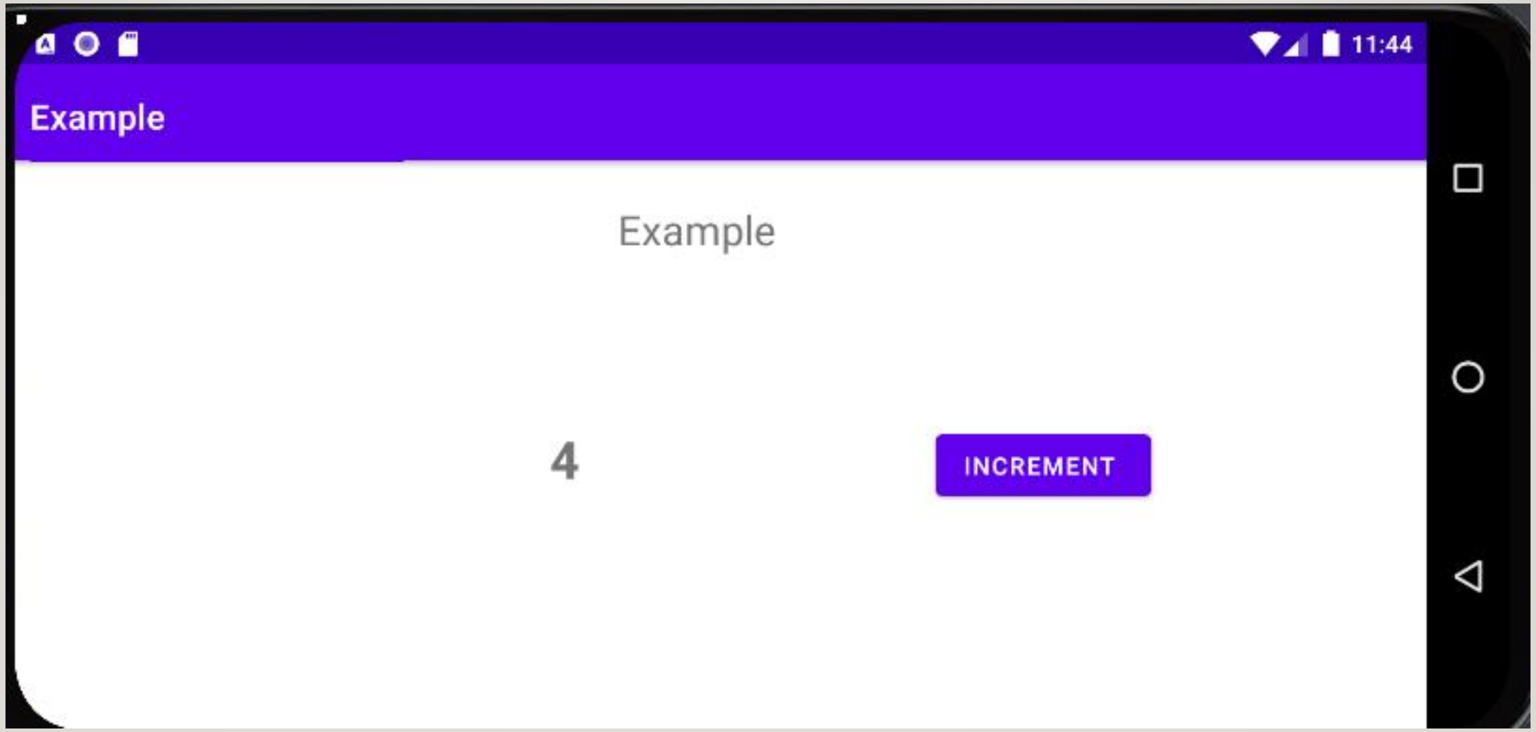
TOPICS

- Creating Landscape Layout
- View Model
- Live Data
- Alert Dialog

EXAMPLE



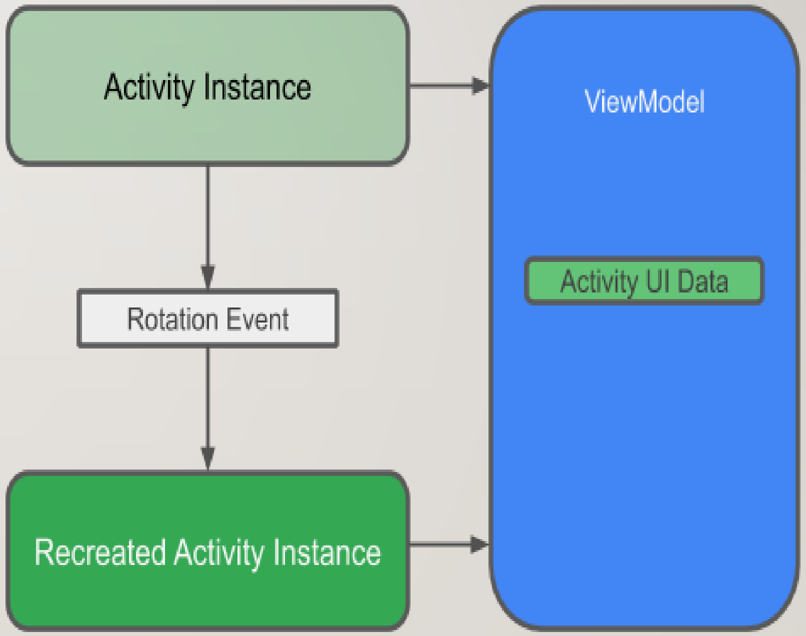
CREATING LANDSCAPE LAYOUT



VIEW MODEL

VIEW MODEL

- The ViewModel's role is to provide data to the UI and survive configuration changes.
- You can also use a ViewModel to share data between fragments.
- The ViewModel is part of the lifecycle library.



VIEW MODEL CONT.

- A ViewModel holds the app's UI data in a lifecycle-conscious way that survives configuration changes.
- Separating the app's UI data from your Activity / Fragment classes improve the code (implement the single responsibility principle)
  - activities and fragments are responsible for drawing data to the screen, while
  - ViewModel can take care of holding and processing all the data needed for the UI.
- You should use LiveData for changeable data that the UI will use.

EXAMPLE: :- VIEW MODEL

```java
public class MyViewModel extends ViewModel {

    private int value  = 0;
    public int getValue() {
        return value;
    }
    public void increment(){
        value += 1;
    }
    public void setValue(int value) {
        this.value = value;
    }
}
```

VIEW MODEL CONT…

- a viewmodel class is created by extending ViewModel class or AndroidViewModel class.
- If you need the application context (which has a lifecycle that lives as long as the application does), use AndroidViewModel.

USING VIEW MODEL

```
// Main Activity
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
   // creating viewModel object
   viewModel = new ViewModelProvider(this,
            getDefaultViewModelProviderFactory()).get(MyViewModel.class);

    tv = findViewById(R.id.valueTv);
    tv.setText(""+viewModel.getValue());
 }
```

CONNECT WITH THE DATA FROM ACTIVITY

- Use ViewModelProvider to associate your ViewModel with your Activity.
- When your Activity first starts, the ViewModelProvider will create the ViewModel.
- When the activity is destroyed, for example through a configuration change, the ViewModel persists.
- When the activity is re-created, the ViewModelProviders return the existing ViewModel.

-
  ```
  viewModel = new ViewModelProvider(this,
  getDefaultViewModelProviderFactory()).get(MyViewModel.class);
  tv = findViewById(R.id.valueTv);
      tv.setText(""+viewModel.getValue());
  ```

```java
// increment the value when button is clicked
public void incrementValue(View view) {

 viewModel.increment();  // increment
  tv.setText(String.valueOf(viewModel.getValue())); // Update UI

}
```

- USING LiveData class

LIVEDATA CLASS

- 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.

- LiveData considers an observer, which is represented by the Observer class, to be in an active state if its lifecycle is in the STARTED or RESUMED state.
- LiveData only notifies active observers about updates.

LIVE DATA CLASS CONT…

- If you want to update data stored within LiveData, you must use MutableLiveData instead of LiveData.
- The MutableLiveData class has two public methods that allow you to set the value of a LiveData object, setValue(T) and postValue(T).
- Usually, MutableLiveData is used within the ViewModel, and then the ViewModel only exposes immutable LiveData objects to the observers.

```
public class MyViewModel extends ViewModel {

    private MutableLiveData<Integer> number = new MutableLiveData<Integer>(0);

    public LiveData<Integer> getNumber(){
        return number;
    }
    public void incrementNumber(){
        Integer vale = getNumber().getValue()+1;
        number.setValue(vale);

    }
}
```

```
// Main Activity
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
   // viewModel = ViewModelProviders.of(this).get(MyViewModel.class);
   viewModel = new ViewModelProvider(this,
getDefaultViewModelProviderFactory()).get(MyViewModel.class);
    //value =
    tv = findViewById(R.id.valueTv);

    viewModel.getNumber().observe(this, n ->{
        tv.setText(String.valueOf(n));
    });

 }
```

```
// increment the value when button is clicked
public void incrementValue(View view) {

// increment number and UI will be updated
 viewModel.incrementNumber();



}
```

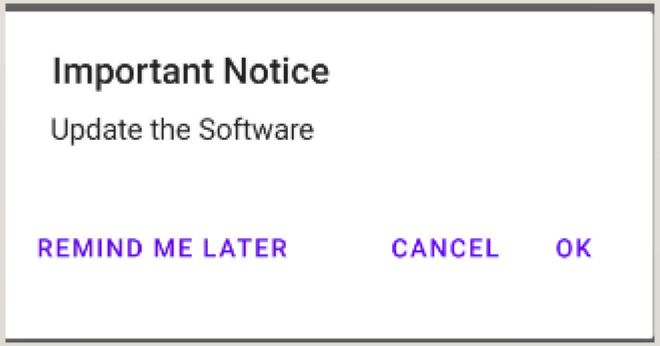
LIVEDATA - BENEFITS

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

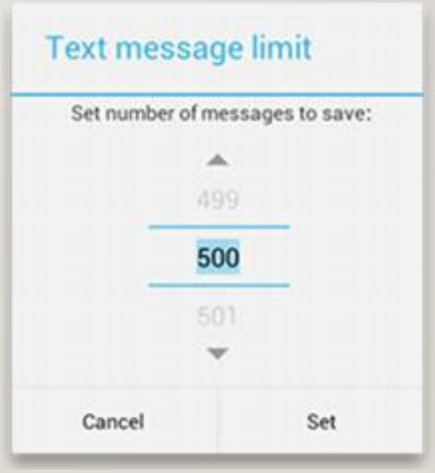
DIALOGS

- A dialog is a small window that prompts the user to make a decision or enter additional information.
- A dialog does not fill the screen and is normally used for events that require users to take an action before they can proceed.
- Can have up

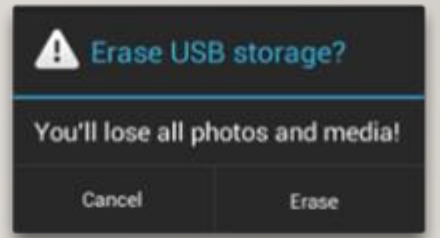
to three buttons


Important Notice

Update the Software


Text message limit

Set number of messages to save:


Erase USB storage?

You'll lose all photos and media!

EXAMPLE :- ALERT DIALOG

```java
AlertDialog.Builder builder  = new AlertDialog.Builder(this);

builder.setTitle(" Important Notice  ")
        .setMessage(" Update the Software")
        .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Toast.makeText(MainActivity.this, "ok", Toast.LENGTH_LONG).show();
            }
        })
        .setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Toast.makeText(MainActivity.this, "Cancel", Toast.LENGTH_LONG).show();
            }
        });
 AlertDialog dialog = builder.create();
 dialog.show();
```

DIALOGS

- You can also use the following
  - setIcon() – sets the icon
  - setPositiveButton()
  - setNeutralButton() (i.e. remind me later…)
  - setNegativeButton() (use to cancel the action)
  - setItems() – If you want to add a selectable list of items

EXERCISE

- Add TextView with Initial text **Hello world.**
- Add button that has onClick method resetText.
- If the button is clicked show an alert dialog to confirm or cancel the action.
- Confirm mean reset text to **Hello Android.**
- Cancel mean do nothing.

REFERENCES

- https://developer.android.com/topic/libraries/architecture/viewmodel
- https://developer.android.com/guide/topics/ui/dialogs
- https://developer.android.com/topic/libraries/architecture/livedata