Application Development for Mobile Computer

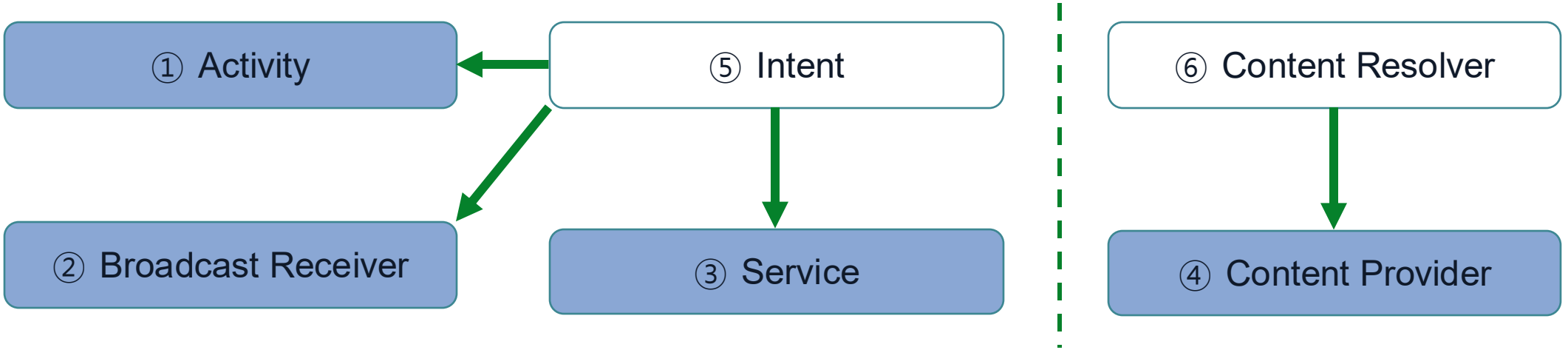
<Week 6-7>

Youn Kyu Lee

Android Components

Android Components

- Android provides four major components
- Each component runs with its own independent lifecycle



① Activity: Component responsible for the screen UI
② Broadcast Receiver: Component that receives messages from the system or user
③ Service: Component responsible for background code processing
④ Content Provider: Component for data sharing between apps
⑤ Intent: A message tool delivered to the system to run components (①–③)
⑥ Content Resolver: A tool for using data provided by the Content Provider

Android Components

Context

- A class that contains information (properties) and tools (methods) for using the system
- Most contexts are created at runtime when components are executed, and the created component provides methods that allow access to these tools
- Types of Context
  - Application Context
  - Base Context
- Functions of Context per Component

	Application	Activity	Serive	Content Provider	Broadcast Receiver
Show a Dialog	No	Yes	No	No	No
Start an Activity	No	Yes	No	No	No
Layout Inflation	No	Yes	No	No	No
Start a Service	Yes	Yes	Yes	Yes	Yes
Bind to a Service	Yes	Yes	Yes	Yes	No
Send a Broadcast	Yes	Yes	Yes	Yes	Yes
Regist BroadcastReceiver	Yes	Yes	Yes	Yes	No
Load Resource Values	Yes	Yes	Yes	Yes	Yes

Android Components

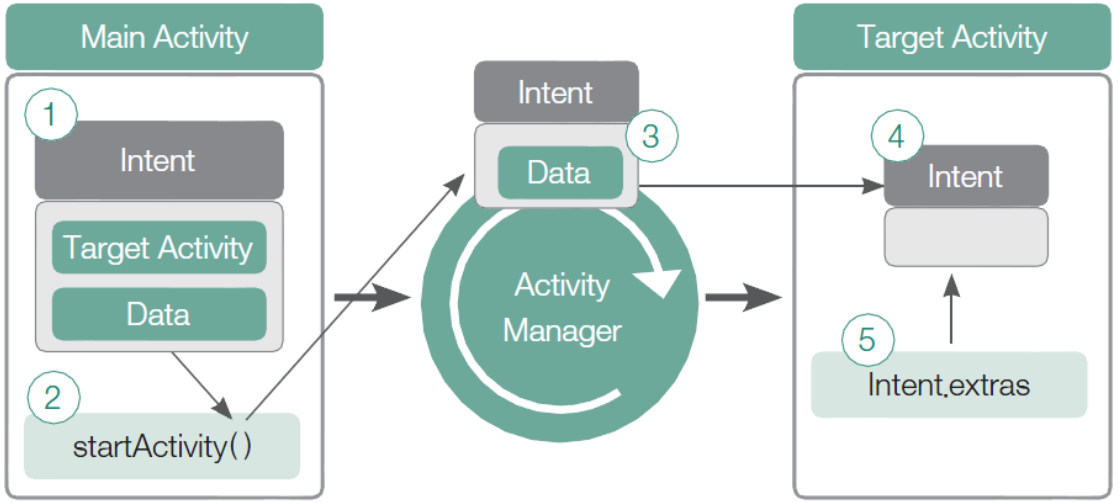
Intent (1)

- A developer places the intended action into an Intent and delivers it to Android, which interprets and executes it
- To run an Activity, an Intent is basically required
- However, the MainActivity created with a new project is automatically registered and executed without special configuration
- To use an Activity other than MainActivity, the new Activity's name must be placed in the Intent and delivered to the system

Android Components

Intent (2)

- Intent Delivery Process
  - ① Create an Intent with the target Activity name and the data to be delivered
  - ② Pass the created Intent to the startActivity() method, which sends it to the Activity Manager
  - ③ The Activity Manager analyzes the Intent and runs the specified Activity
  - ④ The delivered Intent is passed on to the target Activity
  - ⑤ In the target Activity, if the Intent contains data, the data can be retrieved and used



Android Components

Activity Lifecycle (1)

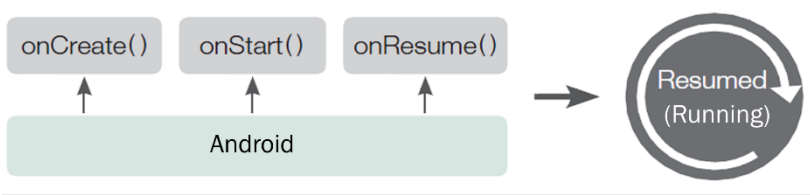
- In Android, whenever there is a state change — such as switching to another Activity, turning off the screen, or closing the app — the system notifies the Activity by calling its lifecycle methods
- Activity lifecycle methods are invoked to reflect these state changes

Method	Activity State	Description
onCreate()	Created	Activity is created (the method where most initialization code is written)
onStart()	Visible	Activity becomes visible on the screen
onResume()	Running	Activity is now in the foreground and interacting with the user. (No separate 'running' method; calling onResume() means running.)
onPause()	Partially hidden	Activity is partially obscured by another Activity
onStop()	Hidden	Another Activity fully covers the screen
onDestroy()	Destroyed	Activity is terminated

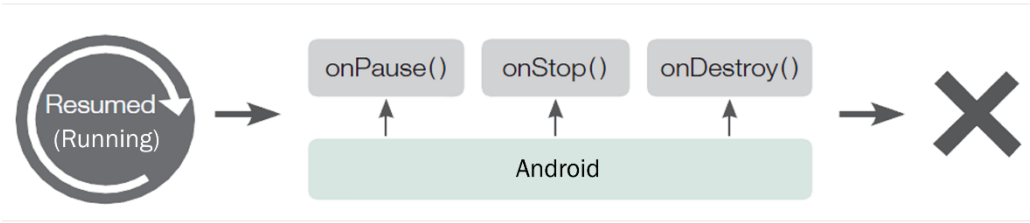
Android Components

Activity Lifecycle (2)

Lifecycle when an Activity is created and shown on the screen

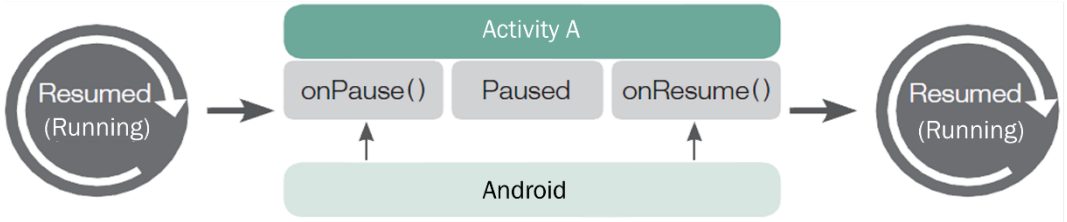


Lifecycle when an Activity is removed from the screen



Current Activity lifecycle when the Activity is created

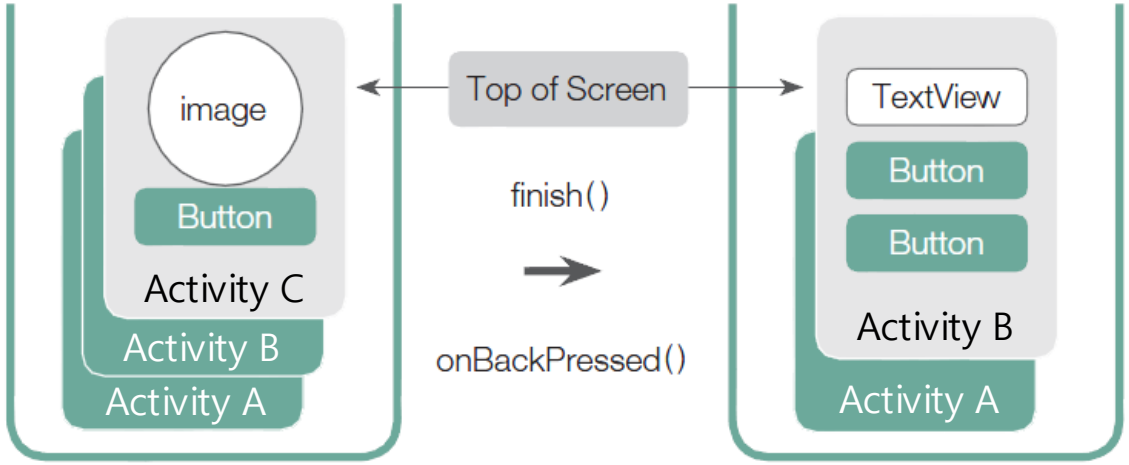
Current Activity lifecycle when a new Activity is created without completely covering the current Activity



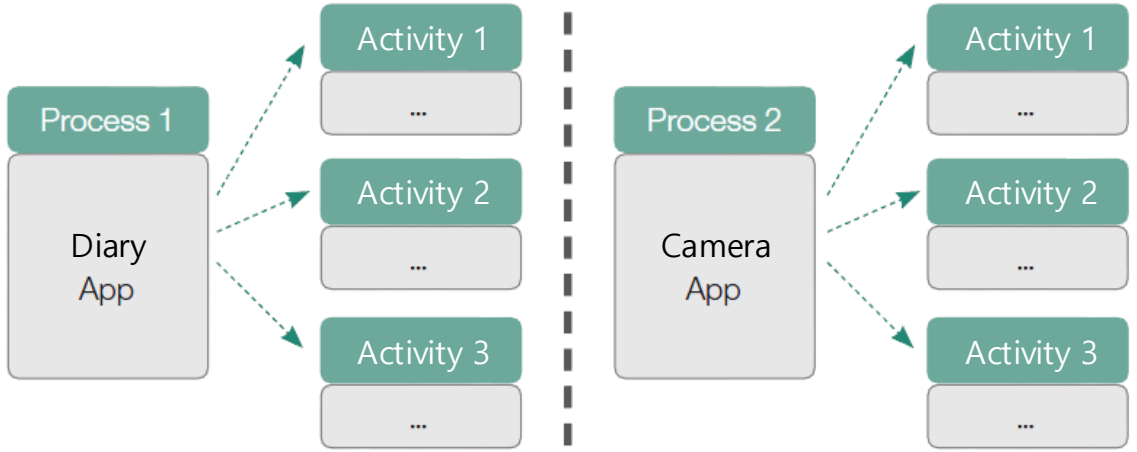
Android Components

Activity Lifecycle (3)

- Activity Back Stack
  - Back Stack is Android's storage space that holds Activities or screen components



- Task and Process
  - Task: a unit of work that manages a Process running in an application



Android Components

Activity Lifecycle (4)

- Managing Activity Tasks
  - Managed via manifest settings

Attribute	Description
launchMode	Determines whether the called Activity is newly created or reused
taskAffinity	Decides the task stack in which the Activity will be placed, depending on allowTaskReparenting
allowTaskReparenting	Allows the Activity to be placed in a task with the same affinity
clearTaskOnLaunch	If true, when relaunched, all Activities except the main Activity are removed
alwaysRetainTaskState	If false (default), system removes all Activities except the main Activity if unused for a certain time
finishOnTaskLaunch	Determines whether to finish existing Activities when the app is reopened

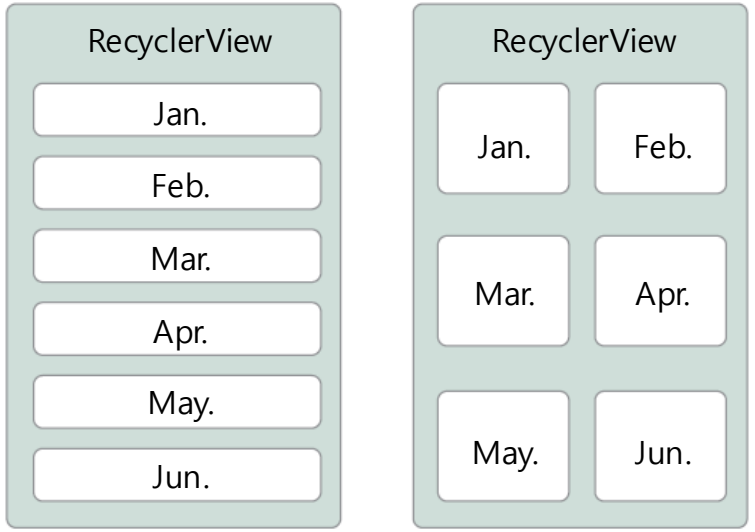
  - Managed via flags

Flag	Description
FLAG_ACTIVITY_CLEAR_TOP	If the called Activity is already in the stack, removes all above it to bring it to the top
FLAG_ACTIVITY_NEW_TASK	Creates a new task and adds the Activity inside it
FLAG_ACTIVITY_MULTIPLE_TASK	Creates a new task with the called Activity as the main Activity
FLAG_ACTIVITY_SINGLE_TOP	If the called Activity is already at the top, it is reused instead of recreated

Android Components

Container: Creating Lists

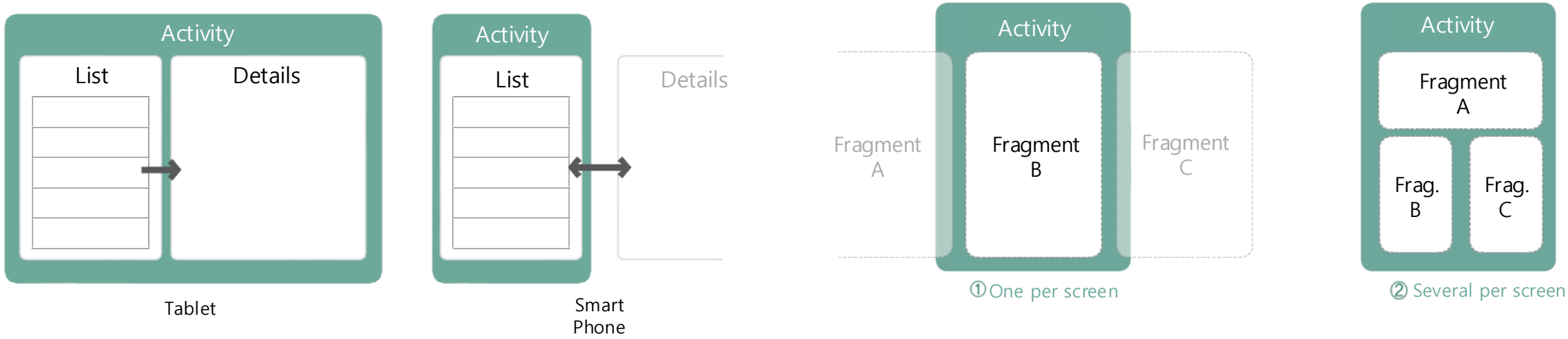
- When displaying data dynamically in a widget or another layout, a container is used.
  - A container is used for repeatedly displaying data.
  - RecyclerView is the representative container used to display a list on the screen.
  - From Android Studio 3.1, RecyclerView is used.
  - The previously used ListView and GridView perform the same functions but are now moved to the Legacy category.



Android Components

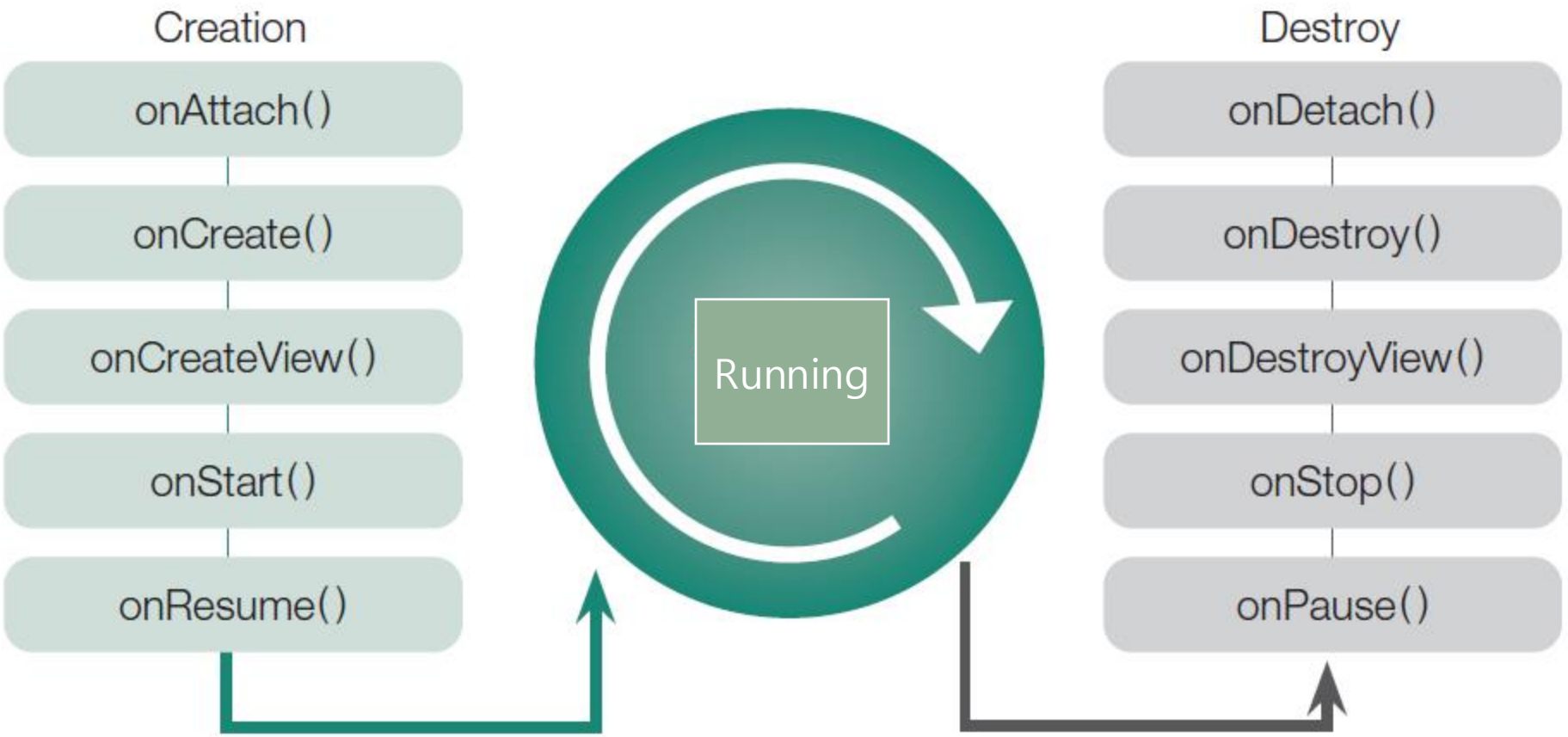
Handling Fragments in Activities

- Activity: The basic unit for representing a screen.
- Fragment: Designed to allow a single Activity to compose different layouts for devices with different screen sizes (tablet, smartphone, etc.).



Android Components

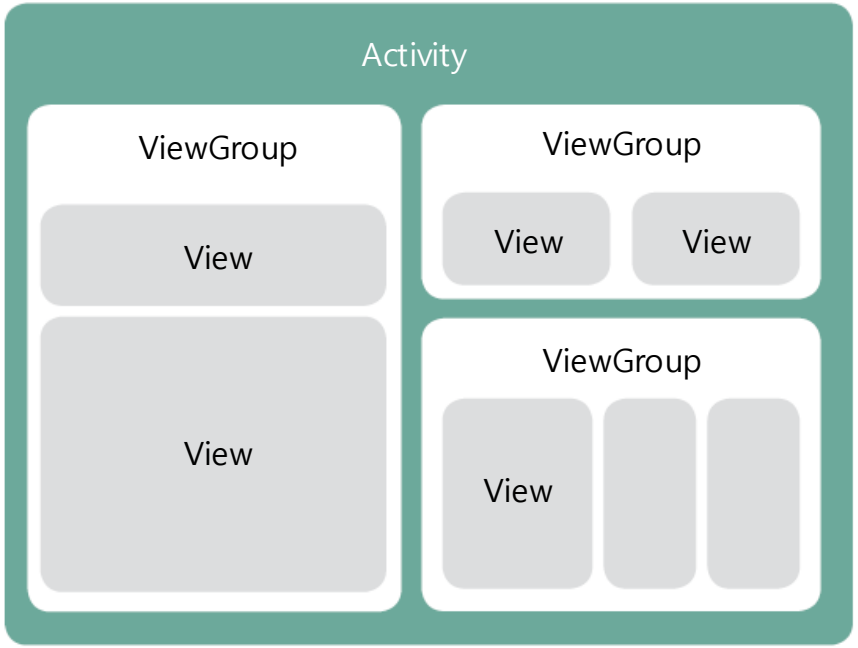
Managing Fragment Lifecycle (1)



Android View

View

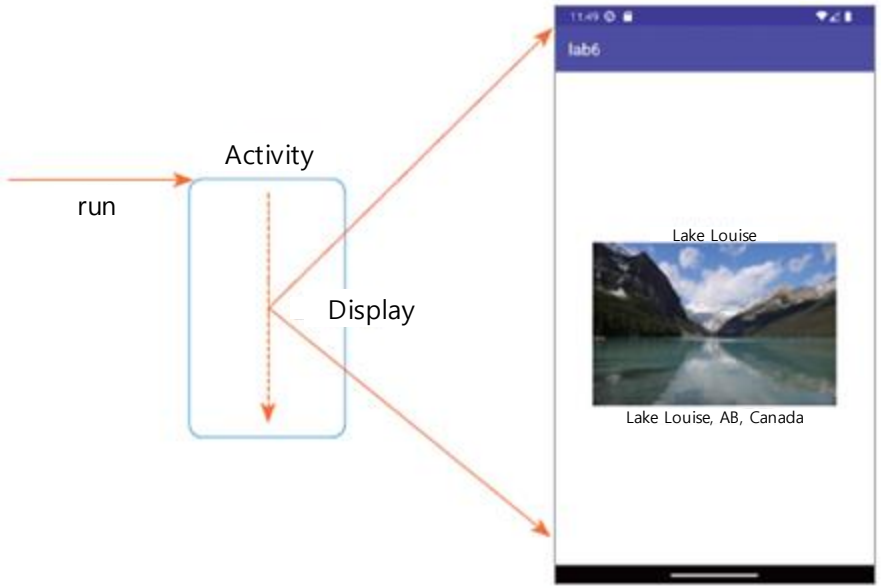
- View: The smallest unit component that composes the screen
  - Hierarchy: App > Activity > (Fragment) > ViewGroup > View
    - ViewGroup: A layout
    - View: Everything in the UI editor's palette

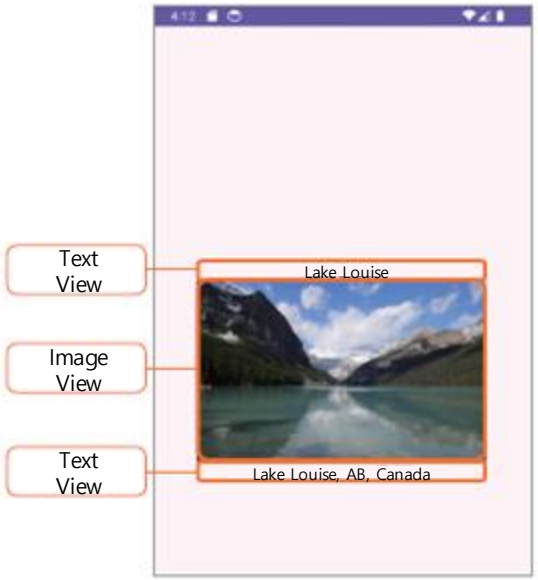


Android View

Activity–View Structure

- Activity is the component that displays the screen
- To show content on the screen, View class is used

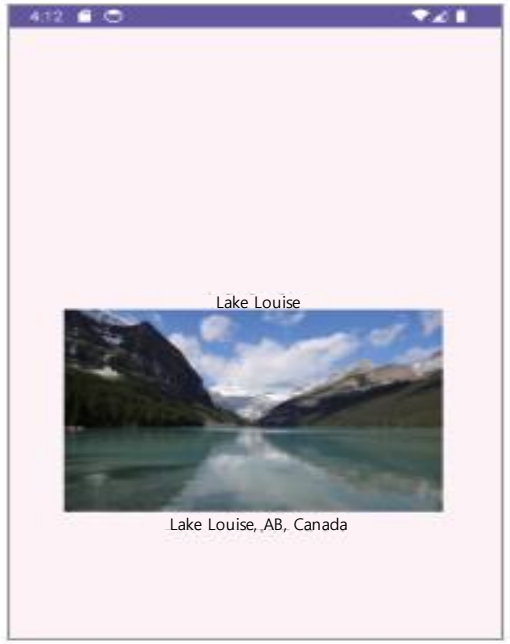




Android View

Building Screens with Activity Code

- Create the View classes that compose the screen directly in the Activity code




```
// Creating a TextView to display name string
val name = TextView(this).apply {
    typeface = Typeface.DEFAULT_BOLD
    text = "Lake Louise"
}
// Creating an ImageView to display an image
val image = ImageView(this).also {
    it.setImageDrawable(ContextCompat.getDrawable(this, R.drawable.lake_1))
}
// Creating an TextView to display address string
val address = TextView(this).apply {
    typeface = Typeface.DEFAULT_BOLD
    text =  "Lake Louise, AB, Canada"
}
val layout = LinearLayout(this).apply {
    orientation = LinearLayout.VERTICAL
    gravity = Gravity.CENTER
    // Adding TextView, ImageView, and TextView objects to a LinearLayout
    addView(name, WRAP_CONTENT, WRAP_CONTENT)
    addView(image, WRAP_CONTENT, WRAP_CONTENT)
    addView(address, WRAP_CONTENT, WRAP_CONTENT)
}
// The LinearLayout is shown on the screen
setContentView(layout)
```

Android View

Building Screens with Layout XML

- A method of composing the screen by defining Views as XML tags

• Defining the Layout for an Activity

```
class MainActivity: AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // Define view as XML
        setContentView(R.layout.activity_main)
    }
}
```

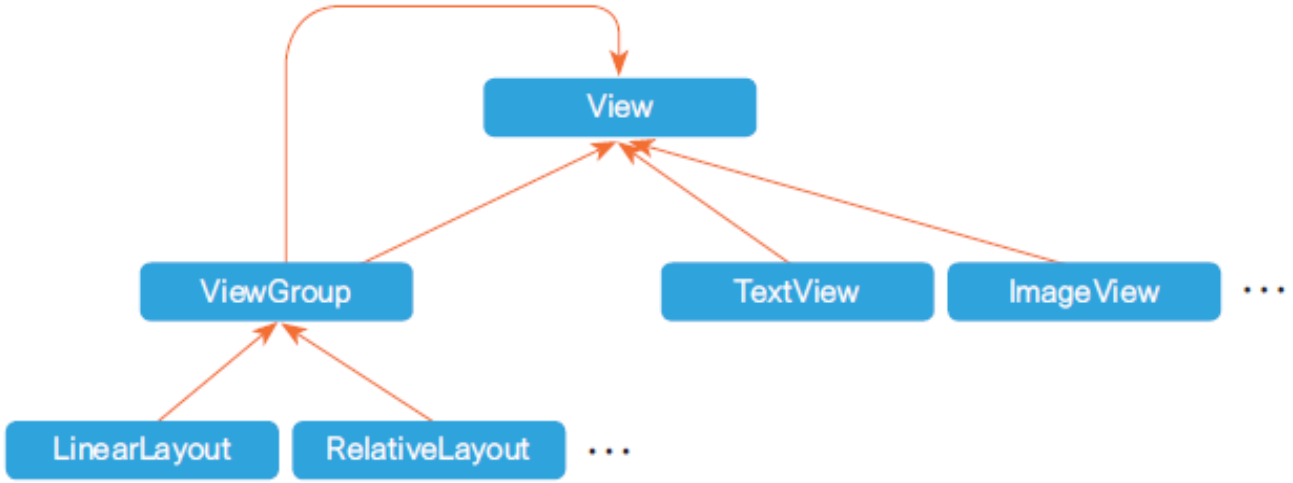
• Designing the Layout in XML (activity_main.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:text="Lake Louise" />
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/lake_1" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:text="Lake Louise, AB, Canada">
</LinearLayout>
```

Android View

Basic Structure of View Classes (1)

- View Object Hierarchy
  - View: The top-level class of all view classes. Only subclasses of View can be displayed on an Activity.
  - ViewGroup: Does not have its own UI, used to group and control multiple Views.
  - TextView: A class used to display specific UI elements.



Android View

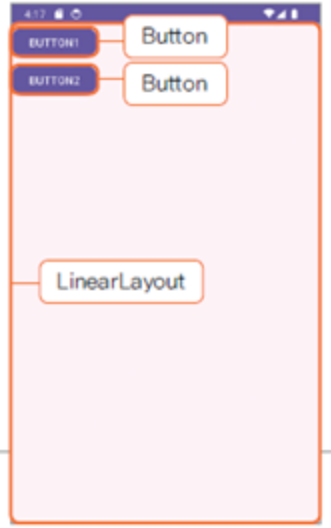
Basic Structure of View Classes (2)

- Layout classes, which are subclasses of ViewGroup, are not for displaying the screen itself.
- Instead, they are used to contain and collectively control multiple view objects (e.g., TextView, ImageView).

• Including a View in a Layout Class

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BUTTON1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BUTTON2" />
</LinearLayout>
```

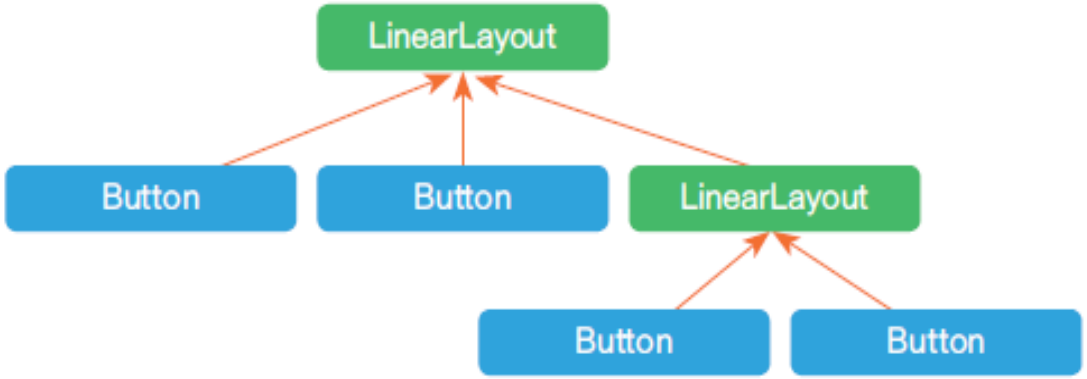
▶ Execution Result



Android View

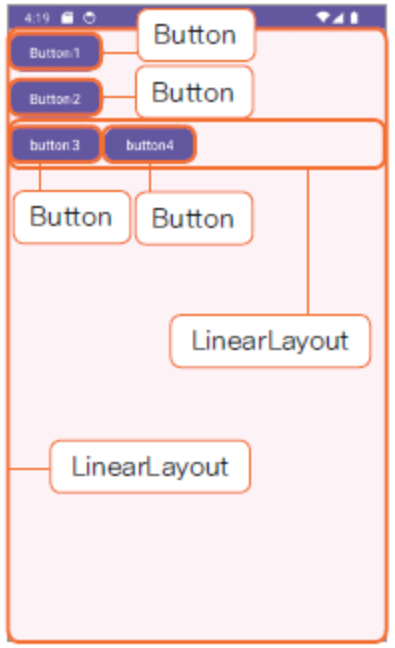
Basic Structure of View Classes (3)

- Layout nesting: The hierarchy of Views can be built in a complex way by nesting layout objects.



• Layout Nesting

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BUTTON1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BUTTON2" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="BUTTON3" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="BUTTON4" />
    </LinearLayout>
</LinearLayout>
```



Android View

Using Views from Layout XML in Code

- The id attribute is used to assign an identifier for an object
- When an id is added in XML, it is automatically added as a constant variable in the R.java file
- In code, the object can be obtained using the findViewById() function

• Assigning an ID attribute

```
<TextView
    android:id="@+id/text1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="hello" />
```

• Accessing Views from XML in Code

```
// Display the XML layout
setContentView(R.layout.activity_main)
// Get the view object by its ID
val textView1: TextView = findViewById(R.id.text1)
```

Display the activity screen by creating view objects from XML

• A View object retrieved using generics

```
// XML layout displayed, view objects created.
setContentView(R.layout.activity_main)
// Get the view object by its ID
val textView1 = findViewById<TextView>(R.id.text1)
```

Android View

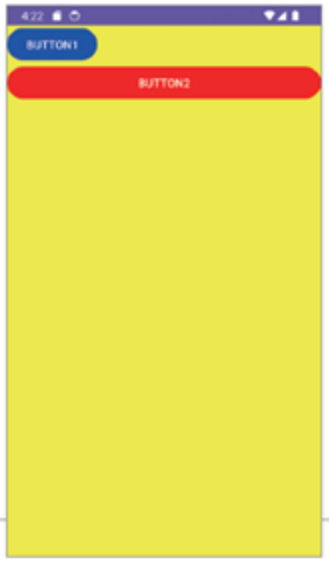
Setting View Size

- The size of a view is essential information for how it should appear on the screen
- Attributes used to set size: layout_width, layout_height
  - Numeric value (e.g., 100dp)
  - match_parent
  - wrap_content

• Example of size setting

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#ffff00">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BUTTON1"
        android:backgroundTint="#0000ff" />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="BUTTON2"
        android:backgroundTint="#ff0000" />
</LinearLayout>
```

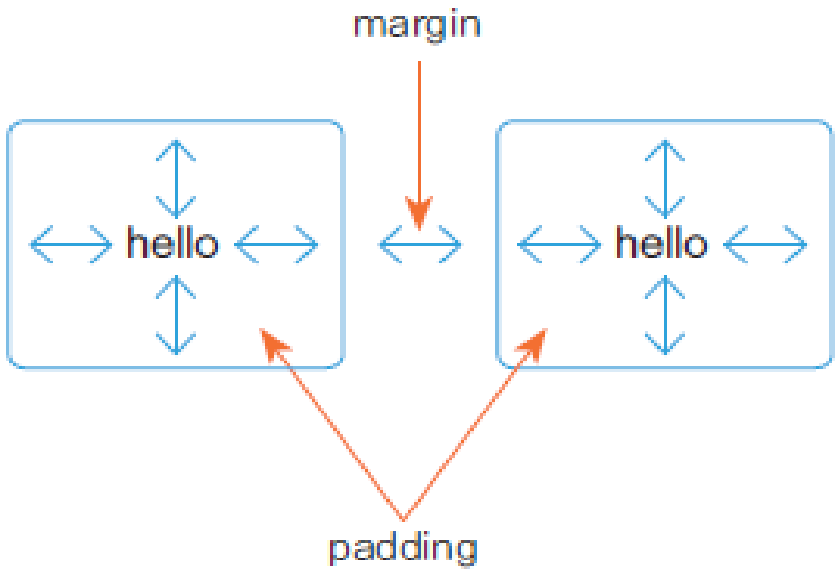
▶ Execution Result



Android View

Setting View Spacing (1)

- View spacing is set using the margin and padding attributes
- With margin and padding, spacing can be set equally in all four directions
- Specific attributes can also be used: paddingLeft, paddingRight, paddingTop, paddingBottom
- layout_marginLeft, layout_marginRight, layout_marginTop, layout_marginBottom



Android View

• Setting View Spacing

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BUTTON1"
        android:backgroundTint="#0000ff"
        android:padding="30dp" />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="BUTTON2"
        android:backgroundTint="#ff0000"
        android:paddingBottom="50dp"
        android:layout_marginLeft="50dp" />
</LinearLayout>
```

▶ Execution Result

Android View

Setting View Visibility (1)

- The visibility attribute controls whether a view should be displayed on the screen
- Options: visible, invisible, gone
- invisible: the view is not visible, but still takes up space
- gone: the view is not visible and does not take up any space

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="BUTTON2"
    android:visibility="invisible" />
```

Setting View Visibility (2)

- To adjust a view's visibility in code, set its value to View.VISIBLE or View.INVISIBLE

• Changing the Visibility Property in Code

```
visibleBtn.setOnClickListener {
    targetView.visibility = View.VISIBLE
}
invisibleBtn.setOnClickListener {
    targetView.visibility = View.INVISIBLE
}
```

Android View

TextView (1)

- TextView is a view that displays strings on the screen
- android:text : Specifies the string to display in the TextView
  - android:text="helloworld"
  - android:text="@string/hello"
- android:textColor : Specifies the color of the string
  - android:textColor="#FF0000"
- android:textSize : Specifies the size of the string
  - android:textSize="20sp"
- android:textStyle : Specifies the style of the string
  - android:textStyle="bold"
  - Options: bold, italic, normal

• TextView Property

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="helloworld"
    android:textColor="#FF0000"
    android:textSize="20sp"
    android:textStyle="bold" />
```

▶ Execution Result

Android View

TextView (2)

- android:autoLink : Analyzes the displayed string and automatically adds links for specific patterns
  - Example: android:autoLink="web"
  - Options: web, phone, email, etc.

• AutoLink Property

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="EasysPublishing – Web Page : http://easyspub.com, Phone Number : 325- 1722,
    E-mail : easy@easypub.co.kr"
    android:autoLink="web|email|phone" />
```

▶ Execution Result

Android View

TextView (3)

- android:maxLines attribute: Limits the string to a specified number of lines
  - Example: android:maxLines="3"
- android:ellipsize attribute: Adds an ellipsis (...) to indicate that more text exists
  - Options: end, middle, start

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/long_text"
    android:singleLine="true"
    android:ellipsize="middle" />
<View
    android:layout_width="match_parent"
    android:layout_height="2dp"
    android:background="#000000" />
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/long_text"
    android:maxLines="3"
    android:ellipsize="end" />
```

Android View

ImageView

- A view used to display images on the screen
- android:src : Sets the image to display
  - Example: android:src="@drawable/image3"
- android:maxWidth, android:maxHeight, android:adjustViewBounds : Define the maximum size of the image
  - maxWidth and maxHeight are used together with adjustViewBounds
  - If adjustViewBounds="true", the view size is adjusted proportionally to the image's width and height

• Matching the View size to the image size

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:maxWidth="100dp"
    android:maxHeight="100dp"
    android:adjustViewBounds="true"
    android:src="@drawable/test"
    android:background="#0000ff"/>
```

▶ Execution Result

Android View

Button, CheckBox, RadioButton

- Button: A view used to handle user events
- CheckBox: A view that allows multiple selections
- RadioButton: A view that allows single selection

```
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="check1" />
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="check2" />
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="radio1" />
    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="radio2" />
```

Execution Result

Android View

EditText

- A view that allows text input
- android:lines, android:maxLines
  - android:lines: Displays the input area with multiple lines from the beginning
  - android:maxLines: Initially displays as a single-line input, but expands up to the specified number of lines
- android:inputType
  - Specifies the keyboard type that appears during text input
  - Example: android:inputType="phone"

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="phone" />
```

Android View

View Binding (1)

- View Binding is a method to easily use view objects declared in a layout XML file within code
- Provides a way to access registered view objects in the layout XML without using findViewById() in the Activity

• Set view binding in the gradle file.

```
android {
    viewBinding.isEnabled = true
}
```

Android View

View Binding (2)

- A class containing the view objects registered in the layout XML file is automatically generated
- The name of the generated class is based on the layout XML filename
  - The first letter is capitalized
  - Underscores (_) are removed, and the following word's first letter is capitalized
  - The suffix Binding is added
    - activity_main.xml → ActivityMainBinding
    - item_main.xml → ItemMainBinding
- Call the inflate() function of the generated class to obtain a binding object
- For displaying the Activity screen, pass binding.root to the setContentView() function

Q & A

aiclasscau@gmail.com