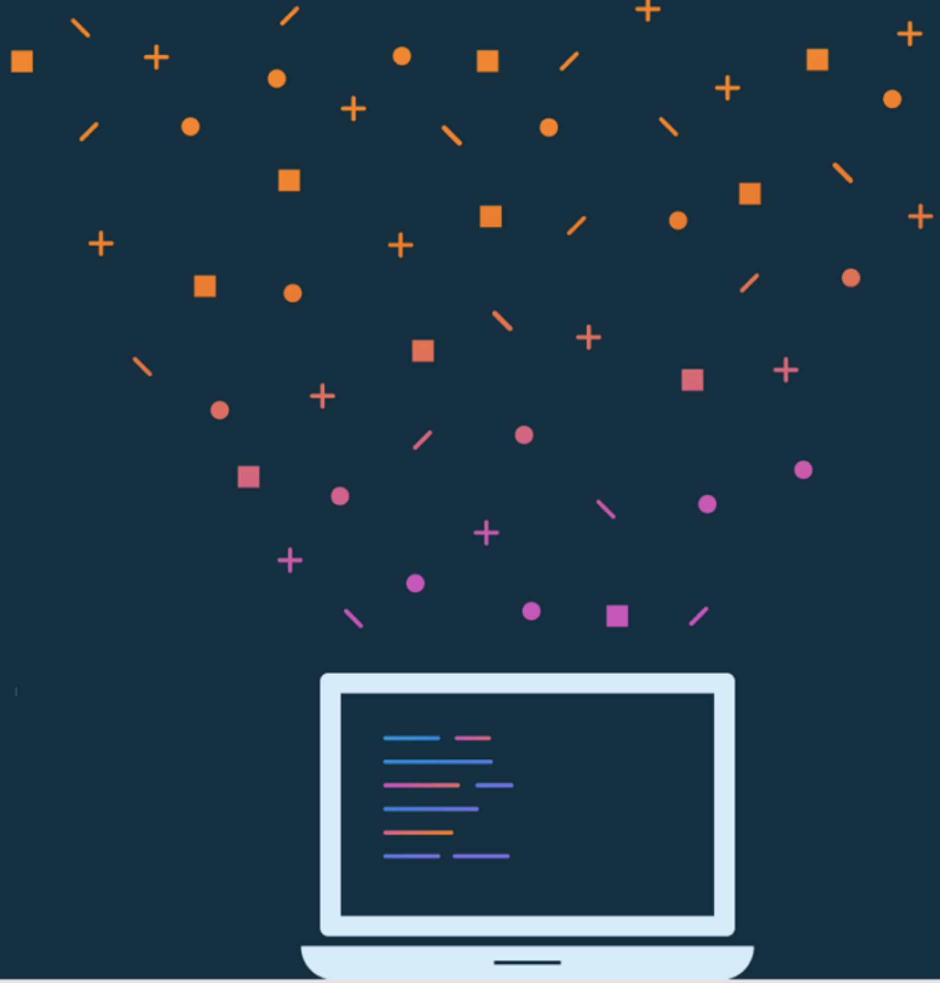
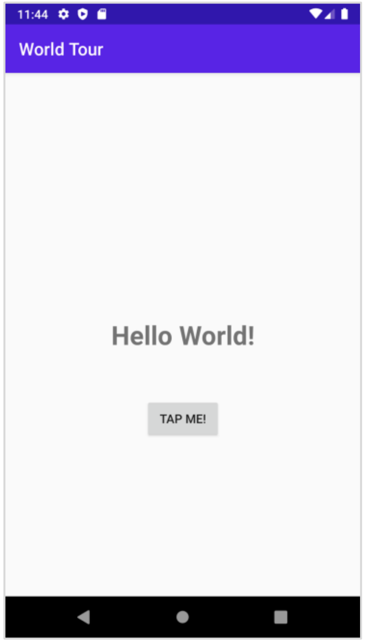
Android Development with Kotlin



Make an app interactive

Define app behavior in Activity

Modify the Activity so the app responds to user input, such as a button tap.



Modify a View dynamically

Within `MainActivity.kt`:

Get a reference to the View in the view hierarchy:

```
val resultTextView: TextView = findViewById(R.id.textView)
```

Change properties or call methods on the View instance:

```
resultTextView.text = "Goodbye!"
```

Event Handling

Events: Something that happens

- In UI: Click, tap, drag
- Device: DetectedActivity such as walking, driving, tilting
- Events are "noticed" by the Android system

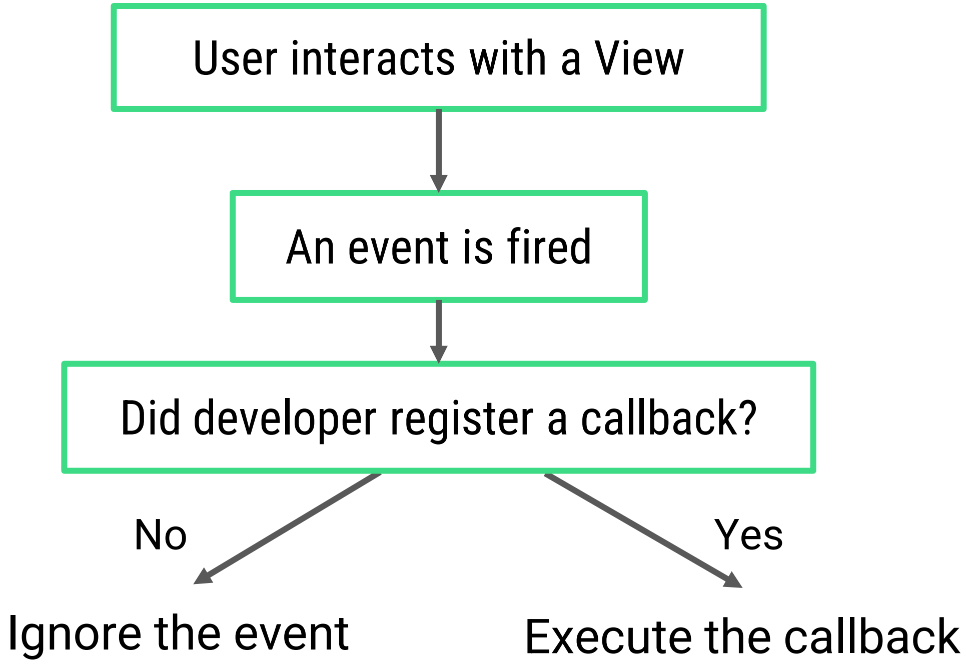
Event Handling

Event Handlers:

Methods that do something in response to a click

- A method, called an **event handler**, is triggered by a specific event and does something in response to the event

Set up listeners for specific events


User interacts with a View
An event is fired
Did developer register a callback?
No
Yes
Ignore the event
Execute the callback

View.OnClickListener

```
class MainActivity : AppCompatActivity(), View.OnClickListener {

   override fun onCreate(savedInstanceState: Bundle?) {
       ...
       val button: Button = findViewById(R.id.button)
       button.setOnClickListener(this)
   }

   override fun onClick(v: View?) {
       TODO("not implemented")
   }
}
```

SAM (single abstract method)

Converts a function into an implementation of an interface

Format: `InterfaceName { lambda body }`

```
val runnable = Runnable { println("Hi there") }
```

is equivalent to

```
val runnable = (object: Runnable {
    override fun run() {
        println("Hi there")
    }
})
```

View.OnClickListener as a SAM

A more concise way to declare a click listener

```
class MainActivity : AppCompatActivity() {

   override fun onCreate(savedInstanceState: Bundle?) {
       ...

       val button: Button = findViewById(R.id.button)
       button.setOnClickListener({ view -> /* do something*/ })
   }
}
```

Late initialization

```
class Student(val id: String) {

    lateinit var records: HashSet<Any>

    init {
        // retrieve records given an id
    }
}
```

Lateinit example in Activity

```
class MainActivity : AppCompatActivity() {

    lateinit var result: TextView

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        result = findViewById(R.id.result_text_view)
    }
}
```

Multiple activities and intents

Multiple screens in an app

Sometimes app functionality may be separated into multiple screens.

Examples:

- View details of a single item (for example, product in a shopping app)
- Create a new item (for example, new email)
- Show settings for an app
- Access services in other apps (for example, photo gallery or browse documents)

What is Intent?

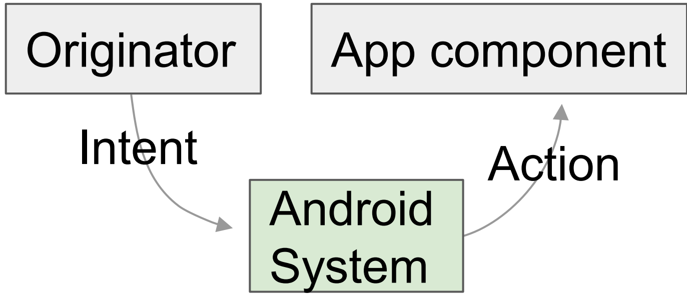
- An Intent is a description of an operation to be performed.
- A facility for late run-time binding between components.
- Can launch a component in the same or a different application.
- A passive data structure holding an abstract description of an operation to be performed.
- An Intent is an object used to request an action from another app component via the Android system.

What can intents do?

- **Start an Activity**
  - A button click starts a new Activity for text entry
  - Clicking Share opens an app that allows you to post a photo

- **Start a Service**

  Initiate downloading a file in the background



- **Deliver Broadcast**

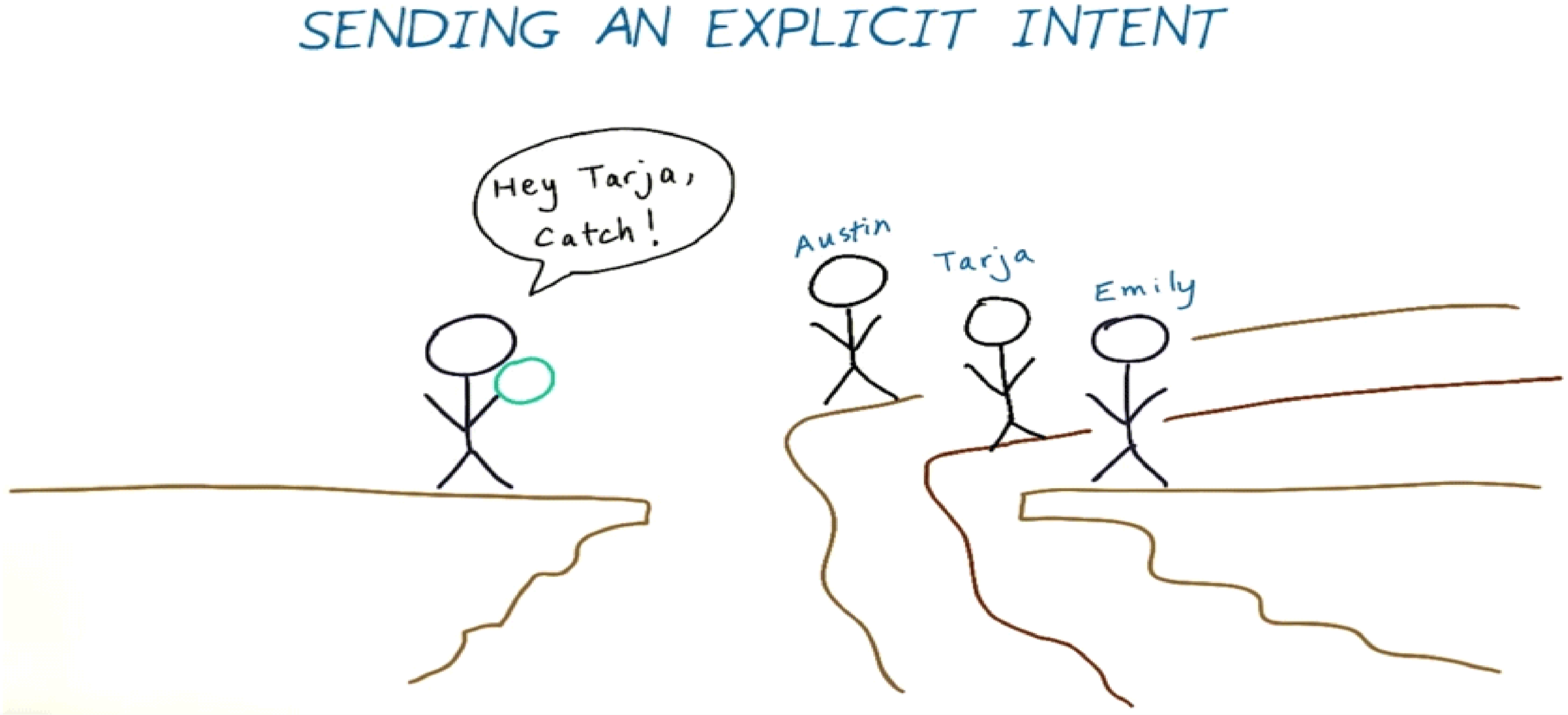
  The system informs everybody that the phone is now charging

Intent

Requests an action from another app component, such as another Activity

- An `Intent` usually has two primary pieces of information:
    - Action to be performed
      (for example, `ACTION_VIEW`, `ACTION_EDIT`, `ACTION_MAIN`)
    - Data to operate on
      (for example, a person's record in the contacts database)
- Commonly used to specify a request to transition to another Activity

Explicit intent



Explicit intent cont’d

Start an Activity with an explicit intent

To start a specific Activity, use an explicit Intent

1. Create an Intent
```
val intent =
        Intent(this, NoteDetailActivity::class.java)
```
2. Use the Intent to start the Activity
```
startActivity(intent)
```

Explicit intent examples

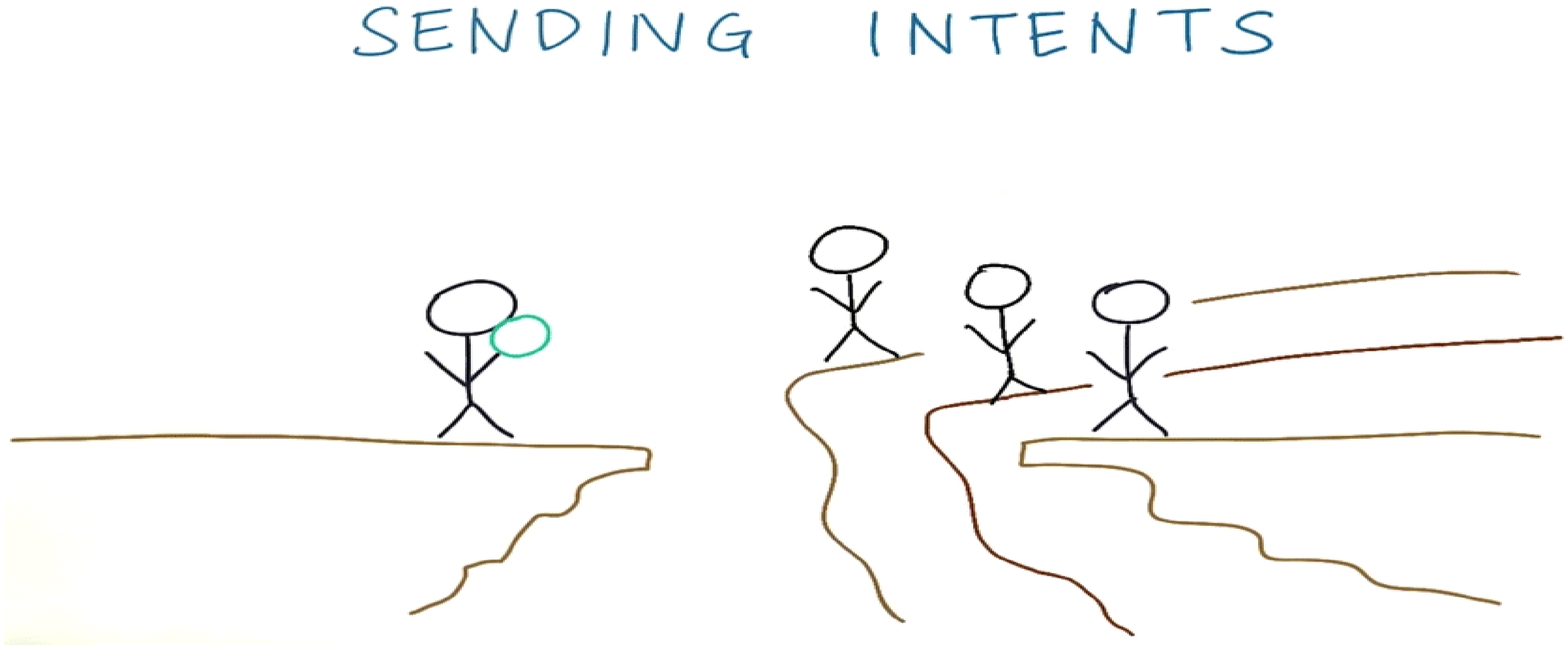
Navigate between activities in your app:

```
fun viewNoteDetail() {
   val intent = Intent(this, NoteDetailActivity::class.java)
   intent.putExtra(NOTE_ID, note.id)
   startActivity(intent)
}
```

Navigate to a specific external app:

```
fun openExternalApp() {
   val intent = Intent("com.example.workapp.FILE_OPEN")
   if (intent.resolveActivity(packageManager) != null) {
       startActivity(intent)
   }
}
```

Implicit intent



Implicit intent cont’d

Start an Activity with an implicit intent

To ask Android to find an Activity to handle your request, use an implicit Intent

1. Create an Intent ➔ `val intent = Intent(action, uri)`
2. Use the Intent to start the Activity ➔ `startActivity(intent)`

- Must have an Action:
  The type of things that the app wants to have done on its behave
- Common Actions:
  - ACTION_VIEW
  - ACTION_EDIT
  - ACTION_DIAL

Implicit intent example

For example, to Start an Activity with an implicit intent:

- **ACTION_VIEW**

Show a web page

```
val uri = Uri.parse("http://www.google.com")
val intent = Intent(Intent.ACTION_VIEW, uri)
startActivity(intent)
```

Implicit intent example

For example, to Start an Activity with an implicit intent:

- **ACTION_DIAL**

Dial a phone number

```
val uri = Uri.parse("tel:8005551234")
val intent = Intent(Intent.ACTION_DIAL, uri)
startActivity(intent)
```

Sending and Retrieving data

In the first (sending) Activity:

1. Create the Intent object
2. Put **data** or **extras** into that Intent
3. Start the new Activity with startActivity()

In the second (receiving) Activity:

1. Get the Intent object, the Activity was started with
2. Retrieve the data or extras from the Intent object

Two types of sending data with intents

- **Data:**
  one piece of information whose data location can be represented by a URI Intent

- **Extras:**
  one or more pieces of information as a collection of key-value pairs in a Bundle

Putting a URI as intent data

```
// A web page URL
intent.data = Uri.parse("http://www.google.com")

// a Sample file URI
intent.data =
      Uri.fromFile(File("/sdcard/sample.jpg"))
```

Putting information into intent extras

- putExtra(String name, int value)
```
intent.putExtra("level", 406)
```
- putExtra(String name, String[ ] value)
```
val foodList = arrayOf("Rice", "Beans", "Fruit")
intent.putExtra("food", foodList)
```
- putExtras(bundle)

  if lots of data, first create a bundle and pass the bundle.
- See documentation for all

Implicit intent example

```
fun sendEmail() {
    val intent = Intent(Intent.ACTION_SEND)
    intent.type = "text/plain"
    intent.putExtra(Intent.EXTRA_EMAIL, emailAddresses)
    intent.putExtra(Intent.EXTRA_TEXT, "How are you?")

    if (intent.resolveActivity(packageManager) != null) {
        startActivity(intent)
    }
}
```

Get data from intents

- getData()

```
    val locationUri = intent.data
```

- int getIntExtra(String keyName, int defaultValue)

```
    val level = intent.getIntExtra("level", 0)
```

- Get all the data at once as a bundle.

```
     val bundle = intent.extras
```

- See documentation for all

Save state

User expects UI state to stay the same after a config change or if the app is terminated when in the background.

- Activity is destroyed and restarted,
  or app is terminated and activity is started.
- Store user data needed to reconstruct app and activity Lifecycle changes:
  - Use `Bundle` provided by `onSaveInstanceState()`.
  - `onCreate()` receives the `Bundle` as an argument when activity is created again.

Activity instance state

- State information is created while the Activity is running, such as a counter, user text, animation progression

- State is lost when the device is rotated, language changes, back-button is pressed, or the system clears the memory

What the system saves?

- System saves only:
    - State of views with a unique ID (android: id) such as text entered into EditText
    - Intent that started activity and data in its extras

- You are responsible for saving other activity and user progress data

Saving instance state

Implement onSaveInstanceState() in your Activity

- Called by Android runtime when there is a possibility the Activity may be destroyed
- Saves data only for this instance of the Activity during the current session

onSaveInstanceState(outState: Bundle)

```
override fun onSaveInstanceState(outstate: Bundle) {
    super.onSaveInstanceState(outstate)

    // adding information for saving to the outState bundle
    outState.putString("key", value)
}
```

Restoring instance state

Two ways to retrieve the saved Bundle

1. in `onCreate(savedInstanceState: Bundle?)`
   Preferred, to ensure that your user interface, including any saved state, is back up and running as quickly as possible

2. Implement callback (called after onStart())
   onRestoreInstanceState(mySavedState: Bundle)

1. Restoring in onCreate()

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    if(savedInstanceState != null) {
        val myValue = savedInstanceState.getString("key")
        if(myValue != null) {
            // Continue your work task with myValue
        }
    }
}
```

2. onRestoreInstanceState()

```
override fun onRestoreInstanceState(savedInstanceState: Bundle) {
    super.onRestoreInstanceState(savedInstanceState)

    val myValue = savedInstanceState.getString("key")
    if(myValue != null) {
        // Continue your work task with myValue
    }
}
```

Instance state and app restart

When you stop and restart a new app session,

the Activity instance states are lost,

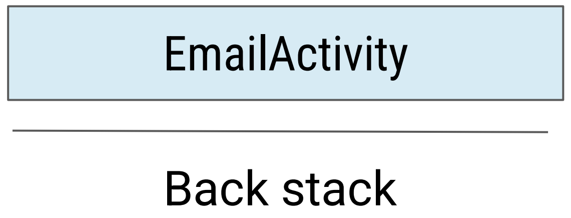
and your activities will revert to their default appearance

If you need to save user data between app sessions,

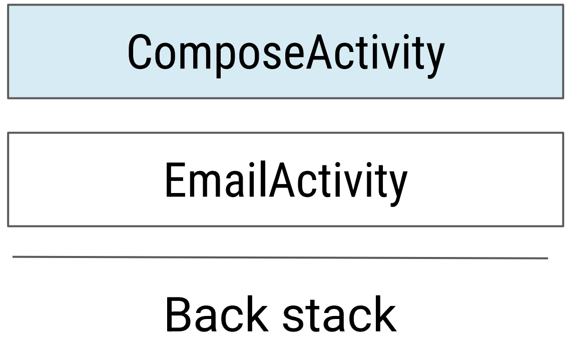
use **shared preferences** or a **database**.

Tasks and back stack

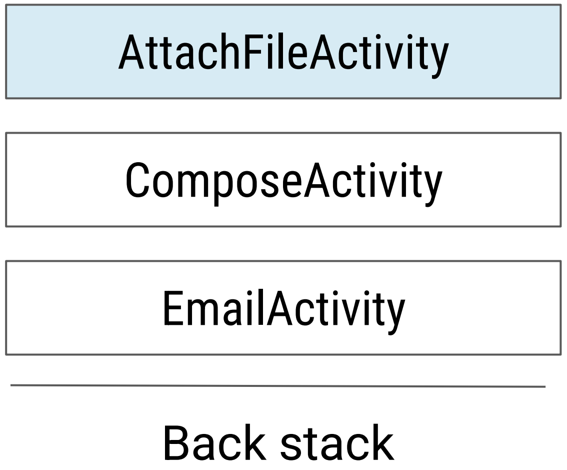
Back stack of activities



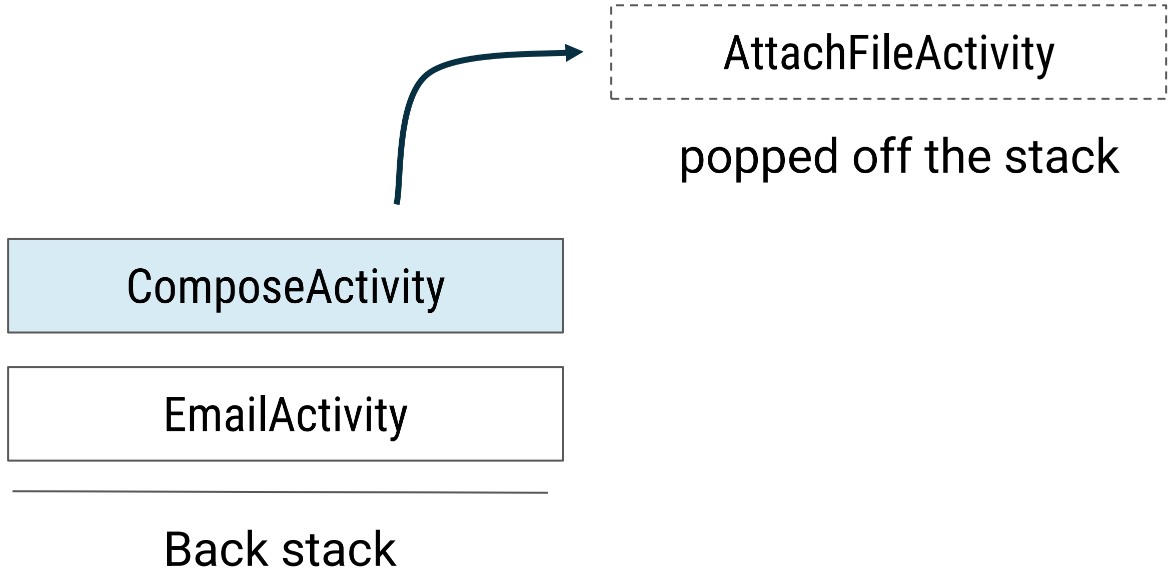
Add to the back stack



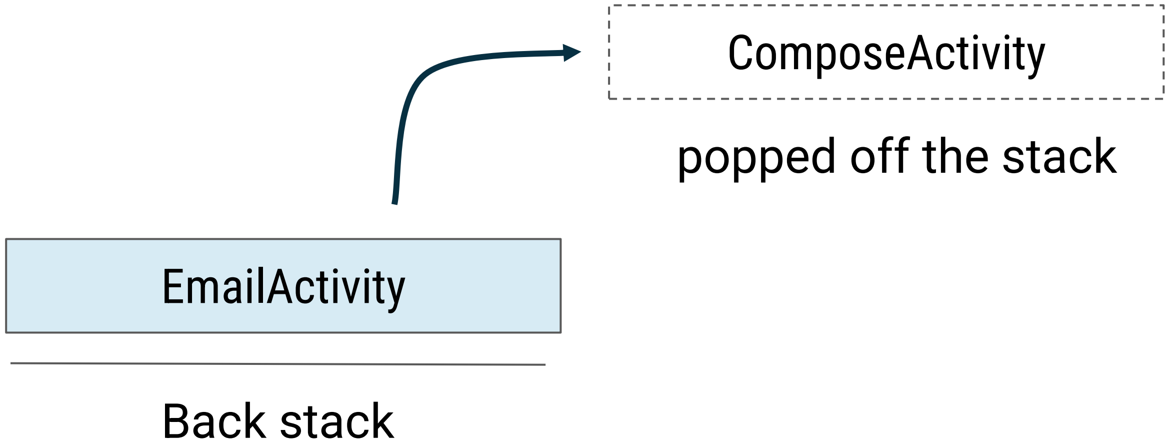
Add to the back stack again



Tap Back button

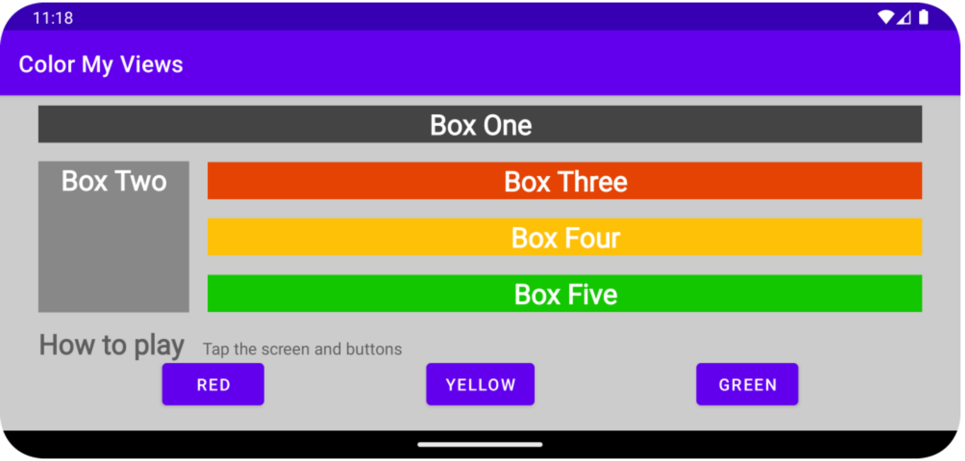


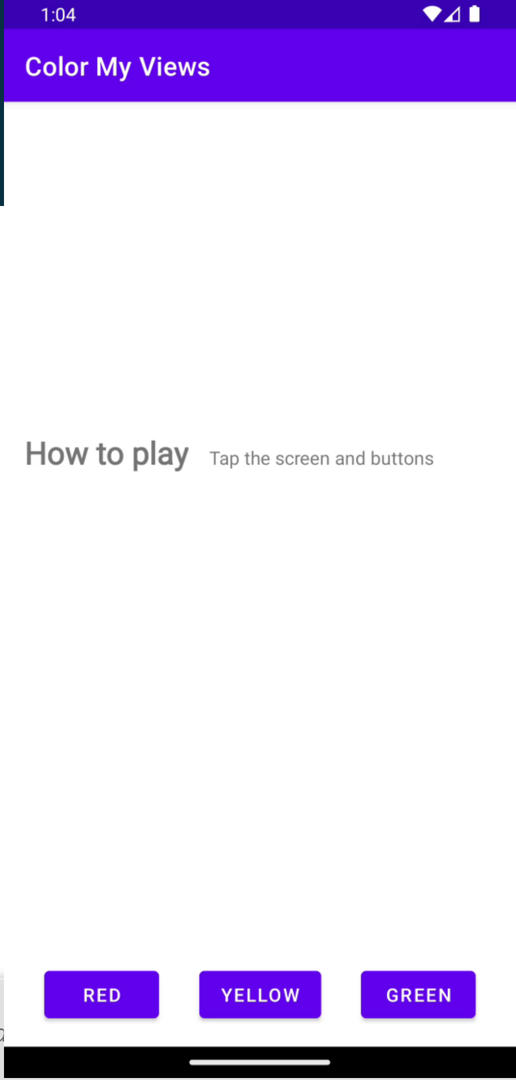
Tap Back button again



Demo

Assignment 1

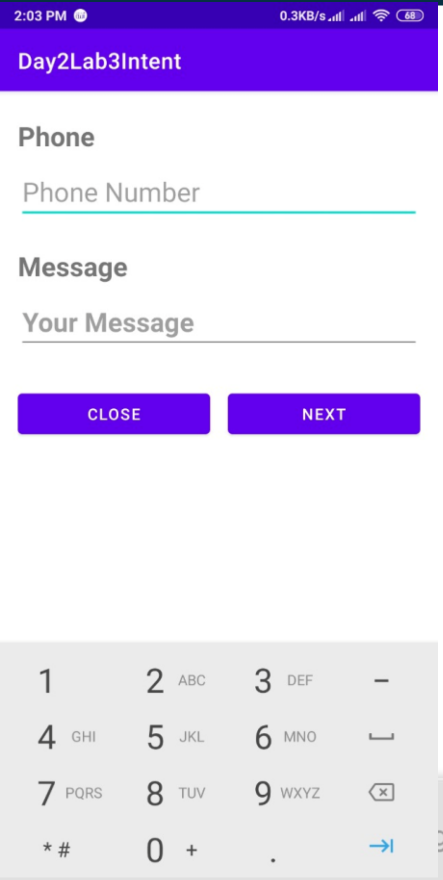




Assignment 2

- Create an Android project with 2 Activities.
- First Activity contains:
  - Two TextView {Mobile Number, Message}
  - Two EditText {MobileNumberValue, MessageValue}
  - Two Buttons {Next, Close}
- Second Activity contains:
  - Two TextView to extract and show {MobileNumberValue, MessageValue} from the intent.
  - One Button{close}

Assignment 2


2:03 PM 0.3KB/s

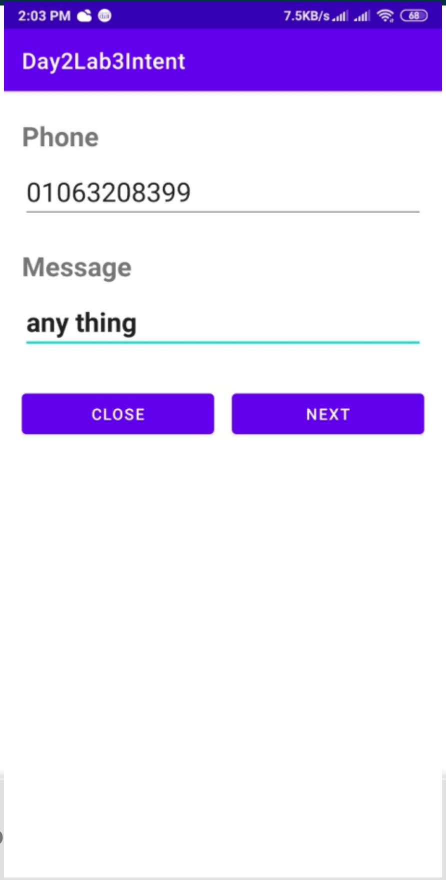
Day2Lab3Intent

Phone

Phone Number

Message

Your Message


2:03 PM 7.5KB/s

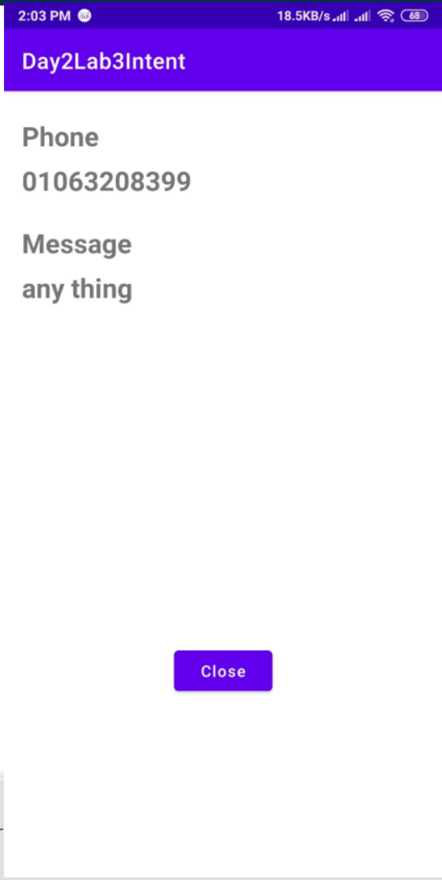
Day2Lab3Intent

Phone

01063208399

Message

any thing


2:03 PM 18.5KB/s

Day2Lab3Intent

Phone

01063208399

Message

any thing

Assignment 3 (Bonus)

- Create a counter application, that displays the number of Mobile rotations ☺
- Hint:
    - Use onSaveInstanceState()
    - Use instance member counter

Day2Lab2Counter
0