Mobile Programming

Fragment Basics

Agenda

- Fragment
- ViewPager2
- Tab Layout

Recap: Null Safety (1/2)

- NPE (NullPointerException)
  - One of the most common pitfalls in many programming languages (e.g., Java!)
  - Accessing a member of a null reference will result in a null reference exception

- Kotlin's type system is aimed at eliminating the danger of null references
  - Kotlin type system distinguishes between *references that can hold null* (nullable references) and *those that cannot* (non-null references)

Recap: Null Safety (2/2)

- Safe calls
  - Second option for accessing a property on a nullable variable is using the safe call operator “?.”

```
val a = "null"
val b: String? = "string"
Log.d("ITM","${(b?.length)}")
Log.d("ITM","${(a?.length)}") // Unnecessary safe call
```

  - This returns b.length if b is not null, and null otherwise!

Lateinit (1/2)

- Use of nullable properties
  - Too many safe-calls!

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

        Person().process()

    }
}

class Person{
    var name:String? = null
    init{
        name = "Lionel"
    }

    fun process(){
        name?.plus(" Messi")
        Log.d("ITM","length = ${name?.length}")
        Log.d("ITM","first char = ${name?.substring(0,1)}")
    }
}
```

Lateinit (2/2)

- To avoid such null checks, use lateinit
  - Late initialization
- Note
  - Can be used on *var* properties declared inside the body of a class, both top-level properties and local variables
  - Variable must be non-null
    - NPE error occurs if not initialized before access
  - Variable must not be a primitive type

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

        Person().process()

    }
}

class Person{
    lateinit var name:String
    init{
        name = "Lionel"
    }
    fun process(){
        name.plus(" Messi")
        Log.d("ITM","length = ${name.length}")
        Log.d("ITM","first char = ${name.substring(0,1)}")
    }
}
```

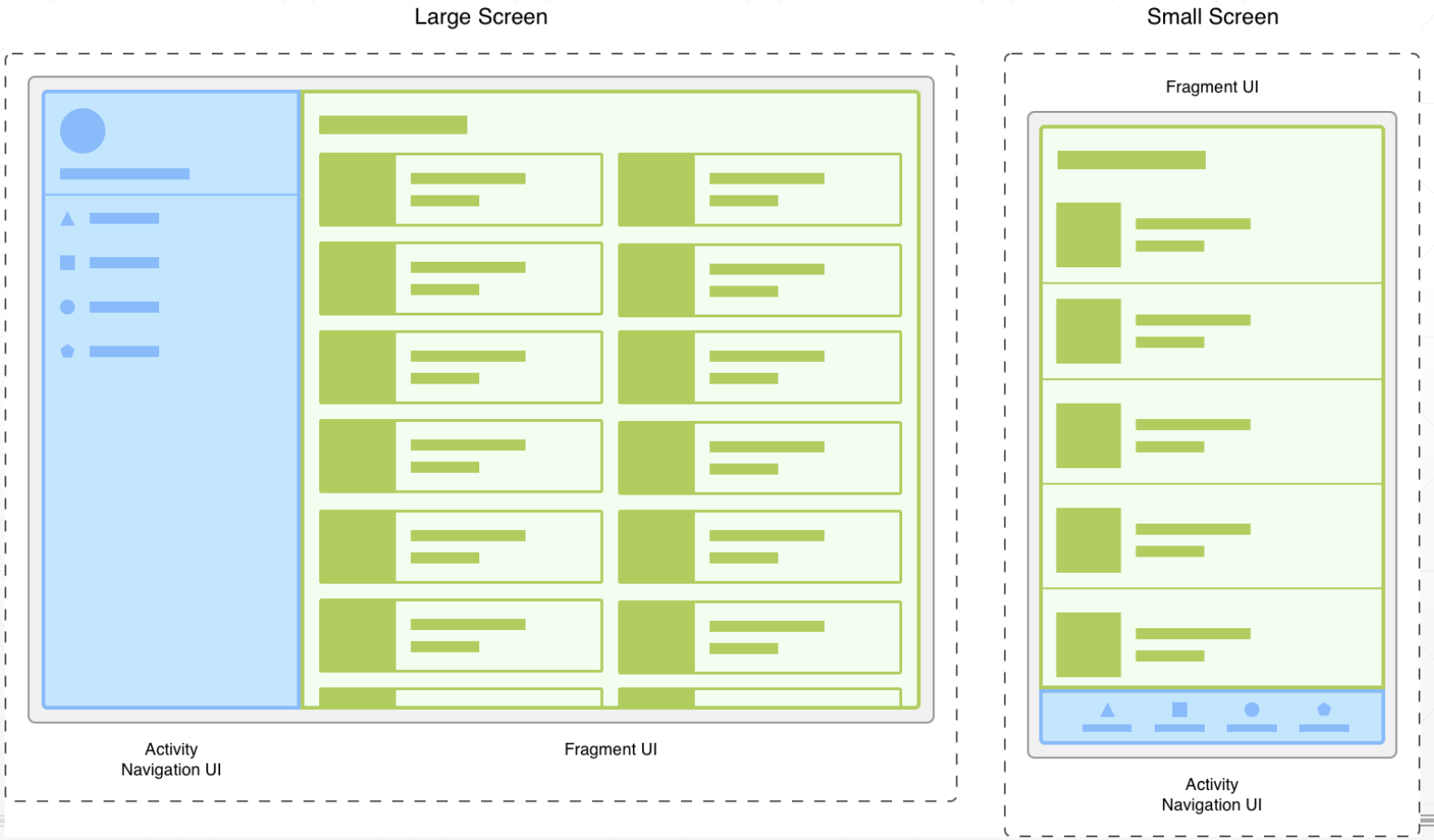
Lazy

- Late initialization with *val*
  - Declaration of *val* first, and followed by “by lazy {*initialization logic*}”
  - Characteristics
    - Declaration and initialization codes are written together
    - Initialized using the logic in “by lazy {}”lazy, at the moment of the first access to the variable

```
class MainActivity : AppCompatActivity() {
    val binding by lazy { ActivityMainBinding.inflate(layoutInflater)}

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(binding.root)

        // ...

    }
}
```

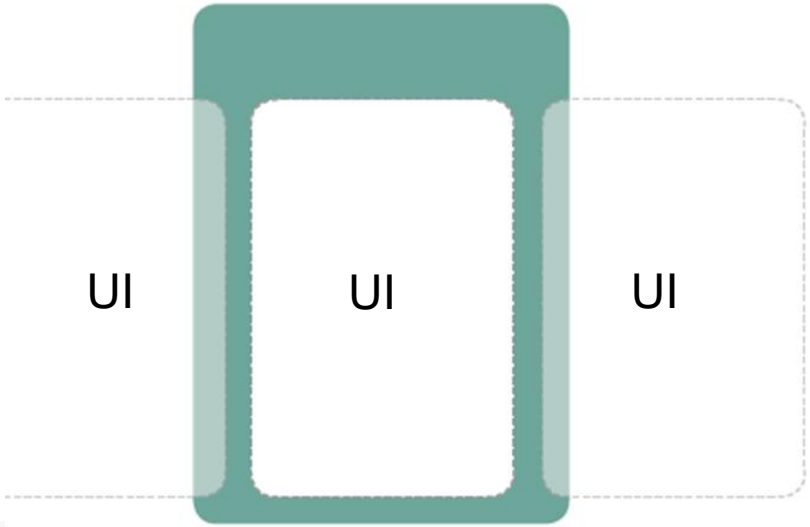
Fragment

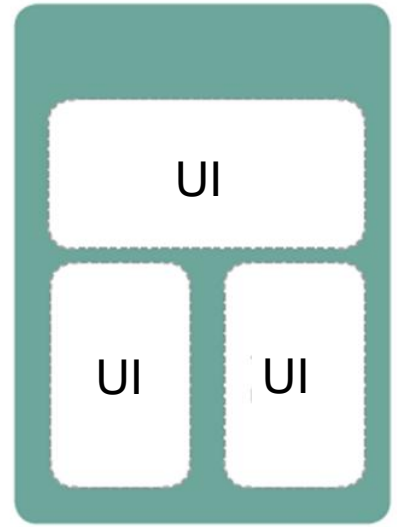
- A single activity with too many roles?
  - e.g. Navigation element + detail view?



Fragment

- A single activity with too many roles?
  - e.g. Navigation element + detail view?
  - Managing all of these variations in the activity can be unwieldy!





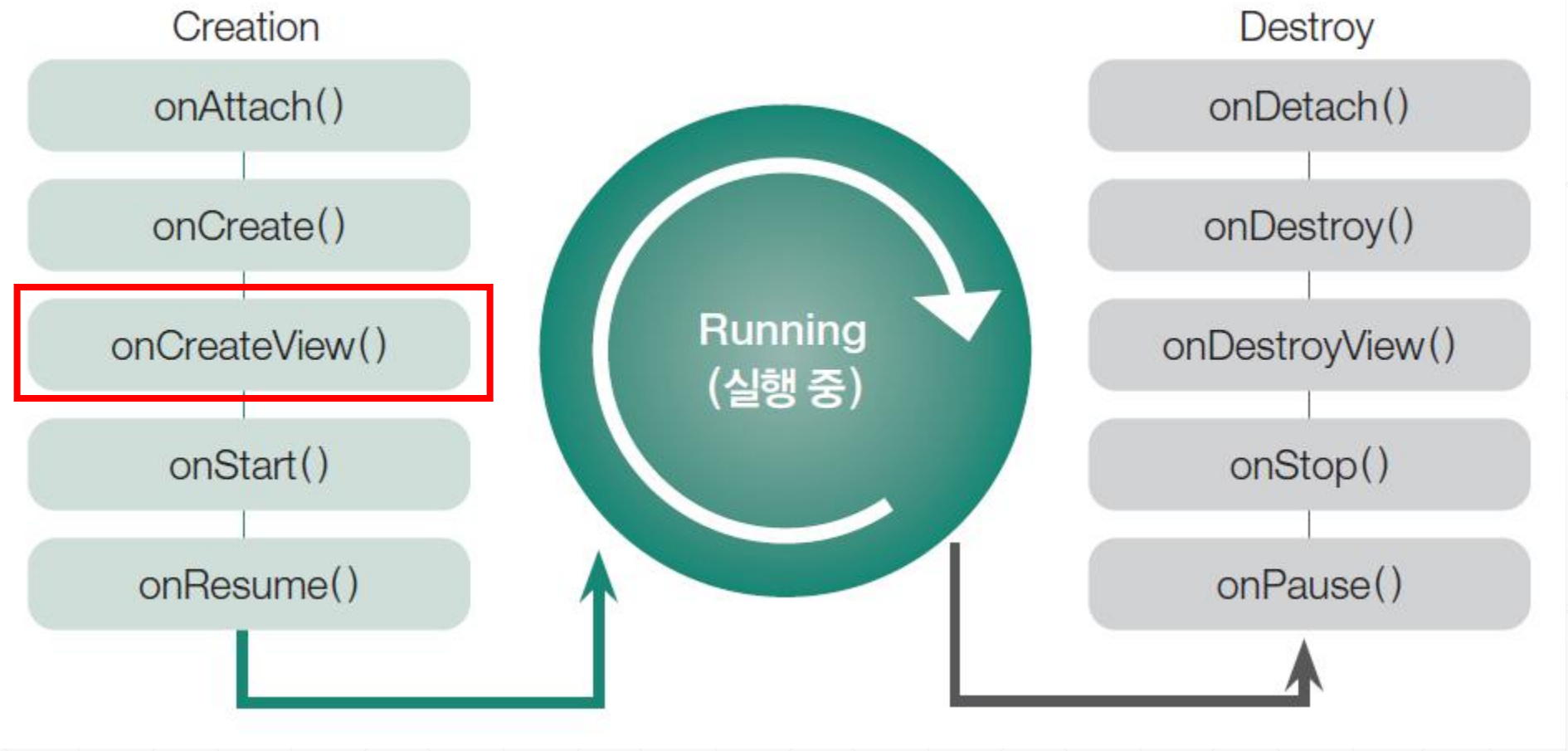
Fragment

- Represents a reusable portion of your app's UI
  - Defines and manages its own layout, has its own lifecycle, and can handle its own input events (like Activity!)
  - Most codes working with Activity also work with Fragment
  - Cannot live on their own! fragment must be hosted by an activity or another fragment

- Introduces modularity and reusability into your activity's UI by dividing the UI into discrete chunks

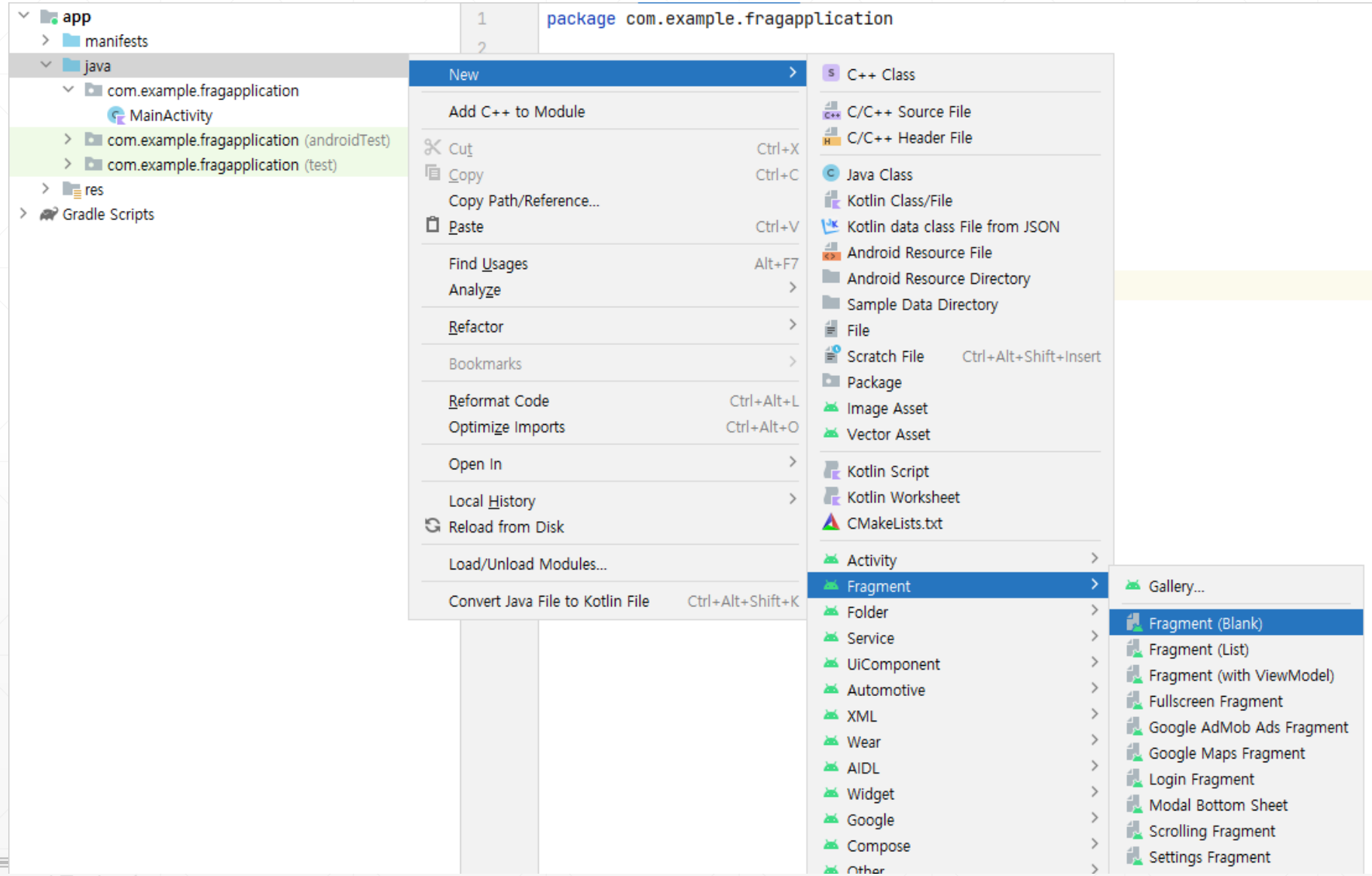
Fragment

- Lifecycle



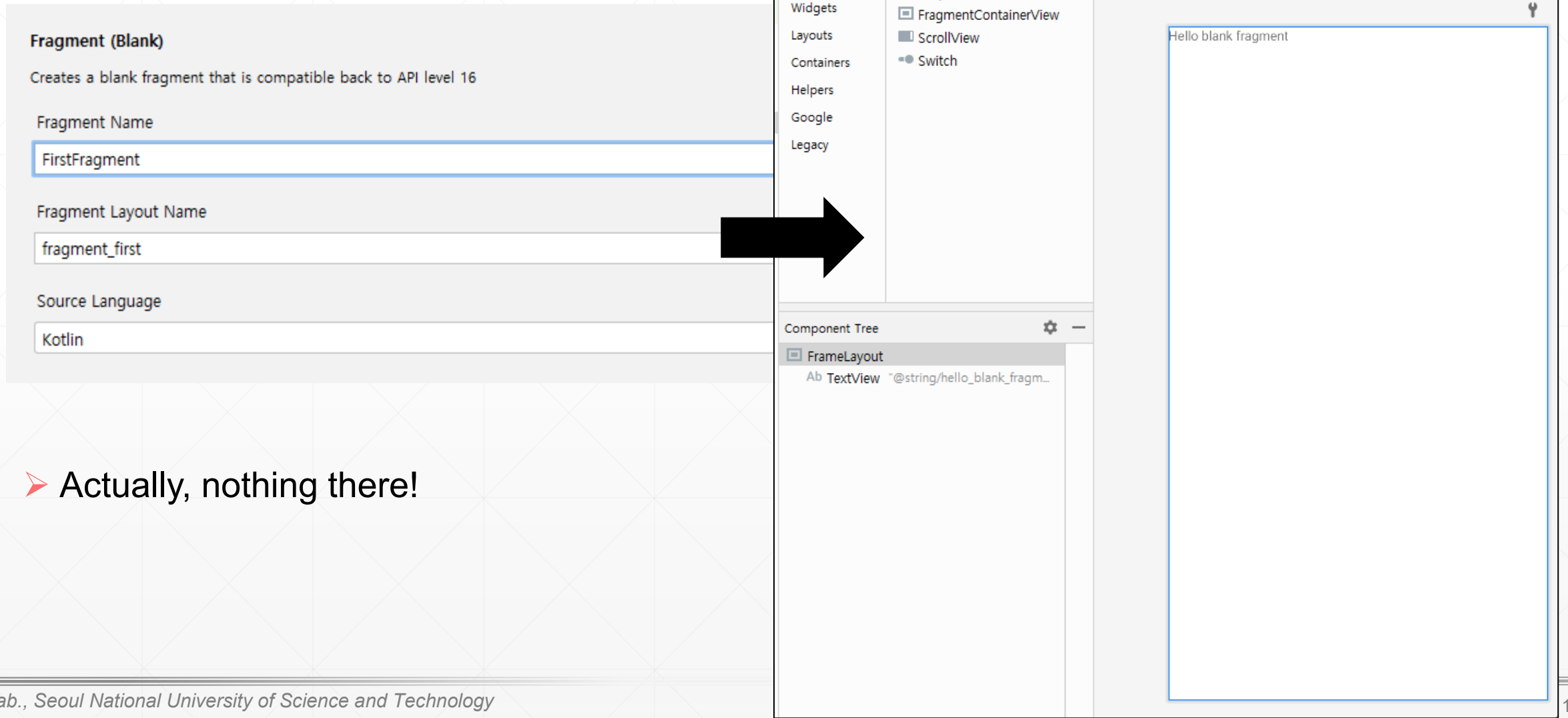
Adding a Single Fragment: Fragment

- New → fragment



Adding a Single Fragment : Fragment

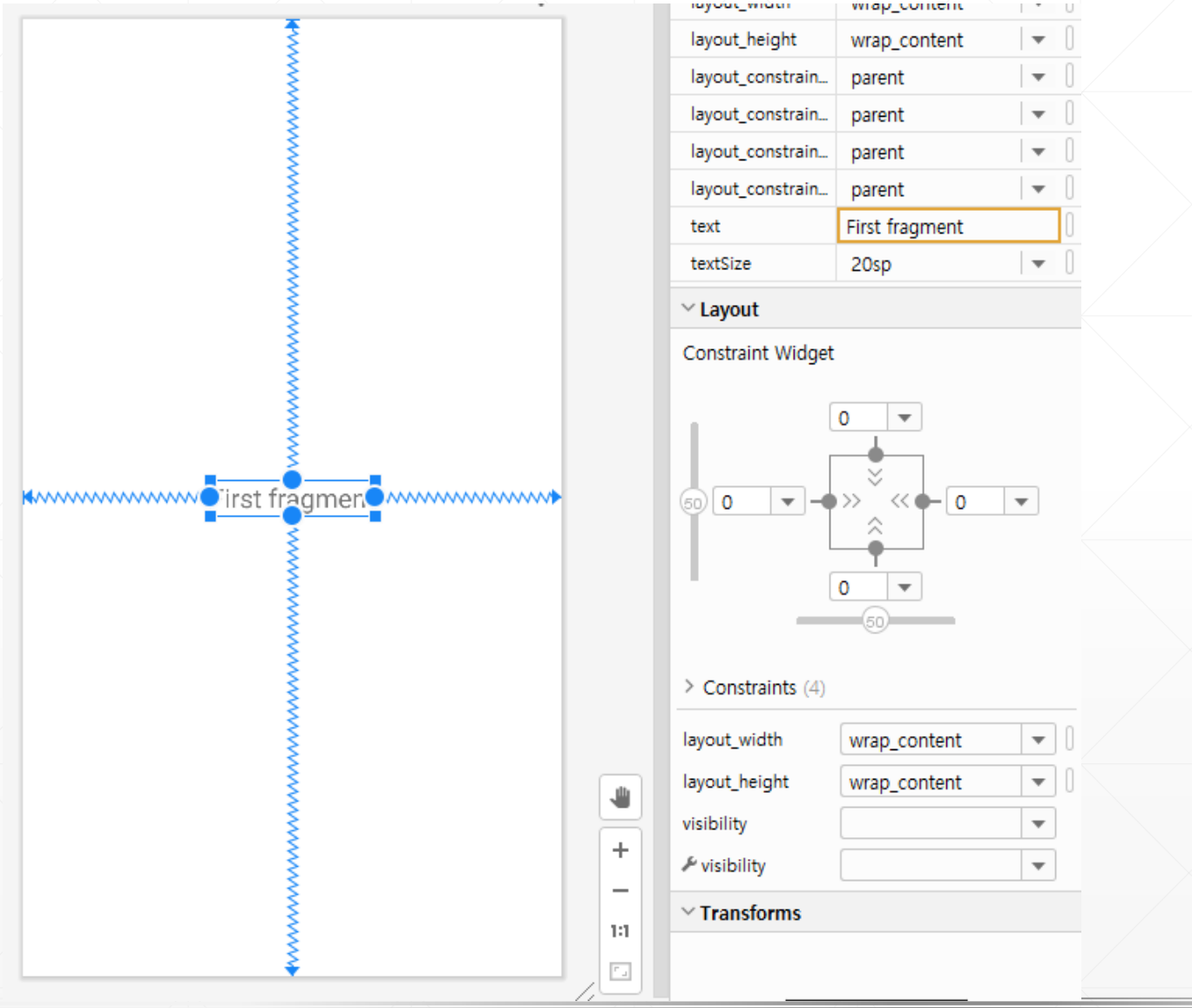
- Name it!



  - Actually, nothing there!

Adding a Single Fragment : Fragment

- Layout design for Fragment
  - FrameLayout → constraint layout
    - Use TextEditor!
  - Re-design the textView



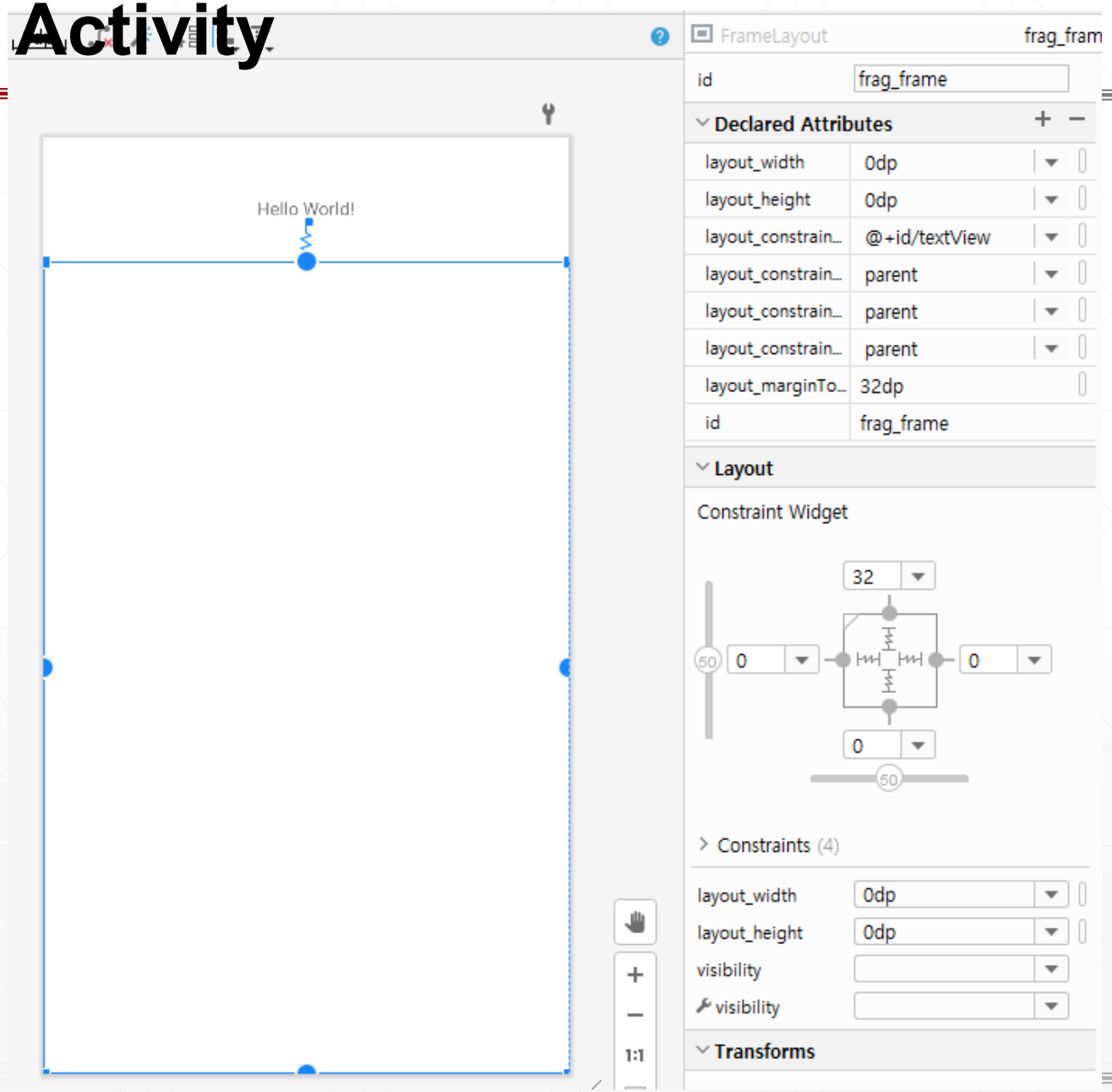
Adding a Single Fragment : Fragment

- What's in the code?
  - viewBinding setting
    - Lateinit declaration for a binding variable
  - onCreateView callback
    - Late initialization of the binding variable
    - Use viewBinding pattern

```
class FirstFragment : Fragment() {
    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null
    lateinit var binding: FragmentFirstBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        arguments?.let {
            param1 = it.getString(ARG_PARAM1)
            param2 = it.getString(ARG_PARAM2)
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        binding = FragmentFirstBinding.inflate(inflater, container, false)
        Log.d("ITM",param1+param2)
        return binding.root
    }
…
```

Adding a Single Fragment: Activity

- Activity
  - Add a container for Fragment
    - FrameLayout is a common choice



Adding a Single Fragment: Activity

- Add a fragment into the container!

- FragmentManager
  - Class responsible for performing actions on your app's fragments, such as adding, removing, or replacing them, and adding them to the back stack!

- FragmentTransaction
  - Each set of fragment changes that you commit
    - add(), remove(), replace()
    - The final call on each FragmentTransaction must commit the transaction
    - More details: https://developer.android.com/guide/fragments/transactions

Adding a Single Fragment: Activity

- FragmentManager

Kotlin Java

```
val fragmentManager = supportFragmentManager
val fragmentTransaction = fragmentManager.beginTransaction()
```

- FragmentTransaction

Kotlin Java

```
val fragmentManager = ...
// The fragment-ktx module provides a commit block that automatically
// calls beginTransaction and commit for you.
fragmentManager.commit {
    // Add operations here
}
```

```
41
42  dependencies {
43      implementation("androidx.fragment:fragment-ktx:1.8.9")
44      implementation(libs.androidx.core.ktx)
45      implementation(libs.androidx.appcompat)
46      implementation(libs.material)
47      implementation(libs.androidx.activity)
48      implementation(libs.androidx.constraintlayout)
49      testImplementation(libs.junit)
50      androidTestImplementation(libs.androidx.junit)
51      androidTestImplementation(libs.androidx.espresso.core)
52  }
```

Groovy Kotlin

```
dependencies {
    implementation("androidx.fragment:fragment-ktx:1.5.3")
}
```

Use the latest version (1.8.9)

Adding a Single Fragment: Activity

- Adding a fragment using FragmentManager & FragmentTransaction

```
class MainActivity : AppCompatActivity() {

    val binding by lazy {ActivityMainBinding.inflate(layoutInflater)}

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(binding.root)

        var listFragment = FirstFragment()

        val fManager = supportFragmentManager
        val transaction = fManager.beginTransaction()

        transaction.add(binding.fragFrame.id, listFragment)
        transaction.commit()
    }
}
```

Instantiate a fragment

Get manger and begin a transaction

Add changes, do commit!

Adding a Single Fragment : Activity

- Adding a fragment using FragmentManager & FragmentTransaction
  - Equivalent form

```
class MainActivity : AppCompatActivity() {

    val binding by lazy {ActivityMainBinding.inflate(layoutInflater)}

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(binding.root)

        var listFragment = FirstFragment()

        val fManager = supportFragmentManager
        fManager.commit {
            add(binding.fragFrame.id,listFragment)
        }

    }
}
```

→ Instantiate a fragment

→ Add changes, do commit!

Adding a Single Fragment : Activity

- Adding a fragment with passing values
  - Use arguments of Fragment with Bundle object
  - Bundle
    - Key-value mapping for storing data
    - put* ←→ get* methods

- Step
  - Fragment implements newInstance() method to return a new Fragment instance with parameters stored into arguments using Bundle interface
  - Activity instantiates a Fragment with a set of parameters using newInstance() method
  - Fragment can take the parameters from arguments using Bundle interface

Adding a Single Fragment : Fragment

- Adding a fragment with passing values
  - Class properties
  - onCreate()
    - Take parameters from arguments
    - with Bundle interface
  - onCreateView()
    - Updates the view using parameter values

```
class FirstFragment : Fragment() {

    private var param1: String? = null
    private var param2: String? = null
    lateinit var binding: FragmentFirstBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        arguments?.let {
            param1 = it.getString(ARG_PARAM1)
            param2 = it.getString(ARG_PARAM2)
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        binding = FragmentFirstBinding.inflate(inflater, container, false)
        binding.fragText.text = param1+param2

        return binding.root
    }
```

Adding a Single Fragment : Fragment

- Adding a fragment with passing values
  - Companion object
    - Factory pattern to create a new instance
  - newInstance()
    - Put the values into arguments
    - with Bundle interface
  - The result is FirstFragment instance with the specified parameters: param1 and param2

```
companion object {
    /**
     * Use this factory method to create a new instance of
     * this fragment using the provided parameters.
     *
     * @param param1 Parameter 1.
     * @param param2 Parameter 2.
     * @return A new instance of fragment FirstFragment.
     */

    @JvmStatic
    fun newInstance(param1: String, param2: String) =
        FirstFragment().apply {
            arguments = Bundle().apply {
                putString(ARG_PARAM1, param1)
                putString(ARG_PARAM2, param2)
            }
        }
}
```

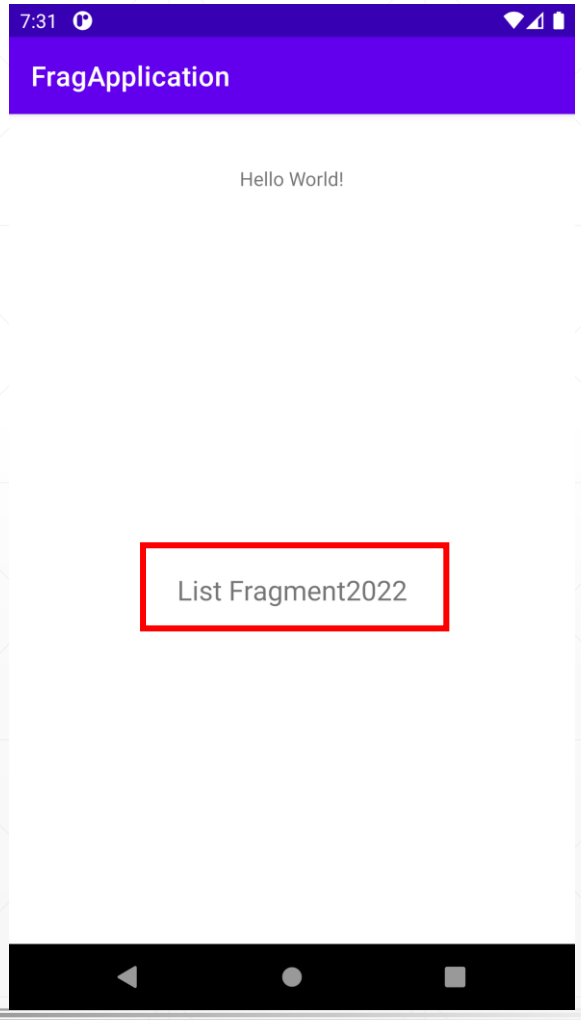
Adding a Single Fragment : Activity

- Adding a fragment with passing values
  - Instantiate a new Fragment
    - Use newInstance() method of a target Fragment
  - Pass the parameters

```
class MainActivity : AppCompatActivity() {

    val binding by lazy {ActivityMainBinding.inflate(layoutInflater)}

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(binding.root)

        var listFragment = FirstFragment.newInstance("List Fragment","2022")

        val fManager = supportFragmentManager
        fManager.commit {
            add(binding.fragFrame.id,listFragment)
        }

    }
}
```

Adding a Single Fragment : Activity

- Adding a fragment with passing values
  - Result)



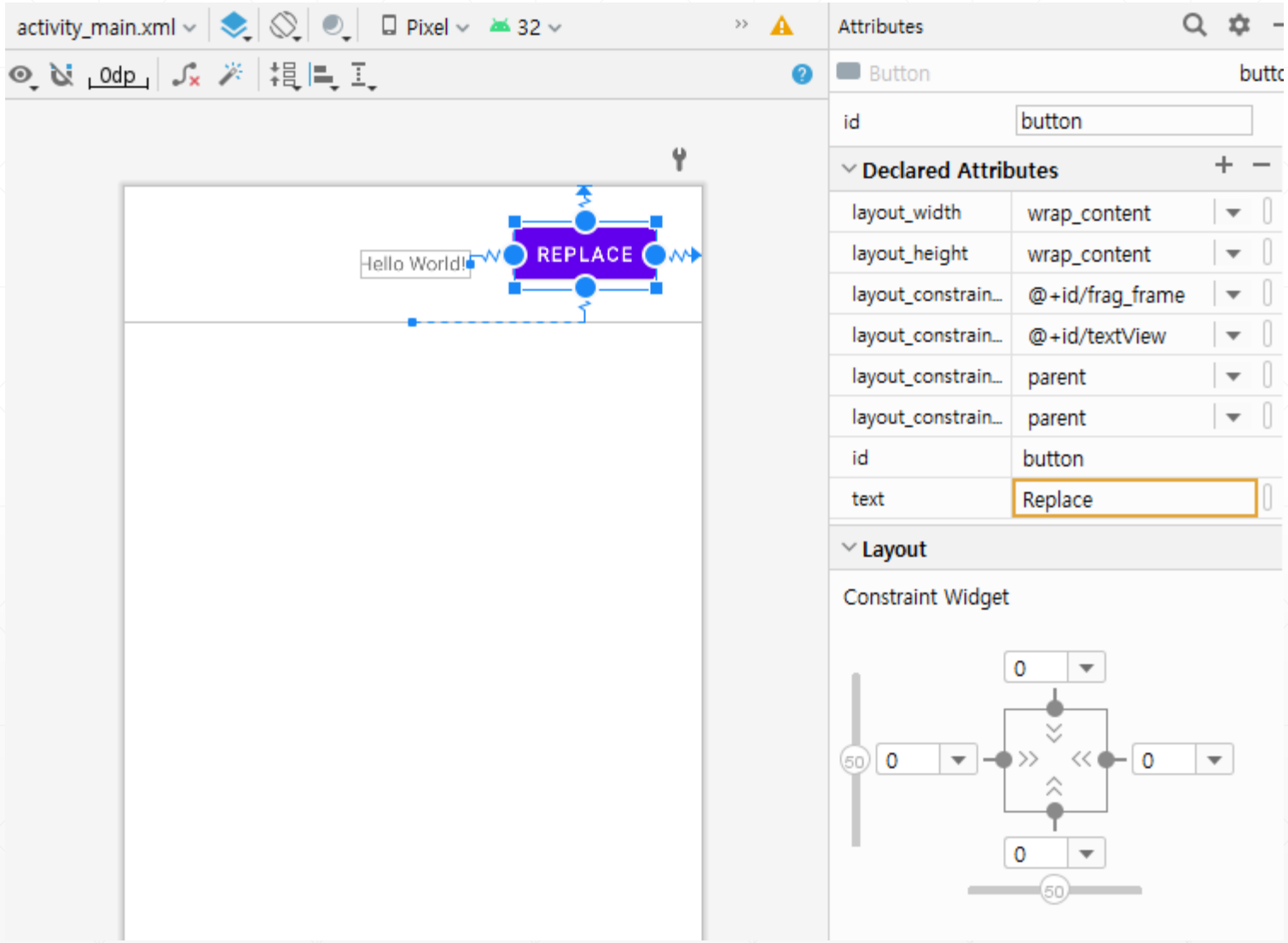

```
class MainActivity : AppCompatActivity() {

    val binding by lazy {ActivityMainBinding.inflate(layoutInflater)}

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(binding.root)

        var listFragment = FirstFragment.newInstance("List Fragment","2022")

        val fManager = supportFragmentManager
        fManager.commit {
            add(binding.fragFrame.id,listFragment)
        }

    }
}
```

Replacing a Fragment

- Add a second fragment
  - New → fragment
  - Name it! (e.g., SecondFragment)
  - Layout design for Fragment
    - FrameLayout → constraint layout
      - Use TextEditor!
    - Re-design the textView

Replacing a Fragment: Activity

- Edit Activity Layout
  - Add a button
  - Edit the text for the button



Replacing a Fragment: Activity

- Add codes to add/replace fragments
  - Simple! Use replace() method!

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(binding.root)

    var firFragment = FirstFragment.newInstance("First","2022")
    var secFragment = SecondFragment.newInstance("Second","2022")

    val fManager = supportFragmentManager
    fManager.commit {
        add(binding.fragFrame.id,firFragment)
    }

    binding.button.setOnClickListener{
        fManager.commit {

            replace(binding.fragFrame.id,secFragment)
        }
    }

}
```

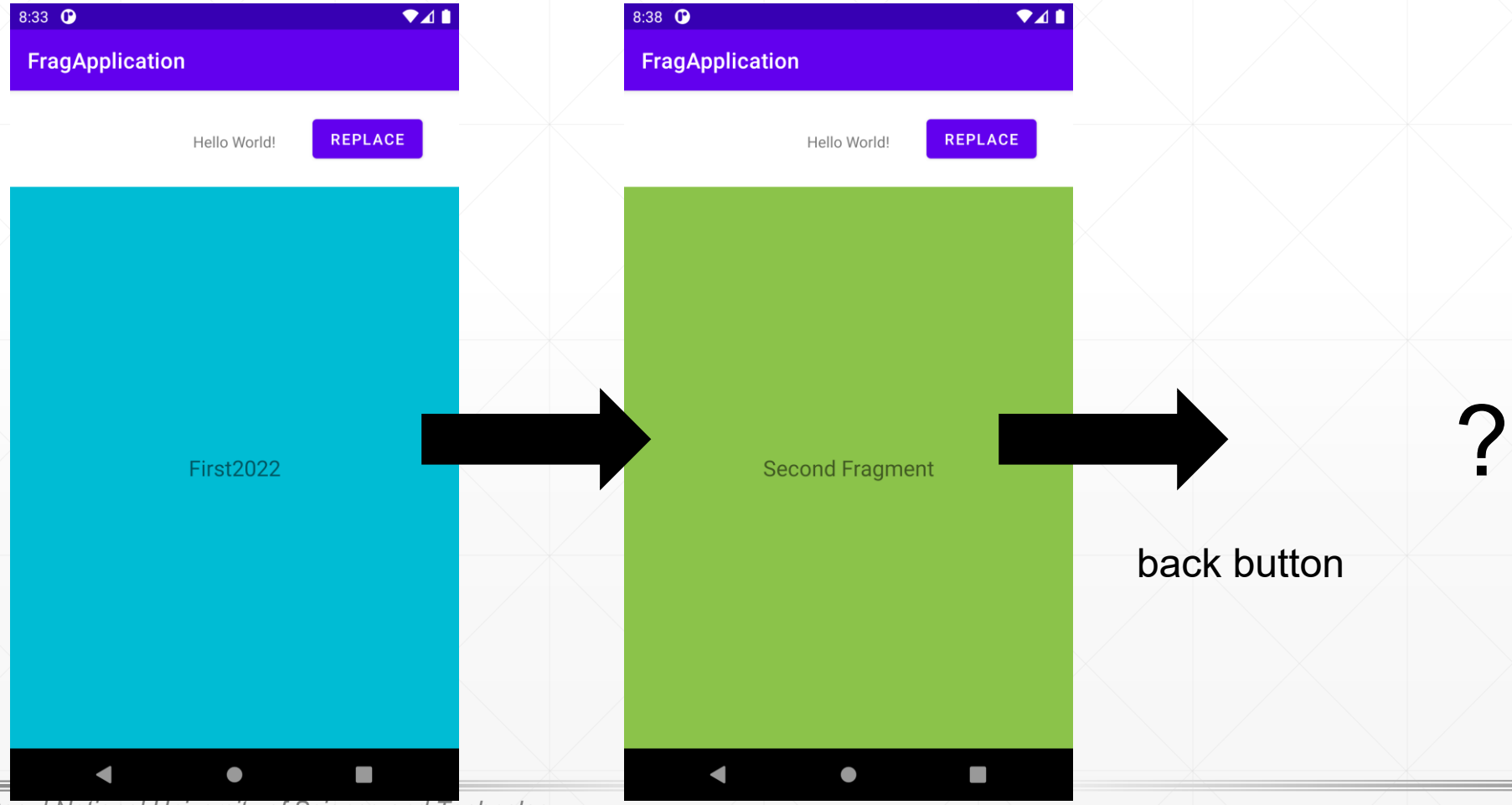
Replacing a Fragment: Fragment

- Add Log.d() routines for the following lifecycle callbacks to the fragments
  - onCreate
  - onCreateView
  - onDestroyView
  - onDestroy

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    arguments?.let {
        param1 = it.getString(ARG_PARAM1)
        param2 = it.getString(ARG_PARAM2)
    }
    Log.d("ITM","first fragment created")
}

override fun onDestroyView() {
    super.onDestroyView()
    Log.d("ITM","first fragment view destroyed")
}

override fun onDestroy() {
    super.onDestroy()
    Log.d("ITM","first fragment destroyed")
}

override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    // Inflate the layout for this fragment
    binding = FragmentFirstBinding.inflate(inflater, container, false)
    binding.fragText.text = param1+param2
    Log.d("ITM","first fragment view created")
    return binding.root
}
```

Replacing a Fragment

- Let's add and replace fragments!
  - … and see what happens? If you press back button



Replacing a Fragment

- addToBackStack()
  - We can also use back-stack concept for the fragment transactions!
  - Add a transaction to the back stack, which means that the transaction will be remembered after it is committed, and will reverse its operation when later popped off the stack.

```
binding.button.setOnClickListener{
    fManager.commit {
        setReorderingAllowed(true)
        addToBackStack(null)
        replace(binding.fragFrame.id,secFragment)
    }
}
```

- setReorderingAllowed()
  - Allows optimizing operations within and across transactions
  - This will remove redundant operations, eliminating operations that cancel

Replacing a Fragment: Activity

- Add codes to add/replace fragments
  - With addToBackStack() method calls
- Let's add and replace fragments!
  - … and see what happens?
    If you press back button

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(binding.root)

    var firFragment = FirstFragment.newInstance("First","2022")
    var secFragment = SecondFragment.newInstance("Second","2022")

    val fManager = supportFragmentManager
    fManager.commit {
        setReorderingAllowed(true)
        addToBackStack(null)
        add(binding.fragFrame.id,firFragment)
    }

    binding.button.setOnClickListener{
        fManager.commit {
            setReorderingAllowed(true)
            addToBackStack(null)
            replace(binding.fragFrame.id,secFragment)
        }
    }
}
```

Activity ←→ Fragment

- From activity to fragment
  - Simple! Just call the method of a target fragment!

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(binding.root)

    var firFragment = FirstFragment.newInstance("First","2022")
    var secFragment = SecondFragment.newInstance("Second","2022")

    val fManager = supportFragmentManager
    fManager.commit {
        setReorderingAllowed(true)
        addToBackStack(null)
        add(binding.fragFrame.id,firFragment)
    }

    binding.button.setOnClickListener{

        firFragment.setTitle("From the Main Activity!")
    }

}
```

```
class FirstFragment : Fragment() {
    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null
    lateinit var binding: FragmentFirstBinding

    fun setTitle(text: String) {
        binding.fragText.text = text
    }
…
```

Activity ←→ Fragment

- From fragment to activity
  - Fragment declares an interface for communication
  - Activity implements that interface
  - Fragment takes an Activity (interface realization) and invoke the interface method

- Add a spinner to the second fragment
  - If we select a particular item, the contents of that item will be passed to the activity

```
val myList = listOf("C", "C++", "C#", "Java", "Javascript", "Kotlin", "GoLang")
```

class property

```
val myAdapter = ArrayAdapter<String>(requireContext(), android.R.layout.simple_list_item_1, myList)
binding.spinner.adapter = myAdapter
binding.spinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
    override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
        callback.onItemPrint(myList.get(p2))
    }

    override fun onNothingSelected(p0: AdapterView<*>?) {}
}
```

Inside onCreateView callback

Activity ←→ Fragment

- From fragment to activity
  - Fragment declares an interface for communication

```
class SecondFragment : Fragment() {
    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null
    lateinit var binding: FragmentSecondBinding
    lateinit var callback: Callbacks

    val myList = listOf("C", "C++", "C#", "Java", "Javascript", "Kotlin", "GoLang")

    interface Callbacks{
        fun onItemPrint(num:String)
    }
```

Activity ←→ Fragment

- From fragment to activity
  - Fragment declares an interface for communication
  - Activity implements that interface

```
class MainActivity : AppCompatActivity(), SecondFragment.Callbacks {
    override fun onItemPrint(num: String) {
        binding.textView.text=num
    }
...
```

Activity ←→ Fragment

- From fragment to activity
  - Fragment declares an interface for communication
  - Activity implements that interface
  - Fragment takes an Activity (interface realization) and invoke the interface method

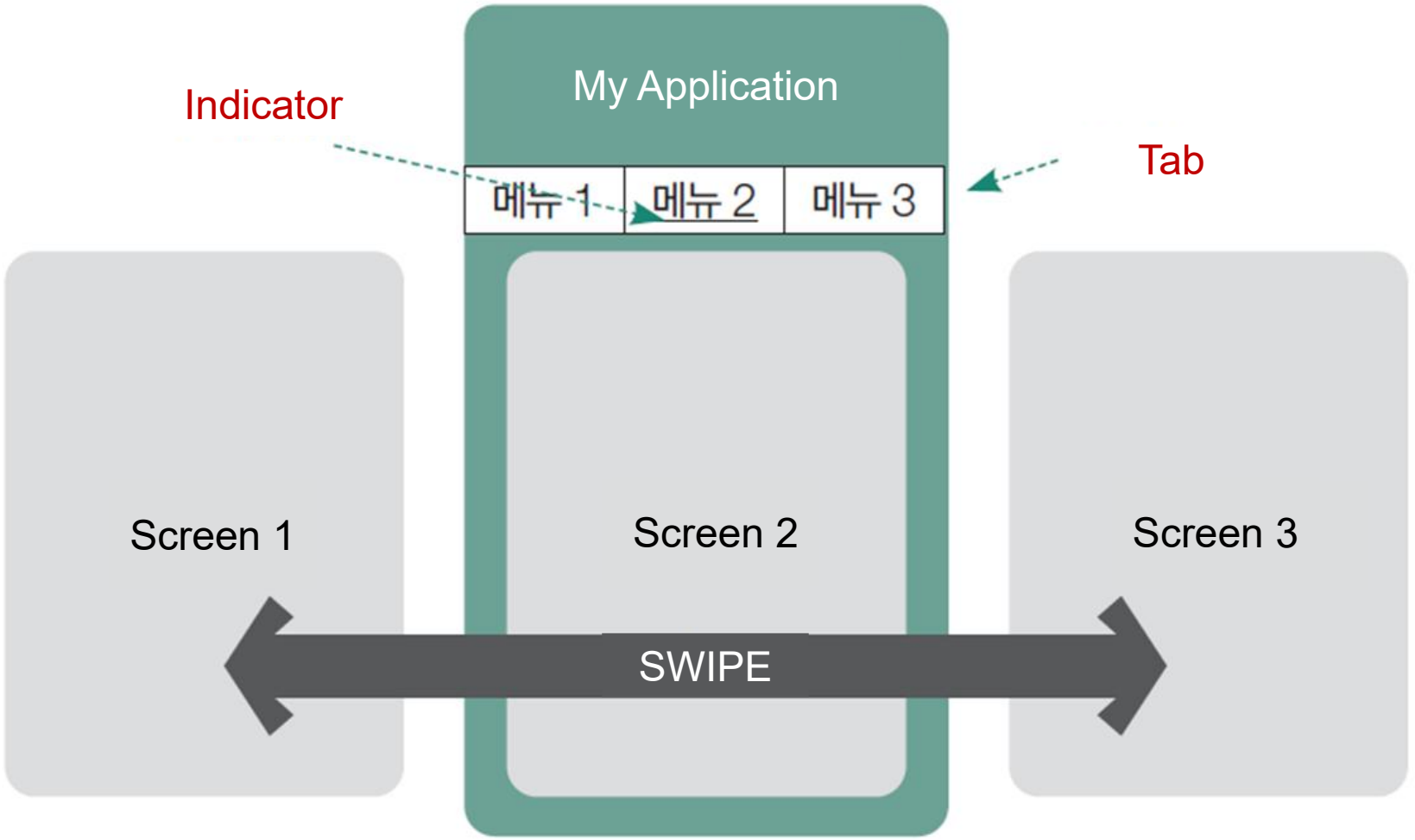
```
class SecondFragment : Fragment() {
    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null
    lateinit var binding: FragmentSecondBinding
    lateinit var callback: Callbacks

    val myList = listOf("C", "C++", "C#", "Java", "Javascript", "Kotlin", "GoLang")

    interface Callbacks{
        fun onItemPrint(num:String)
    }

    override fun onAttach(context: Context) {
        super.onAttach(context)
        callback = context as Callbacks
    }
…
```

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    // Inflate the layout for this fragment
    binding = FragmentSecondBinding.inflate(inflater, container, false)
    Log.d("ITM", "second fragment view created")
    val myAdapter = ArrayAdapter<String>(requireContext(), R.layout.simple_list_item_1, myList)
    binding.spinner.adapter = myAdapter
    binding.spinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
        override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
            callback.onItemPrint(myList.get(p2))
        }
        override fun onNothingSelected(p0: AdapterView<*>?) {}
    }
    return binding.root
}
```

Activity ←→ Fragment

- Communication between the fragments
  - ViewModel
  - Fragment Result API
  - …

ViewPager2 & TabLayout



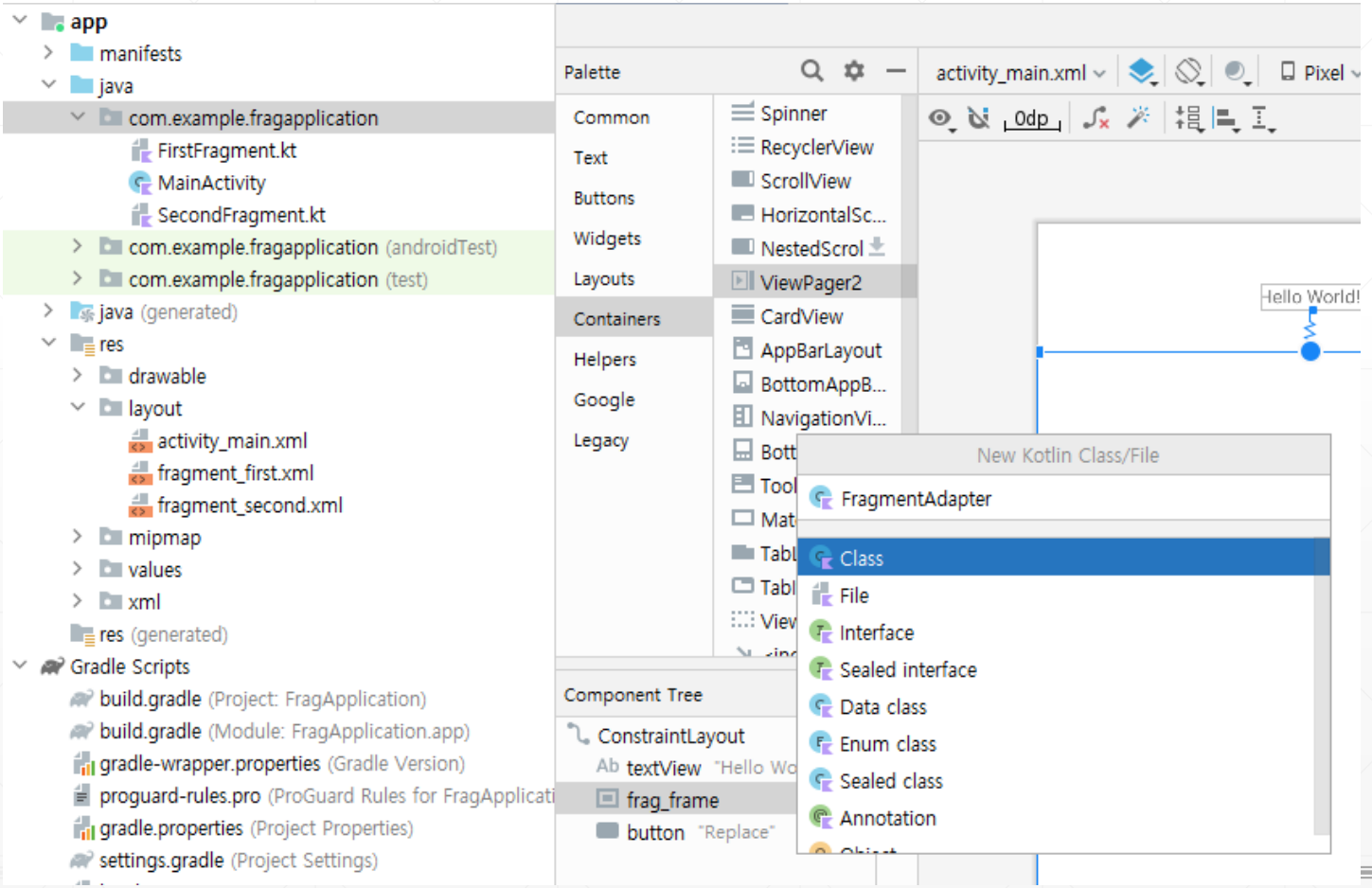
ViewPager2 & TabLayout

- ViewPager2
  - Very similar to recyclerView
  - Can be thought of like recyclerView with only a single page visible

- FragmentStateAdapter
  - Adapter for ViewPager2 with fragments

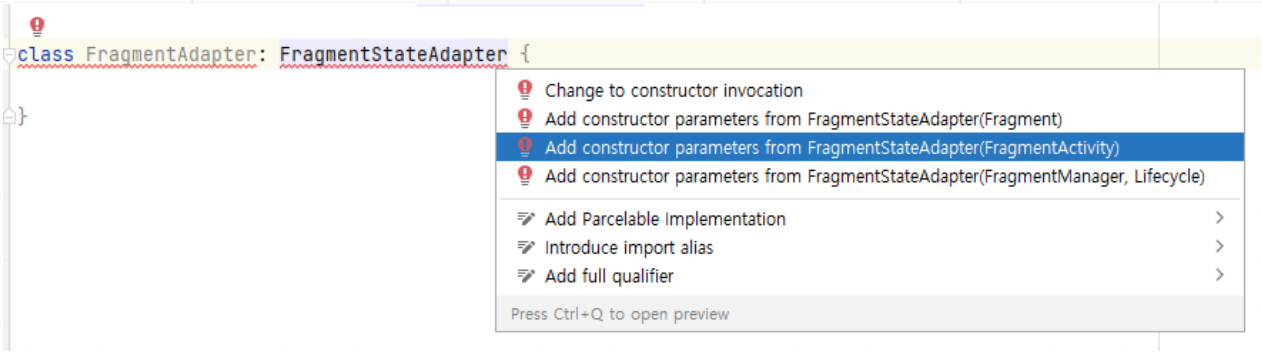
ViewPager2 & TabLayout

- FragmentStateAdapter
  - Adapter for ViewPager2 with fragments
  - Add a new class!



ViewPager2 & TabLayout

- Add constructor parameters



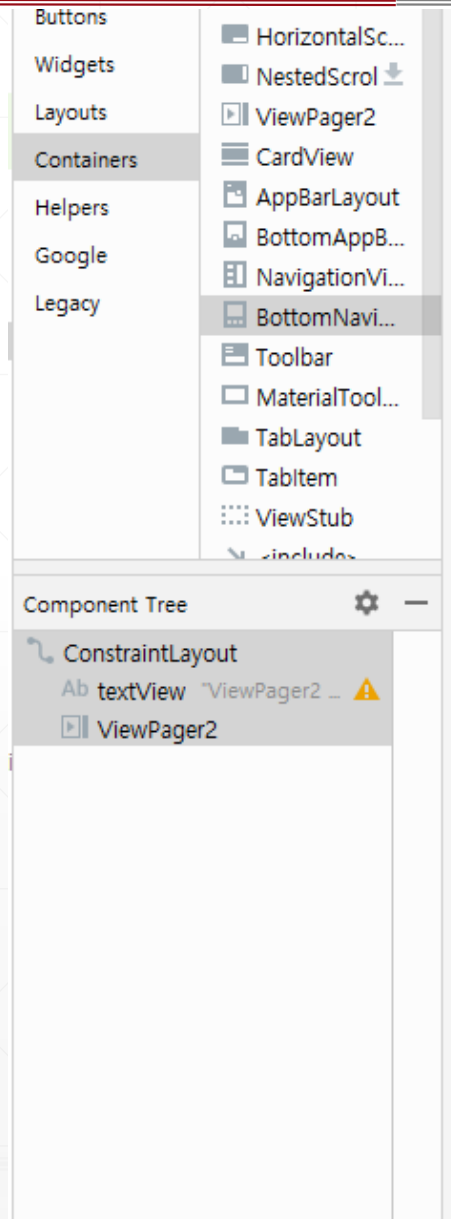
- Implement abstract methods and add variable for a list

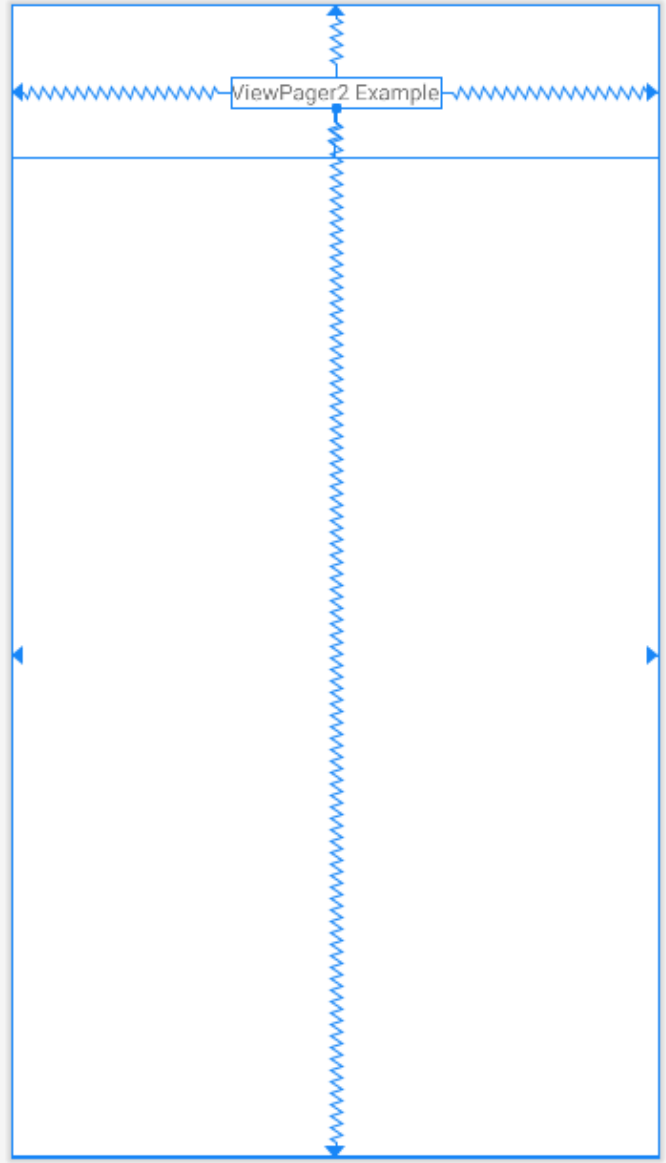
```
class FragmentAdapter(fragmentActivity: FragmentActivity) : FragmentStateAdapter(fragmentActivity) {
    var fragList = listOf<Fragment>()

    override fun getItemCount(): Int {
        return fragList.size
    }
    override fun createFragment(position: Int): Fragment {
        return fragList.get(position)
    }
}
```

ViewPager2 & TabLayout

- Add ViewPager2 to the Activity layout
  - Remove unnecessary UIs
    - The Original container (frame layout)
    - Button
  - Add viewPager2
    - Set ID
  - Map the viewPager2 to the adapter

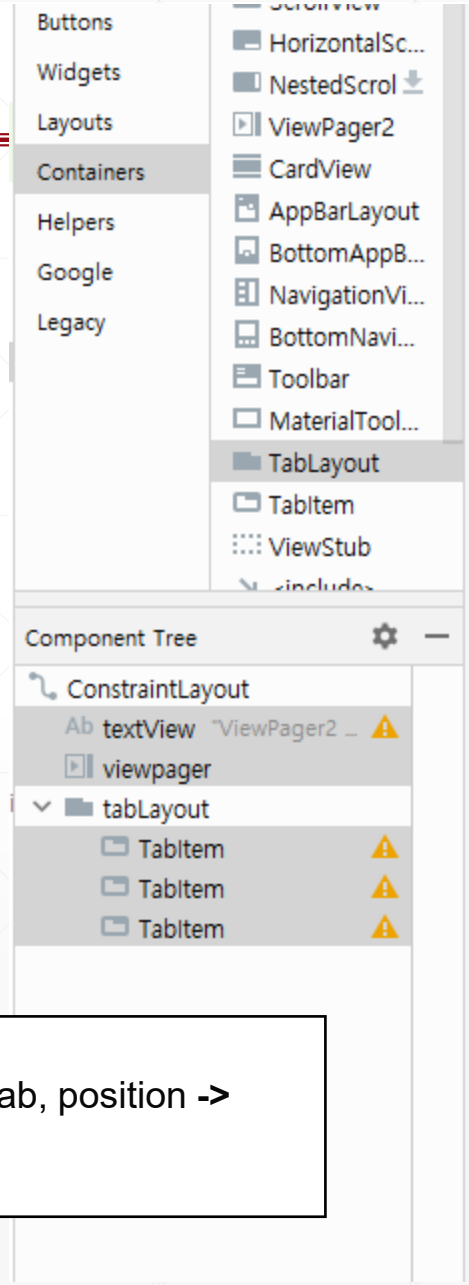
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(binding.root)

    var firFragment = FirstFragment.newInstance("First","2022")
    var secFragment = SecondFragment.newInstance("Second","2022")

    val myFrags = listOf(firFragment,secFragment)
    val fragAdapter = FragmentAdapter(this)
    fragAdapter.fragList = myFrags

    binding.viewpager.adapter = fragAdapter
…
```

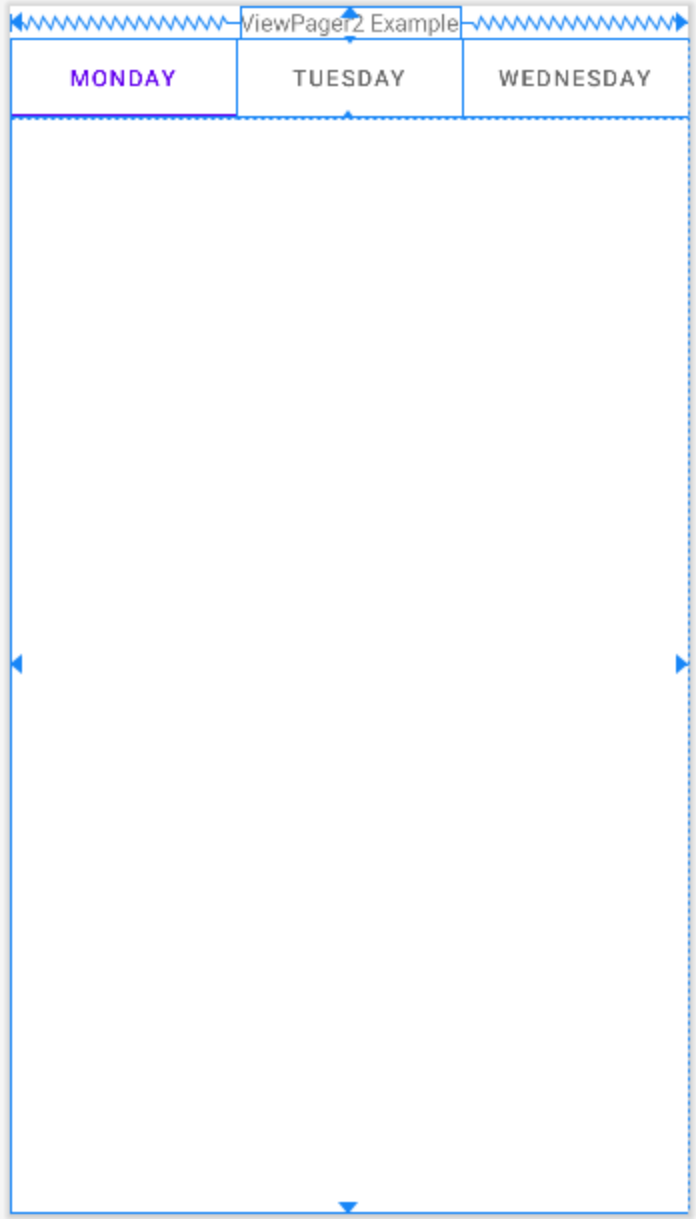




ViewPager2 & TabLayout

- Add TabLayout
  - Set ID!
- Use TabLayoutMediator
  - List for tab titles
  - TabLayout view
  - viewPager2 view






```
val tabs = listOf("First","Second")
TabLayoutMediator(binding.tabLayout,binding.viewpager){ tab, position ->
    tab.text = tabs.get(position)
}.attach()
```

ViewPager2 & TabLayout

- Result)