Jetpack Compose

What is compose?

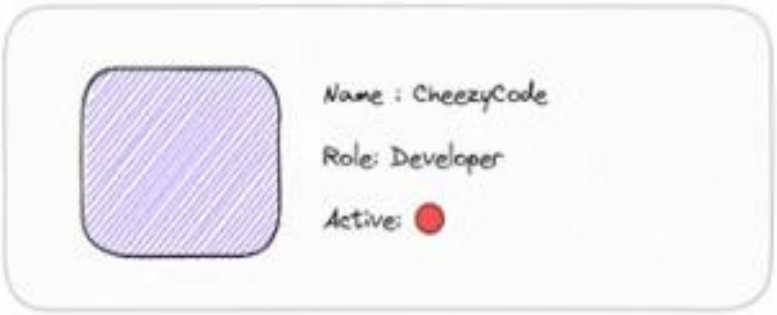
- Android Native UI Toolkit written in Kotlin.
- Not part of the framework, it is a Jetpack Library.
- Basic Building Block – Composable (Kotlin Function with @Composable)

Imperative vs Declarative:

Imperative is How? -----> xml

Declarative is What? -----> jetpack compose

```
{
    "isActive" : true,
    "imageURL" : "https://www.cheezycode.com",
    "name": "CheezyCode",
    "role": "Developer"
}
```



Imperative:

Declarative:

```
@Composable
fun Profile(obj: Profile) {
    Row {
        Image(
            painter = painterResource(id = R.drawable.ic_launcher_background),
            contentDescription = "Logo"
        )
        Column {
            Text(text = "Name : ${obj.name}")
            Text(text = "Role : ${obj.role}")
            if (obj.isActive) {
                Row {
                    Text(text = "Active")
                    Checkbox(checked = true, onCheckedChange = {})
                }
            }
        }
    }
}
```

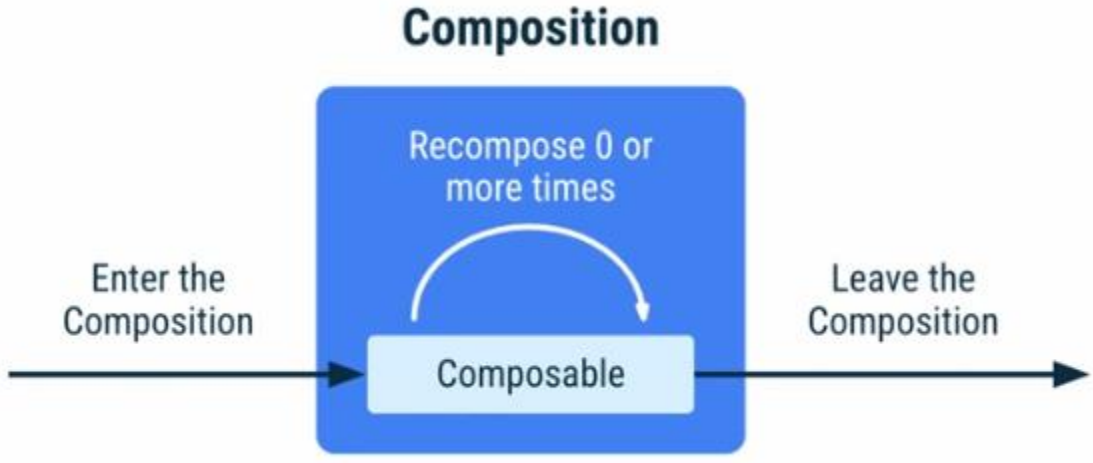
fun(data) = UI

Inheritance Tree:

- All the views inherit from View Class.
- For large codebase, Inheritance becomes complex.
- Composition over Inheritance.

Recomposition:

In simple words, whenever your state changes, it will recreate the UI.



 # Program1: Use compose in a xml file.

build.gradle.kts:

```
android {
   buildFeatures {
      compose = true
   }
   composeOptions {
      kotlinCompilerExtensionVersion = "1.4.3"
   }
}
dependencies {
   implementation(platform("androidx. compose:compose-bom:2023.03.00"))
   implementation("androidx.compose.ui:ui")
   implementation("androidx.compose.ui:ui-graphics")
   implementation("androidx.compose.ui:ui-tooling-preview")
   implementation("androidx.compose.material3:material3")
   androidTestImplementation(platform("androidx.compose:compose-bom:2023.03.00"))
   androidTestImplementation("androidx.compose.ui:ui-test-junit4")
   debugImplementation("androidx.compose.ui:ui-tooling")
   debugImplementation("androidx.compose.ui:ui-test-manifest")
}
```

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">

   <androidx.compose.ui.platform.ComposeView
      android:id="@+id/compose_layout"
      android:layout_width="0dp"
      android:layout_height="0dp"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintHorizontal_bias="0.0"
      app:layout_constraintStart_toStartOf="parent"
```

```
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.0" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

MainActivity.kt:

```
package com.example.viewswithcompose
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.platform.ComposeView

class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      val composeLayout = findViewById<ComposeView>(R.id.compose_layout)
      composeLayout.setContent {
         SayCheeze("Jony")
      }
   }
}

@Composable
fun SayCheeze( name:String){
   Text("Hello! $name")
}
```

 # Program2: Use of Preview in a compose.

build.gradle.kts:

```
android {
   buildFeatures {
      compose = true
   }
   composeOptions {
      kotlinCompilerExtensionVersion = "1.4.3"
   }
}
```

```
dependencies {
   implementation(platform("androidx. compose:compose-bom:2023.03.00"))
   implementation("androidx.compose.ui:ui")
   implementation("androidx.compose.ui:ui-graphics")
   implementation("androidx.compose.ui:ui-tooling-preview")
   implementation("androidx.compose.material3:material3")
   androidTestImplementation(platform("androidx.compose:compose-bom:2023.03.00"))
   androidTestImplementation("androidx.compose.ui:ui-test-junit4")
   debugImplementation("androidx.compose.ui:ui-tooling")
   debugImplementation("androidx.compose.ui:ui-test-manifest")
}
```

MainActivity.kt:

```
package com.example.jetpackcompose1

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.tooling.preview.Preview

class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
         Text("Hello! CheezeCode")
      }
   }
}

@Preview(showBackground = true, name = "msg1")
@Composable
fun SayCheeze1(name: String = "Rony") {
   Text("Hello! $name")
}

@Preview(showBackground = true, name = "msg2")
@Composable
fun SayCheeze2(name: String = "Jony") {
   Text("Hello! $name")
}
```

```
@Preview(showBackground = true, showSystemUi = true)
@Composable
fun SayCheeze3(name: String = "Rony") {
   Text("Hello! $name")
}

@Preview(showBackground = true, widthDp = 300, heightDp = 300)
@Composable
fun SayCheeze4(name: String = "Rony") {
   Text("Hello! $name")
}

@Composable
fun SayCheeze5(name: String) {
   Text("Hello! $name")
}

@Preview(showBackground = true, widthDp = 300, heightDp = 300)
@Composable
private fun SayCheeze6() {
   SayCheeze5(name = "Rony")
}
```

 # Program3: Use of basic Composables.

MainActivity.kt:

```
package com.example.basiccomposables
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.graphics.Color
```

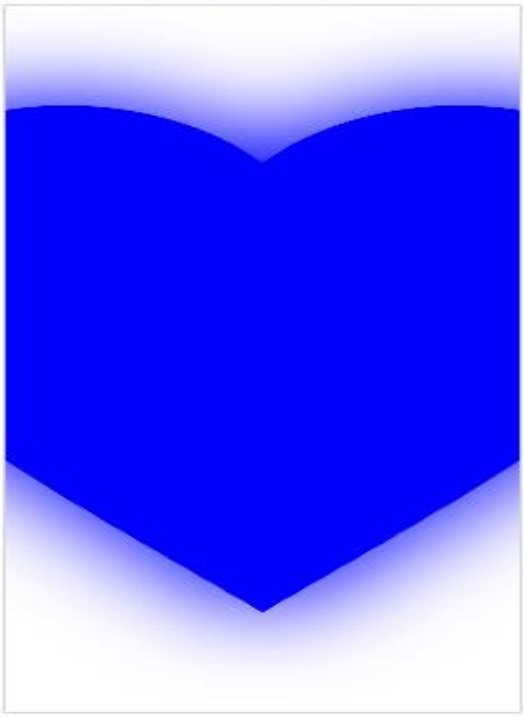
```
import androidx.compose.ui.graphics.ColorFilter
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.sp


class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
         UseOfTextFieldComposable2()
      }
   }
}


@Preview
@Composable
fun UseOfTextComposable() {
   Text(
      text = "Hello! World",
      fontSize = 36.sp,
      fontStyle = FontStyle.Italic,
      fontWeight = FontWeight.ExtraBold,
      textAlign = TextAlign.Center,
      color = Color.Red
   )
}


@Preview
@Composable
fun UseOfImageComposable(){
   Image(
      painter = painterResource(id = R.drawable.heart),
      contentDescription = "Dummy image.",
      colorFilter = ColorFilter.tint(Color.Blue),
      contentScale = ContentScale.Crop
   )
}
```

```
@Preview
@Composable
fun UseOfButtonComposable(){
 Button(
    onClick = { },
    enabled = true,
    colors = ButtonDefaults.buttonColors(
       contentColor = Color.White,
       containerColor = Color.Black
    )
 ) {
    Text(
       text = "Hello",
       fontSize =25.sp
    )
    Image(
       painter = painterResource(id = R.drawable.heart),
       contentDescription = "Dummy image.",
   )
  }
}

@OptIn(ExperimentalMaterial3Api::class)
@Preview
@Composable
fun UseOfTextFieldComposable1(){
   TextField(
      value = "Hello! World",
      onValueChange = {
         Log.d("My World",it)
      },
      label = { Text(text = "Enter message")},
   )
}

@OptIn(ExperimentalMaterial3Api::class)
@Preview
@Composable
fun UseOfTextFieldComposable2(){
   val state = remember{mutableStateOf("")}
   TextField(
      value = state.value,
      onValueChange = {
         state.value = it
      },
      label = { Text(text = "Enter message")},
   )
}
```

UseOfTextComposable

Hello! World

UseOfImageComposable



UseOfButtonComposable

UseOfTextFieldComposable1

Enter message
Hello! World

UseOfTextFieldComposable2

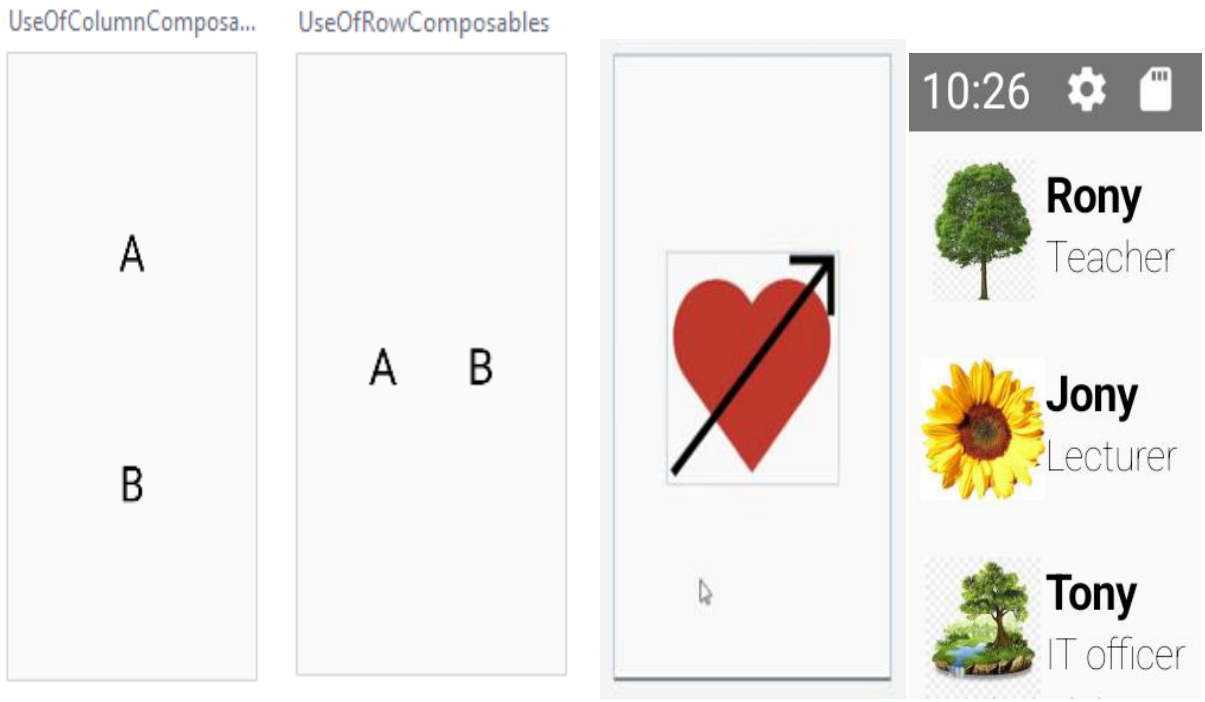
Enter message

 # Program4: Use of Layout Composables.

MainActivity.kt:

```
package com.example.useoflayoutcomposables

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
         Column {
            ListViewItem(R.drawable.tree,"Rony","Teacher")
            ListViewItem(R.drawable.flower,"Jony","Lecturer")
            ListViewItem(R.drawable.nature,"Tony","IT officer")
         }
      }
   }
}
```

```
@Preview(showBackground = true, widthDp = 150, heightDp = 300)
@Composable
fun UseOfColumnComposables() {
   Column(
      verticalArrangement = Arrangement.SpaceEvenly,
      horizontalAlignment = Alignment.CenterHorizontally
   ){
      Text(text = "A", fontSize = 24.sp)
      Text(text = "B", fontSize = 24.sp)
   }
}

@Preview(showBackground = true, widthDp = 150, heightDp = 300)
@Composable
fun UseOfRowComposables() {
   Row(
      horizontalArrangement = Arrangement.SpaceEvenly,
      verticalAlignment = Alignment.CenterVertically
   ){
      Text(text = "A", fontSize = 24.sp)
      Text(text = "B", fontSize = 24.sp)
   }
}

@Preview(showBackground = true, widthDp = 150, heightDp = 300)
@Composable
fun UseOfBoxComposables() {
   Box(
      contentAlignment = Alignment.Center
   ){
     Image(painter = painterResource(id = R.drawable.heart), contentDescription = "")
     Image(painter = painterResource(id = R.drawable.arrow), contentDescription = "")
   }
}

@Composable
fun ListViewItem(imgId:Int,name:String,occupation:String) {
   Row(Modifier.padding(8.dp)){
      Image(painter = painterResource(id = imgId), contentDescription = "",Modifier.size(40.dp))
      Column {
         Text(text = name, fontWeight = FontWeight.Bold)
         Text(text = occupation, fontWeight = FontWeight.Thin, fontSize = 12.sp)
      }
   }
}
```

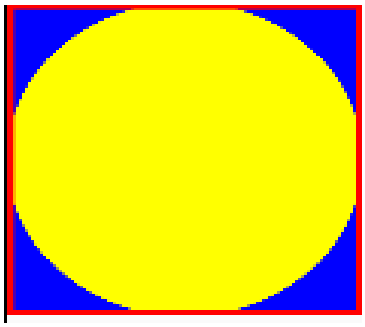


 # Program5: Use of Modifier Composables.

MainActivity.kt:

```
package com.example.modifiercompose

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
```

```
class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
         //ModifierCompose1()
         ModifierCompose2()
      }
   }
}

@Preview(showBackground = true, widthDp = 300, heightDp = 500)
@Composable
fun ModifierCompose1() {
   Text(
      text = "Hello",
      color = Color.White,
      modifier = Modifier.background(Color.Blue)
         .size(200.dp)
         .border(4.dp,Color.Red)
         .clip(CircleShape)
         .background(Color.Yellow)
         .clickable {  }
   )
}

@Preview
@Composable
fun ModifierCompose2(){
   Image(
      painter = painterResource(id = R.drawable.nature),
      contentDescription = "Nature image.",
      contentScale = ContentScale.Crop,
      modifier = Modifier.size(80.dp)
         .clip(CircleShape)
         .border(2.dp, Color.LightGray, CircleShape)
   )
}
```



 # Program6: RecyclerView using List & LazyColumn.

MainActivity.kt:

```
package com.example.recyclerviewcompose

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent

class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
         //PreviewItem1()
         PreviewItem2()
      }
   }
}
```

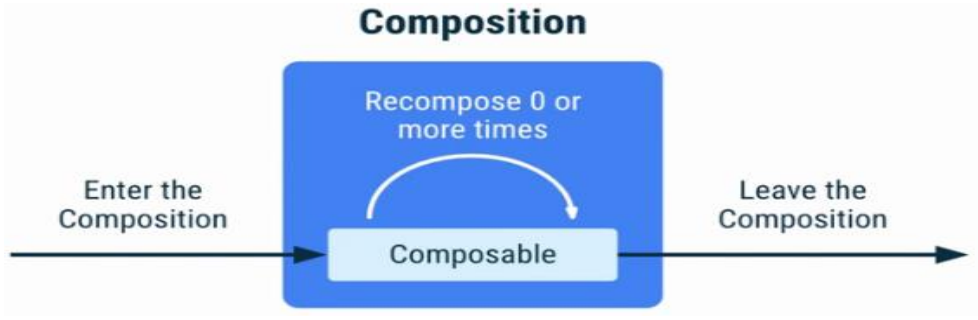
Screen.kt:

```
package com.example.recyclerviewcompose

import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
```

```
@Preview(heightDp = 400, widthDp = 300)
@Composable
fun PreviewItem1(){
   //List load all items whatever scroll or not.
   Column(modifier =  Modifier.verticalScroll(rememberScrollState())) {
      getCategoryList().map { item ->
         BlogCategory(img=item.img,title=item.title,subtitle=item.subtitle)
      }
   }
}

@Preview(heightDp = 400, widthDp = 300)
@Composable
fun PreviewItem2(){
   //LazyColumn load needed items when scroll.
   LazyColumn(content = {
      items(getCategoryList()){ item ->
         BlogCategory(img=item.img,title=item.title,subtitle=item.subtitle)
      }
   })
}

data class Category (val img:Int,val title: String, val subtitle: String)

fun getCategoryList(): MutableList<Category>{
   val list = mutableListOf<Category>()
   list.add(Category(img=R.drawable.men,title="Android",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="AI",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="ML",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="DL",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="NLP",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="CV",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="MT",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="Android",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="AI",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="ML",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="DL",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="NLP",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="CV",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="MT",subtitle="Learn different languages"))
   list.add(Category(img=R.drawable.men,title="Android",subtitle="Learn different languages"))
   return list
}
```

```
@Composable
fun BlogCategory(img:Int,title:String,subtitle:String){
   Card(elevation = CardDefaults.cardElevation(8.dp), modifier = Modifier.padding(8.dp)) {
      Row(verticalAlignment = Alignment.CenterVertically,modifier = Modifier.padding(8.dp)) {
         Image(painter = painterResource(id = img), contentDescription = "", modifier = Modifier
            .size(48.dp)
            .padding(8.dp)
            .weight(.2f))
         ItemDescription(title, subtitle,Modifier.weight(.8f))
      }
   }
}

@Composable
fun ItemDescription(title: String, subtitle: String,modifier: Modifier) {
   Column(modifier = modifier) {
      Text(
         text = title,
         fontWeight = FontWeight.Bold,
         style = MaterialTheme.typography.titleMedium
      )
      Text(
         text = subtitle,
         fontWeight = FontWeight.Thin,
         style = MaterialTheme.typography.bodySmall
      )
   }
}
```

Composition:

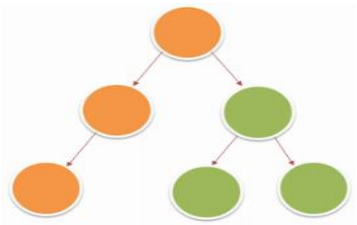


1. Initial Composition
2. Recomposition

State Object:

```
@Composable
fun TextInput() {
    val state = remember { mutableStateOf("") }
    TextField(
        value = state.value,
        onValueChange = {
            state.value = it
        },
        label = { Text(text = "Enter Message") },
    )
}
```

Mutable State is Observable State that works Like a Live Data.
When this state change,then only change this portion that are affected by this state.



Recomposition:

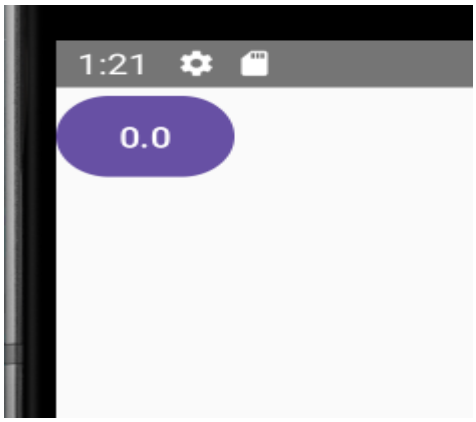
- In simple words – Whenever your state changes, it will recreate the UI.

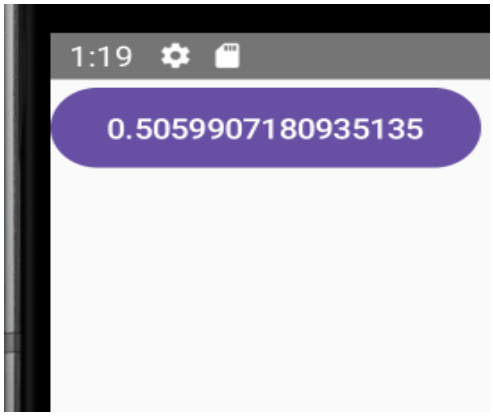
```
@Composable
fun ButtonRow() {
    MyFancyNavigation {
        StartScreen()
        MiddleScreen()
        EndScreen()
    }
}
```

- Composable functions can execute in any order.
- Composable functions can run in parallel
- Recomposition skips as many composable functions and lambdas as possible.
- Recomposition is optimistic and may be canceled.
- A composable function might be run quite frequently, as often as every frame of an animation

MainActivity.kt:

```
package com.example.jetpackrecomposition

import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Recomposable()
        }
    }
}

@Composable
fun Recomposable(){
    val state = remember{mutableStateOf(0.0)}
    Log.d("Tagged","Logged during initial composition")
    Button(onClick = {state.value = Math.random()}) {
        Log.d("Tagged","Logged during both composition & recomposition")
        Text(text = state.value.toString())
    }
}
```





 # Program7: Use of State variable .

MainActivity.kt:

```
package com.example.jetpackcomposestate
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import com.example.jetpackcomposestate.ui.theme.JetpackComposeStateTheme

class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
         NotificationScreen()
      }
   }
}
```

StateExample.kt:

```
package com.example.jetpackcomposestate
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.outlined.Favorite
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.MutableState
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
```

```
@Preview
@Composable
fun NotificationScreen(){
   var count: MutableState<Int> = rememberSaveable{mutableStateOf(0)}
   Column(verticalArrangement=Arrangement.Center,
         horizontalAlignment=Alignment.CenterHorizontally,
         modifier=Modifier.fillMaxSize(1f)){
      NotificationCounter(count.value) { count.value++ }
      MessageBar(count.value)
   }
}


@Composable
fun NotificationCounter(count: Int, increment: () -> Int) {
   Column(verticalArrangement = Arrangement.Center) {
      Text(text = "You have sent $count notification")
      Button(onClick = { increment() }) {
         Text(text = "Send notification")
      }
   }
}


@Composable
fun MessageBar(count: Int) {
   Card( elevation = CardDefaults.cardElevation(4.dp)){
      Row(modifier= Modifier.padding(8.dp),
         verticalAlignment = Alignment.CenterVertically){
         Image(imageVector = Icons.Outlined.Favorite,
              contentDescription ="",
              modifier= Modifier.padding(8.dp))
         Text(text = "Messages sent so far $count")
      }
   }
}
```

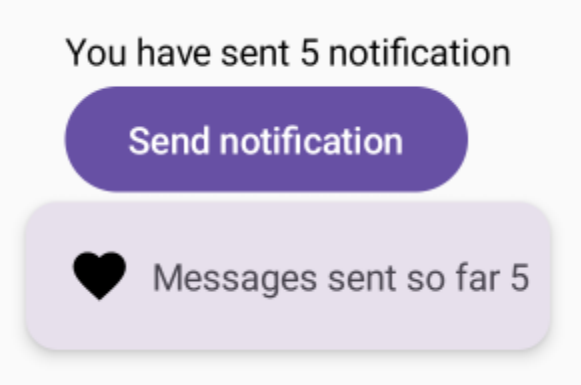
/*
var count = 0
Normal variable not inform compose to update the UI.
So that we need state variable which is obserable.
State variable inform compose to update the UI by recalling function.

var count: MutableState<Int> = remember{mutableStateOf(0)}
Data stored in composition.
So that when we rotate mobile,
we lost data because new activity is created.

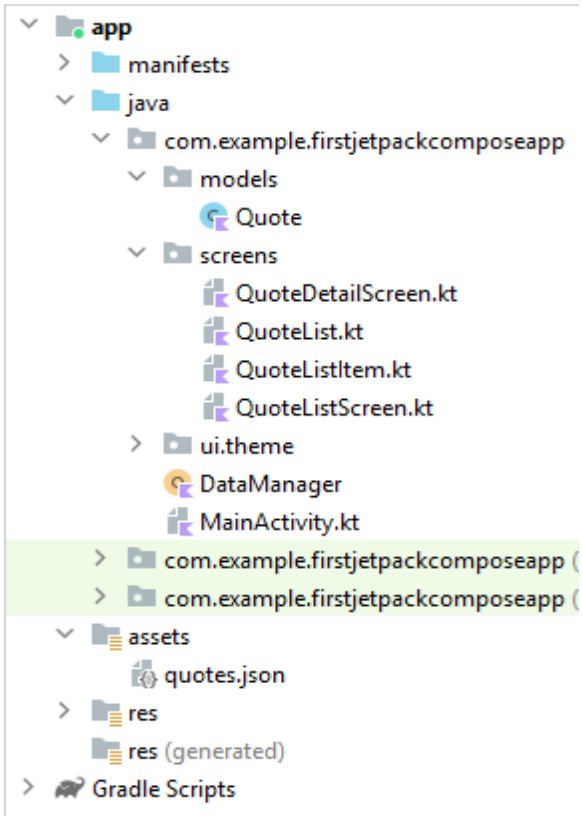
var count: MutableState<Int> = rememberSaveable{mutableStateOf(0)}
Data stored in bundle of onCreate().
So that when we rotate mobile, data not lost.

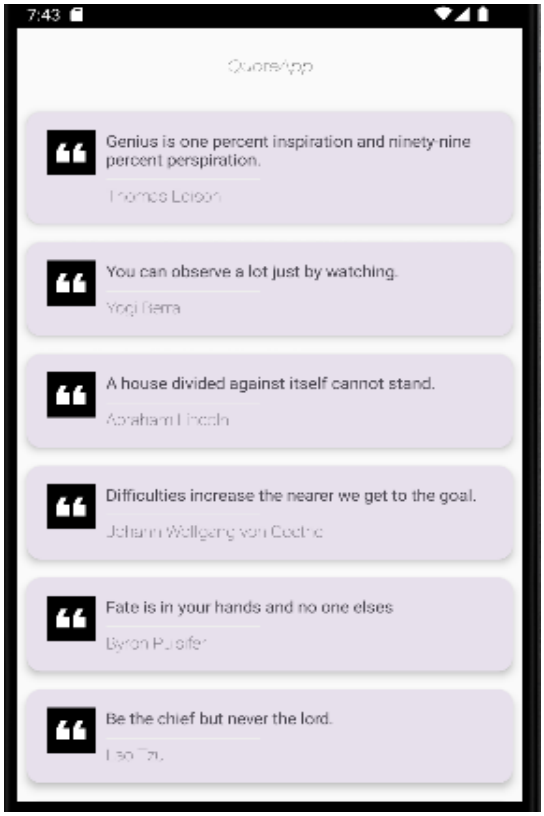
Stateful composable-->NotificationScreen() because it hold state variable count.
Stateless composable-->NotificationCounter(),MessageBar()

Uni-directional flow:
State variable --> Top to Bottom.
Even i.e. button --> Bottom to Top.
ViewModel maintain uni-directional flow.
*/



 # Program8: FirstJetpackComposeApp.





MainActivity.kt:

```
package com.example.firstjetpackcomposeapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import com.example.firstjetpackcomposeapp.screens.QuoteDetail
import com.example.firstjetpackcomposeapp.screens.QuoteListScreen
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      CoroutineScope(Dispatchers.IO).launch {//Flow-1S
         DataManager.loadAssetsFromFile(applicationContext)
      }
      setContent {
        App()  //Flow-3
      }
   }
}

@Composable
fun App(){

   if (DataManager.isDataLoaded.value){
      if(DataManager.currentPage.value == Pages.LISTING){
         QuoteListScreen(data = DataManager.data) {//Flow-9,17
            DataManager.switchPages(it)
         }
      }
      else{   //Flow-5
         DataManager.currentQuote?.let { QuoteDetail(quote = it) }
      }
   }
```

```
    else{
        Box(   //Flow-4
            contentAlignment = Alignment.Center,
            modifier = Modifier.fillMaxSize(1f)
        ) {
            Text(text = "Loading.....",
                style = MaterialTheme.typography.bodyMedium
            )
        }
    }
}

enum class Pages {
    LISTING,
    DETAIL
}
```

QuoteListScreen.kt:

```
package com.example.firstjetpackcomposeapp.screens
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.Font
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import com.example.firstjetpackcomposeapp.R
import com.example.firstjetpackcomposeapp.models.Quote

@Composable
fun QuoteListScreen(data:Array<Quote>, onClick: (quote:Quote)-> Unit) {
    Column {
        Text(
            text = "QuoteApp",
            textAlign = TextAlign.Center,
            modifier = Modifier.padding(8.dp,24.dp).fillMaxWidth(1f),
            fontFamily = FontFamily(Font(R.font.montserrat))
        )
        QuoteList(data = data,onClick) //Flow-10,18
    }
}
```

QuoteList.kt:

```
package com.example.firstjetpackcomposeapp.screens
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.runtime.Composable
import com.example.firstjetpackcomposeapp.models.Quote

@Composable
fun QuoteList(data:Array<Quote>, onClick: (quote:Quote)-> Unit){

   LazyColumn(content = {
      items(data) {
         QuoteListItem(quote = it,onClick) //Flow-11,19
      }
   })
}
```

QuoteListItem.kt:

```
package com.example.firstjetpackcomposeapp.screens
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.FormatQuote
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.rotate
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.ColorFilter
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import com.example.firstjetpackcomposeapp.models.Quote
```

```
@Composable   //Flow-12,20F
fun QuoteListItem(quote: Quote, onClick: (quote:Quote)-> Unit) {
   Card(
      elevation = CardDefaults.cardElevation(4.dp),
      modifier = Modifier
         .clickable{onClick(quote)}  //Flow-13
         .padding(8.dp),
   ) {
      Row(
        modifier = Modifier.padding(16.dp)
      ) {
         Image(
            imageVector = Icons.Filled.FormatQuote,
            contentDescription = "Quote",
            colorFilter = ColorFilter.tint(Color.White),
            alignment = Alignment.TopStart,
            modifier = Modifier
               .size(40.dp)
               .rotate(180f)
               .background(Color.Black)
        )
         Spacer(
              modifier = Modifier.padding(4.dp)
        )
         Column(
              modifier = Modifier.weight(1f)
        ) {
              Text(
                 text = quote.text,
                 modifier = Modifier.padding(0.dp, 0.dp, 0.dp, 8.dp)
              )
              Box(
                 modifier = Modifier
                  .background(Color(0xFFEEEEEE))
                  .fillMaxWidth(.4f)
                  .height(1.dp)
            )
             Text(
                text = quote.author,
                fontWeight = FontWeight.Thin,
                modifier = Modifier.padding(top = 4.dp)
            )
        }
      }
   }
}
```

QuoteDetailScreen.kt:

```
package com.example.firstjetpackcomposeapp.screens
import androidx.activity.compose.BackHandler
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.FormatQuote
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.rotate
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.Font
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.unit.dp
import com.example.firstjetpackcomposeapp.DataManager
import com.example.firstjetpackcomposeapp.R
import com.example.firstjetpackcomposeapp.models.Quote

@Composable
fun QuoteDetail(quote: Quote) {  //Flow-6 ,14
    BackHandler {  //Flow-7,15
        DataManager.switchPages(null)
    }

    Box(
        contentAlignment = Alignment.Center,
        modifier = Modifier
            .fillMaxSize(1f)
            .background(
                Brush.sweepGradient(
                    colors = listOf(
                        Color(0xFFffffff),
                        Color(0xFFE3E3E3)
                    )
                )
            )
```

```
    ) {
        Card(
            elevation = CardDefaults.cardElevation(4.dp),
            modifier = Modifier.padding(32.dp)
        ) {
            Column(
                verticalArrangement = Arrangement.Center,
                modifier = Modifier.padding(16.dp)
            ) {
                Image(
                    imageVector = Icons.Filled.FormatQuote,
                    contentDescription = "Quote",
                    modifier = Modifier.size(80.dp).rotate(180F)
                )

                Text(
                    text = quote.text,
                    fontFamily = FontFamily(Font(R.font.montserrat))
                )
                Spacer(
                    modifier = Modifier.height(16.dp)
                )
                Text(
                    text = quote.author,
                    fontFamily = FontFamily(Font(R.font.montserrat))
                )
            }
        }
    }
}
```

DataManager.kt:

```
package com.example.firstjetpackcomposeapp

import android.content.Context
import androidx.compose.runtime.mutableStateOf
import com.example.firstjetpackcomposeapp.models.Quote
import com.google.gson.Gson

object DataManager {

    var data = emptyArray<Quote>()
    var currentQuote:Quote? = null
```

```
    var currentPage = mutableStateOf(Pages.LISTING)
    var isDataLoaded = mutableStateOf(false)

    fun loadAssetsFromFile(context: Context){   //Flow-2
        val inputStream = context.assets.open("quotes.json")
        val size: Int = inputStream.available()
        val buffer = ByteArray(size)
        inputStream.read(buffer)
        inputStream.close()
        val json = String(buffer, Charsets.UTF_8)
        val gson = Gson()
        data =  gson.fromJson(json, Array<Quote>::class.java)
        isDataLoaded.value = true
    }

    fun switchPages(quote: Quote?){  //Flow-8,16
        if(currentPage.value == Pages.LISTING){
            currentQuote = quote
            currentPage.value == Pages.DETAIL
        }
        else{
            currentPage.value == Pages.LISTING
        }
    }

}
```

Quote.kt:

```
package com.example.firstjetpackcomposeapp.models

data class Quote(val text:String,val author:String)
```

themes.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="Theme.FirstJetpackComposeApp"
parent="android:Theme.Material.Light.NoActionBar">
        <item name="android:statusBarColor" type="color">@color/black</item >
    </style>
</resources>
```

 # Program9: Theme customizing.

MainActivity.kt:

```
package com.ghani.themecustomizing

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.Button
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalConfiguration
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.tooling.preview.Preview
import com.ghani.themecustomizing.ui.theme.ThemeCustomizingTheme

class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)

      setContent {

        /*ThemeCustomizingTheme {
            /*It assign to [content: @Composable () -> Unit]
            in [fun ThemeCustomizingTheme()] of [Theme.kt]*/
            Surface(
               modifier = Modifier.fillMaxSize(),
               color = MaterialTheme.colors.background
            ){
               Greeting("Android")
            }
         }

         ThemeCustomizingTheme(content = {
            Text(text = "Hello")
         })
```

```
        ThemeCustomizingTheme{
           Text(text = "Hello")
        }

        ThemeCustomizingTheme{
           Text(text = "Hello",
              style = MaterialTheme.typography.h1)
        }*/

        App()

      }
   }
}


@Composable
fun App(){
   var theme = remember{mutableStateOf(false)}  //Step-2
   ThemeCustomizingTheme(theme.value){  //Step-3
      Column(Modifier.background(MaterialTheme.colors.background)) {
         Text(text = "Hello! World",style = MaterialTheme.typography.h1)
         Button(onClick = { theme.value = !theme.value }) {  //Step-1
            Text(text = "Change Theme")
         }
      }
   }
}
```

Theme.kt:

```
package com.ghani.themecustomizing.ui.theme

import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material.MaterialTheme
import androidx.compose.material.darkColors
import androidx.compose.material.lightColors
import androidx.compose.runtime.Composable
import androidx.compose.ui.graphics.Color
```

```
private val DarkColorPalette = darkColors(   //Step-6F
   primary = Purple200,
   primaryVariant = Purple700,
   secondary = Teal200,
   background = Color.Black,
   surface = Color.Green
)

private val LightColorPalette = lightColors(
   primary = Purple500,
   primaryVariant = Purple700,
   secondary = Teal200,
   background = Color.Red,
   surface = Color.Cyan
)

   /* Other default colors to override
   background = Color.White,
   surface = Color.White,
   onPrimary = Color.White,
   onSecondary = Color.Black,
   onBackground = Color.Black,
   onSurface = Color.Black */

@Composable
fun ThemeCustomizingTheme(
   darkTheme: Boolean = isSystemInDarkTheme(), //Step-4
   content: @Composable () -> Unit
) {

   val colors = if (darkTheme) { //Step-5
      DarkColorPalette
   } else {
      LightColorPalette
   }

   MaterialTheme(
      colors = colors,
      typography = Typography,
      shapes = Shapes,
      content = content
   )
 }
```

Type.kt:

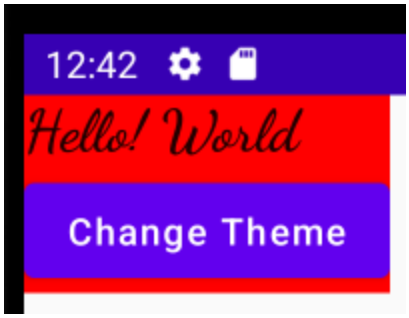
```
package com.ghani.themecustomizing.ui.theme

import androidx.compose.material.Typography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp

// Set of Material typography styles to start with
val Typography = Typography(

    body1 = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp
    ),

    h1 = TextStyle(
    fontFamily = FontFamily.Cursive,
    fontWeight = FontWeight.Bold,
    fontSize = 22.sp
    )

    /* Other default text styles to override
    button = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.W500,
        fontSize = 14.sp
    ),

    caption = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 12.sp
    )*/

)
```

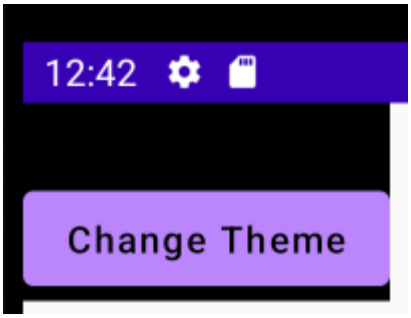
Shape.kt:

```
package com.ghani.themecustomizing.ui.theme

import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Shapes
import androidx.compose.ui.unit.dp

val Shapes = Shapes(
    small = RoundedCornerShape(4.dp),
    medium = RoundedCornerShape(4.dp),
    large = RoundedCornerShape(0.dp)
)
```

Color.kt:

```
package com.ghani.themecustomizing.ui.theme

import androidx.compose.ui.graphics.Color

val Purple200 = Color(0xFFBB86FC)
val Purple500 = Color(0xFF6200EE)
val Purple700 = Color(0xFF3700B3)
val Teal200 = Color(0xFF03DAC5)
```



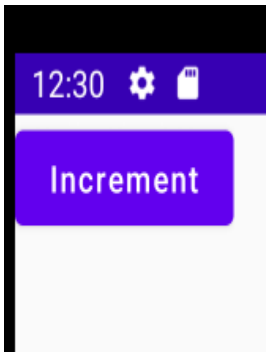


 # Program10: Side Effects.

MainActivity.kt:

```
package com.ghani.sideeffectsjetpackcompose

import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.Button
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import com.ghani.sideeffectsjetpackcompose.ui.theme.SideEffectsJetpackComposeTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            //ListComposable()
            //Counter1()
            Counter2()
        }
    }
}

var count = 0
@Composable
fun HasSideEffects() {
    count++  //Problem count++
    Text(text = "Hello! World")
}
```

```
@Composable
fun ListComposable(){
   val categoryState = remember{mutableStateOf(emptyList<String>())}
   //categoryState.value = fetchCategories() //Problem fetchCategories()

   LaunchedEffect(key1 = Unit){ //Run one time in CoroutineScope.
      categoryState.value = fetchCategories()
   }

   LazyColumn {
      items(categoryState.value){  item ->
         Text(text = item)
      }
   }
}
fun fetchCategories(): List<String> {
   //assuming network call
   return listOf("One","Two","Three")
}

@Composable
fun Counter1(){
   var count = remember{mutableStateOf(0)}
   LaunchedEffect(key1 = false){
      Log.d("Counter1 ","Current count:${count.value}")
   }
   Button(onClick = { count.value++ }) {
      Text(text = "Increment")
   }
}

@Composable
fun Counter2(){
   var count = remember{mutableStateOf(0)}
   var key = count.value % 3 == 0
   LaunchedEffect(key1 = key){
      Log.d("Counter","Current count:${count.value}")
   }
   Button(onClick = { count.value++ }) {
      Text(text = "Increment")
   }
}
```




```
Counter1                com.ghani.sideeffectsjetpackcompose  D  Current count:0
```

```
Counter                 com.example.blogapp                  D  Current count: 0
Counter                 com.example.blogapp                  D  Current count: 1
Counter                 com.example.blogapp                  D  Current count: 3
Counter                 com.example.blogapp                  D  Current count: 4
Counter                 com.example.blogapp                  D  Current count: 6
```

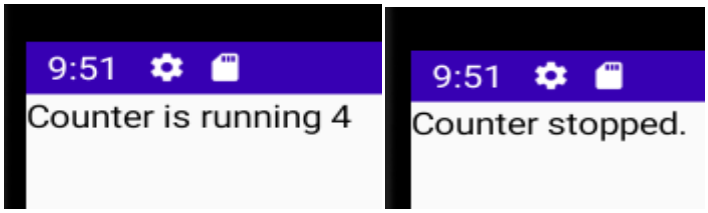
 # Program11: Side Effects With Coroutine.

MainActivity.kt:

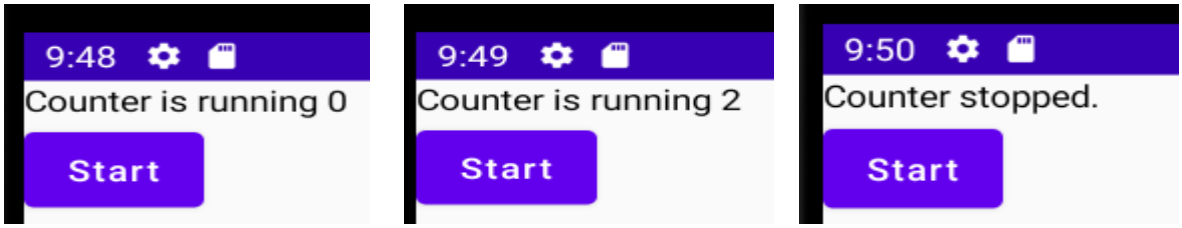
```
package com.ghani.sideeffectwithcoroutine

import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.Button
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import com.ghani.sideeffectwithcoroutine.ui.theme.SideEffectWithCoroutineTheme
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
```

```
class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
         //LaunchedEffectComposable()
         CoroutineScopeComposable()
      }
   }
}

@Composable
fun LaunchedEffectComposable() {
   val counter = remember{mutableStateOf(0)}

   LaunchedEffect(key1 = Unit){
      Log.d("LaunchedEffect","Started")
      try {
        for (i in 1 .. 10){
         counter.value++
         delay(1000)
        }
      }catch (e : Exception){
         Log.d("LaunchedEffectComposable","Exception: ${e.message.toString()}")
      }
   }

   var text = "Counter is running ${counter.value}"
   if (counter.value == 10){
      text = "Counter stopped."
   }
   Text(text = text)
}



/*Lack of LaunchedEffectComposable:
(1)Only can run in initial composition,
  but not in button or any other event composition.
(2)Coroutine can't launch  independently,
  we can't use rememberCoroutineScope().
(3)Coroutine can't cancel/delay/join.
*/
```

```
@Composable
fun CoroutineScopeComposable() {
   val counter = remember{mutableStateOf(0)}
   var scope = rememberCoroutineScope()

   var text = "Counter is running ${counter.value}"
   if (counter.value == 10){
      text = "Counter stopped."
   }

   Column {
      Text(text = text)
      Button(onClick = {
         scope.launch{
            Log.d("CoroutineScopeComposable","Started")
            try {
               for (i in 1 .. 10){
                  counter.value++
                  delay(1000)
               }
            }catch (e : Exception){
               Log.d("CoroutineScopeComposable","Exception: ${e.message.toString()}")
            }
         }
      }) {
            Text(text = "Start")
      }
   }
}
```



LaunchedEffectComposable()



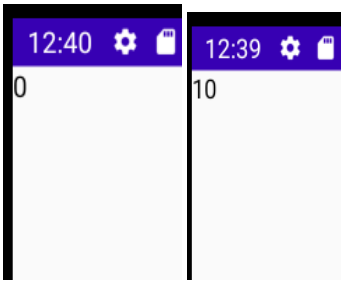
CoroutineScopeComposable()

 # Program12: Side Effects Handlers.

MainActivity.kt:

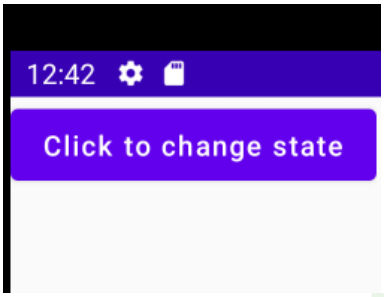
```
package com.ghani.sideeffecthandlers

import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.runtime.*
import kotlinx.coroutines.delay

class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
         //App1()
         App2()
      }
   }
}

@Composable
fun App1(){
   val counter = remember{mutableStateOf(0)}
   LaunchedEffect(key1 = Unit){
      delay(5000)
      counter.value = 10
   }
   Counter(counter.value)
}

/*
@Composable
fun Counter(value: Int) {
   LaunchedEffect(key1 = Unit){
      delay(15000)
   Log.d("LaunchedEffect",value.toString()) //value = 0
   }
   Text(text = value.toString())
}
```

```
@Composable
fun Counter(value: Int) {
   LaunchedEffect(key1 = value){
      delay(15000)  //Affected
      Log.d("LaunchedEffect",value.toString())  //value = 10
   }
   Text(text = value.toString())
}
*/

@Composable
fun Counter(value: Int) {
   val state = rememberUpdatedState(newValue = value)
   LaunchedEffect(key1 = Unit){
      delay(15000)   //Not affected
      Log.d("LaunchedEffect",state.value.toString())  //state = 10
   }
   Text(text = value.toString())
}




fun A(){Log.d("Side Effect Handlers","I am A from app")}
fun B(){Log.d("Side Effect Handlers","I am B from app")}

@Composable
fun App2(){
   val state = remember{mutableStateOf(::A)}
   Button(onClick = { state.value = ::B }) {
      Text(text = "Click to change state")
   }
 LoadingScreen(state.value)
}

@Composable
fun LoadingScreen(onTimeout: () -> Unit) {
   val currentOnTimeout by rememberUpdatedState(onTimeout)
   LaunchedEffect(true){
     delay(5000)
     currentOnTimeout()
   }
}
```




```
LaunchedEffect          com.ghani.sideeffecthandlers          D  0
LaunchedEffect          com.ghani.sideeffecthandlers          D  10
```

App1()




```
Side Effect Handlers    com.ghani.sideeffecthandlers          D  I am A from app
Side Effect Handlers    com.ghani.sideeffecthandlers          D  I am B from app
```

App2()

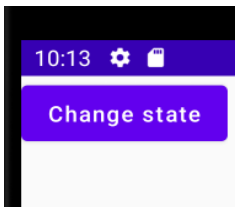
 # Program13: DisposableSide Effects.

MainActivity.kt:

```
package com.ghani.disposablessideeffect
import android.media.MediaPlayer
import android.os.Bundle
import android.util.Log
import android.view.ViewTreeObserver
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.material.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.DisposableEffect
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalView
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
```

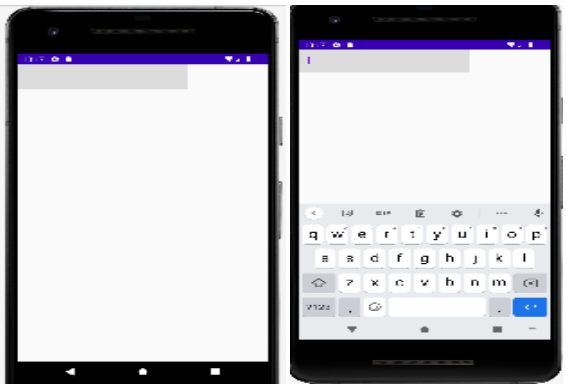
```
class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
         //App()
         //mediaDisposable()
         KeyboardApp()
      }
   }
}

@Composable
fun App(){
   var state = remember{mutableStateOf(false)}
   DisposableEffect(key1 = state.value){
      Log.d("DisposableEffect","Disposable effect started")
      onDispose {
         Log.d("DisposableEffect","Clean up side effects")
      }
   }
   Button(onClick = { state.value = !state.value }) {
      Text(text = "Change state")
   }
}

/*
Button click -> State change -> key1 change -> DisposableEffect again run
-> onDispose run at first -> Rest of code then run.
*/




@Composable
fun mediaDisposable(){
   val context = LocalContext.current
   DisposableEffect(key1 = Unit){
      val mediaPlayer = MediaPlayer.create(context,R.raw.desert_voices)
      mediaPlayer.start()
      onDispose {
         mediaPlayer.stop()
         mediaPlayer.release()
      }
   }
}
```

```
@Composable
fun KeyboardApp(){
   KeyboardComposable()
   TextField(value = "", onValueChange = { } )
}
@Composable
fun KeyboardComposable(){
   val view = LocalView.current
   //Mobile layout current view.
   DisposableEffect(key1 = Unit){
      val listener = ViewTreeObserver.OnGlobalLayoutListener {
         val insets = ViewCompat.getRootWindowInsets(view)
         //Calculate no. of rectangular/insets in view.
         val isKeyboardVisible = insets?.isVisible(WindowInsetsCompat.Type.ime())
         //Check is keyboard visible in insets where .ime is keyboard key.
         Log.d("isKeyboardVisible",isKeyboardVisible.toString())
      }
      view.viewTreeObserver.addOnGlobalLayoutListener(listener)
      //When something change in mobile layout, val listener will call.
      onDispose {
         view.viewTreeObserver.removeOnGlobalLayoutListener(listener)
         // Remove val listener for memory leak.
      }
   }
}
```



App()

```
DisposableEffect        com.ghani.disposablessideeffect      D  Disposable effect started
DisposableEffect        com.ghani.disposablessideeffect      D  Clean up side effects
DisposableEffect        com.ghani.disposablessideeffect      D  Disposable effect started
```



KeyboardApp()

```
isKeyboardVisible       com.ghani.disposablessideeffect      D  false
isKeyboardVisible       com.ghani.disposablessideeffect      D  true
```

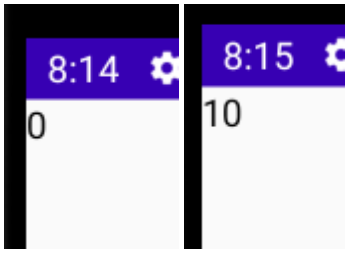
 # Program14: produceState & derivedState.

MainActivity.kt:

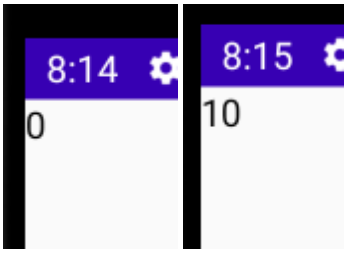
```
package com.ghani.sideeffectproducestate
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Refresh
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.rotate
import androidx.compose.ui.unit.dp
import kotlinx.coroutines.delay

class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
         //Counter1()
         //Counter2()
         //Loader()
         //Derived1()
         Derived2()
      }
   }
}

@Composable
fun Counter1() {
   val state = remember { mutableStateOf(0) }
   LaunchedEffect(key1 = Unit) {
      for (i in 1..10) {
         delay(1000)
         state.value++
      }
   }
   Text(text = state.value.toString())
}
```

```
@Composable
fun Counter2() {
   val state = produceState(initialValue = 0) {
      for (i in 1..10) {
         delay(1000)
         value += 1
      }
   }
   Text(text = state.value.toString())
}

@Composable
fun Loader() {
   val degree = produceState(initialValue = 0 ){
      while(true){
         delay(16)
         value = (value + 20) % 360
      }
   }
   Box(contentAlignment = Alignment.Center,
      modifier = Modifier.fillMaxSize(1f),
      content = {
         Column(horizontalAlignment = Alignment.CenterHorizontally) {
            Image(imageVector = Icons.Default.Refresh,
               contentDescription = "",
               modifier = Modifier
                  .size(60.dp)
                  .rotate(degree.value.toFloat()))
            Text(text = "Loading")
         }
      }
   )
}

@Composable
fun Derived1() {
   val tableOf = remember { mutableStateOf(5) }
   val index =  remember { mutableStateOf(1) }
   val message = derivedStateOf {
      "${tableOf.value } * ${index.value} = ${tableOf.value * index.value }"
   }
   Box(contentAlignment = Alignment.Center,
      modifier = Modifier.fillMaxSize(1f)
   ){
      Text(text = message.value)
   }
}
```

```
@Composable
fun Derived2() {
   val tableOf = remember { mutableStateOf(5) }
   val index =  produceState(initialValue = 1){
      repeat(9){
         delay(1000)
         value += 1
      }
   }
   val message = derivedStateOf {
      "${tableOf.value } * ${index.value} = ${tableOf.value * index.value }"
   }
   Box(contentAlignment = Alignment.Center,
      modifier = Modifier.fillMaxSize(1f)
   ){
      Text(text = message.value)
   }
}
```

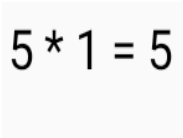


Counter1()

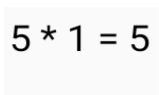


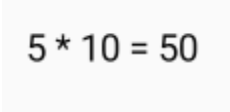
Counter2()

Loader()



Derived1()





Derived2()

 # Program15: App using navigation.

TweetListItem.kt:

```
package com.example.tweetsy.models

data class TweetListItem(  //Step-19
   val category: String,
   val text: String
)
```

TweetsyAPI.kt:

```
package com.example.tweetsy.api

import com.example.tweetsy.models.TweetListItem
import retrofit2.Response
import retrofit2.http.GET
import retrofit2.http.Header
import retrofit2.http.Headers

interface TweetsyAPI {

   @GET("/v3/b/64b3dd858e4aa6225ebf1315?meta=false") //Step-18,20
   suspend fun getTweets(@Header("X-JSON-Path") category: String) :
Response<List<TweetListItem>>

   @GET("/v3/b/64b3dd858e4aa6225ebf1315?meta=false")
   @Headers("X-JSON-Path: tweets..category")
   suspend fun getCategories(): Response<List<String>> //Step-6

}
```

NetworkModule.kt:

```
package com.example.tweetsy.di
import com.example.tweetsy.api.TweetsyAPI
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
class NetworkModule {

   @Singleton
   @Provides
   fun providesRetrofit(): Retrofit{
      return Retrofit.Builder().baseUrl("https://api.jsonbin.io/")
         .addConverterFactory(GsonConverterFactory.create())
         .build()
   }

   @Singleton
   @Provides
   fun provideTweetsyAPI(retrofit: Retrofit) : TweetsyAPI{
      return retrofit.create(TweetsyAPI::class.java)
   }

}
```

TweetRepository.kt:

```
package com.example.tweetsy.repository

import com.example.tweetsy.api.TweetsyAPI
import com.example.tweetsy.models.TweetListItem
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import javax.inject.Inject

class TweetRepository @Inject constructor(private val tweetsyAPI: TweetsyAPI) {

    private val _tweets = MutableStateFlow<List<TweetListItem>>(emptyList())
    val tweets: StateFlow<List<TweetListItem>> //Step-21
        get() = _tweets

    private val _categories = MutableStateFlow<List<String>>(emptyList())
    val categories: StateFlow<List<String>>  //Step-7
        get() = _categories


    suspend fun getTweets(category:String) {
        //Step-17
        val result = tweetsyAPI.getTweets("tweets[?(@.category==\"$category\")]")
        if (result.isSuccessful && result.body() != null) {
            _tweets.emit(result.body()!!)
        }
    }

    suspend fun getCategories() {
        val result = tweetsyAPI.getCategories()//Step-5
        if (result.isSuccessful && result.body() != null) {
            _categories.emit(result.body()!!)
        }
    }

}
```

DetailViewModel.kt:

```
package com.example.tweetsy.viewmodels

import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.tweetsy.models.TweetListItem
import com.example.tweetsy.repository.TweetRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class DetailViewModel @Inject constructor(
    private val repository: TweetRepository,
    private val savedStateHandle: SavedStateHandle) : ViewModel() {

    val tweets: StateFlow<List<TweetListItem>>  //Step-22
    get() = repository.tweets

    init {
        viewModelScope.launch {
            val category = savedStateHandle.get<String>("category") ?: "motivation"
            repository.getTweets(category) //Step-16
        }
    }

}
```

DetailScreen.kt:

```
package com.example.tweetsy.screens
import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.Card
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import com.example.tweetsy.viewmodels.DetailViewModel

@Composable
fun DetailScreen() {
   val detailViewModel: DetailViewModel = hiltViewModel()
   //Step-15
   val tweets =  detailViewModel.tweets.collectAsState()
   LazyColumn(content = {
      //Step-23
      items(tweets.value){
         TweetListItem(tweet = it.text) //Step-24
      }
   })
}
@Composable
fun TweetListItem(tweet: String) {
   Card(
      modifier = Modifier
         .fillMaxWidth()
         .padding(16.dp),
      border = BorderStroke(1.dp, Color(0xFFCCCCCC)),
      content = {
         Text(
            text = tweet,//Step-25F
            modifier = Modifier.padding(16.dp),
            style = MaterialTheme.typography.body2
        )
      }
   )
}
```

CategoryViewModel.kt:

```
package com.example.tweetsy.viewmodels

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.tweetsy.repository.TweetRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class CategoryViewModel @Inject constructor(private val repository: TweetRepository) :
ViewModel() {

   val categories: StateFlow<List<String>>  ////Step-8
   get() = repository.categories

   init {
      viewModelScope.launch {
         repository.getCategories() //Step-4
      }
   }
}
```

CategoryScreen.kt:

```
package com.example.tweetsy.screens

import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.grid.GridCells
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.foundation.lazy.grid.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.State
import androidx.compose.runtime.collectAsState
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.draw.paint
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import com.example.tweetsy.viewmodels.CategoryViewModel
import com.example.tweetsy.R

@Composable
fun CategoryScreen(onClick: (category: String) -> Unit) {
    val categoryViewModel: CategoryViewModel = hiltViewModel()
    //Step-3
    val categories: State<List<String>> = categoryViewModel.categories.collectAsState()

    if (categories.value.isEmpty()) {
        Box(
            modifier = Modifier.fillMaxSize(1f),
            contentAlignment = Alignment.Center
        ) {
            Text(text = "Loading...", style = MaterialTheme.typography.h3)
        }
    } else {
        LazyVerticalGrid(
            columns = GridCells.Fixed(2),
```

```
            contentPadding = PaddingValues(8.dp),
            verticalArrangement = Arrangement.SpaceAround,
        ) {
            //Step-9
            items(categories.value.distinct()) {
                CategoryItem(category = it, onClick) //Step-10
            }
        }
    }

}

@Composable
fun CategoryItem(category: String, onClick: (category: String) -> Unit) {
    Box(
        modifier = Modifier
            .padding(4.dp)
            .clickable {  //Step-12
                onClick(category)
            }
            .size(160.dp)
            .clip(RoundedCornerShape(8.dp))
            .paint(
                painter = painterResource(id = R.drawable.bg),
                contentScale = ContentScale.Crop
            )
            .border(1.dp, Color(0xFFEEEEEE)),
        contentAlignment = Alignment.BottomCenter
    ) {
        Text(
            text = category,  //Step-11
            fontSize = 18.sp,
            color = Color.Black,
            modifier = Modifier.padding(0.dp, 20.dp),
            style = MaterialTheme.typography.body1
        )
    }
}
```

MainActivity.kt:

```
package com.example.weetsy
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.padding
import androidx.compose.material.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.navigation.NavType
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import androidx.navigation.navArgument
import com.example.tweetsy.screens.CategoryScreen
import com.example.tweetsy.screens.DetailScreen
import com.example.tweetsy.ui.theme.TweetsyTheme
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContent {
         TweetsyTheme {
            Scaffold(
               topBar = {
                  TopAppBar(
                     title = { Text(text = "Tweetsy") },
                     backgroundColor = Color.Black,
                     contentColor = Color.White
                  )
               }
            ) {
               Box(modifier = Modifier.padding(it)) {
                  App() //Step-1
               }
            }
         }
      }
   }
}
```

```
@Composable
fun App() {
   val navController = rememberNavController()
   NavHost(navController = navController, startDestination = "category") {
      composable(route = "category") {
         //Step-2
         CategoryScreen {
            navController.navigate("detail/${it}") //Step-13
         }
      }
      composable(route = "detail/{category}",
               arguments = listOf(
                  navArgument("category") {
                     type = NavType.StringType
                  }
               )
      ) {
         DetailScreen() //(Step-14)
      }
   }
}
```

Tweetsy.kt:

```
package com.example.tweetsy

import android.app.Application
import dagger.hilt.android.HiltAndroidApp

@HiltAndroidApp
class Tweetsy : Application() {
}
```

AndroidManifest.xml:

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

build.gradle(Project):

```
buildscript {
   ext {
      compose_ui_version = '1.2.0'
   }
}// Top-level build file where you can add configuration options common to all sub-
projects/modules.
plugins {
   id 'com.android.application' version '7.3.1' apply false
   id 'com.android.library' version '7.3.1' apply false
   id 'org.jetbrains.kotlin.android' version '1.8.0' apply false
   id 'com.google.dagger.hilt.android' version '2.44' apply false
}
```

build.gradle(App):

```
plugins {
   id 'kotlin-kapt'
   id 'com.google.dagger.hilt.android'
}
buildFeatures {
   compose true
}
composeOptions {
   kotlinCompilerExtensionVersion '1.4.0'
}
dependencies {
   //HILT
   implementation "com.google.dagger:hilt-android:2.44"
   kapt "com.google.dagger:hilt-compiler:2.44"
   def lifecycle_version = "2.6.1"
   // ViewModel
   implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"
   // ViewModel utilities for Compose
   implementation "androidx.lifecycle:lifecycle-viewmodel-compose:$lifecycle_version"
   //Coroutines
   implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:1.7.0"
   implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.0"
   //Retrofit
   implementation 'com.squareup.retrofit2:retrofit:2.9.0'
   implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
   //Navigation
   implementation "androidx.navigation:navigation-compose:2.6.0"
   implementation 'androidx.hilt:hilt-navigation-compose:1.0.0'
}
```