안드로이드 프로그래밍

2025년 2학기

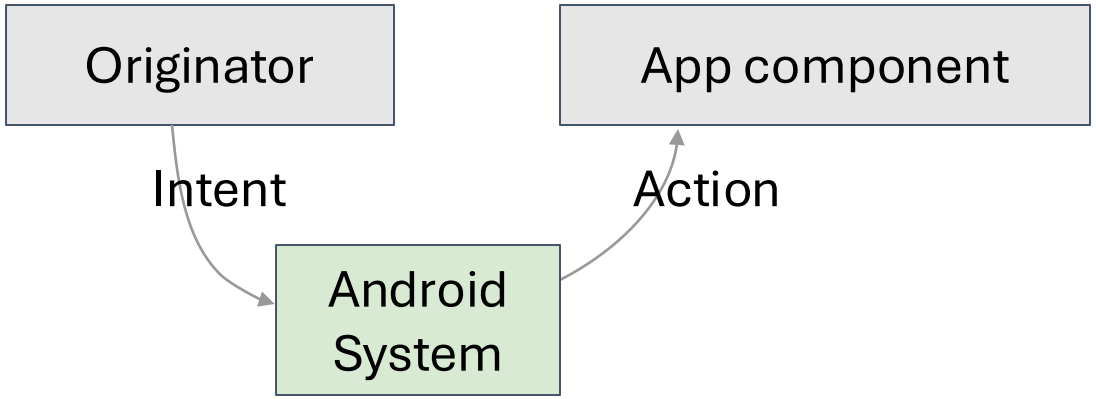
Intents

What is an intent?

- A message that is sent between app components, allowing them to communicate
  - Via direct method call: a reference to an Object and a method on it
  - Via event callbacks: on an event one of our callbacks gets executed by the system
- Intents
  - A component creates an object, gives it to another component, who can then respond upon receiving it
- Intents as like letters to a particular target (e.g., another Activity), and have room for data called extras (held in a Bundle)

What is an intent?

- An intent is a 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 activities
  - A button click starts a new activity for text entry
  - Clicking Share opens an app that allows you to post a photo
- Start services
  - Initiate downloading a file in the background
- Deliver broadcasts
  - The system informs everybody that the phone is now charging

Explicit and implicit intents

- Explicit Intent
  - Starts a specific activity
    - Request tea with milk delivered by Nikita
    - Main activity starts the ViewShoppingCart activity
- Implicit Intent
  - Asks system to find an activity that can handle this request
    - Find an open store that sells green tea
    - Clicking Share opens a chooser with a list of apps

1. Intents for Another Activity (Explicit)

- To start a specific activity (specifying the `Context` it should be delivered from and the `Class` it should be delivered to)
- Instantiate an intent

```
//                    context,              target
val intent = Intent(this@MainActivity, SecondActivity::class.java)
```

- Use the intent to start the activity by delivering the Intent via the `startActivity(intent)`

1. Intents for Another Activity (Explicit)

- Some extra data inside our envelope, referred to as **Extras**
  - Bundle (a set of primitive key-value pairs)

```
intent.putExtra("package.name.key", "value");
```

  - the full package name in Bundle keys, pre-defined key values (constants)
- Get the extras from the Intent in the Activity that receives it

```
val extras = intent.extras //All Activities are started with an Intent!
val value = extras.getString("key")
```

2. Intents for Another App (Implicit)

- We can also send intents to other applications (Implicit Intent).
- An Implicit Intent includes an Action and some Data
  - Action: what the target should do with the intent (a Command)
    - e.g., ACTION_VIEW to view some data, or ACTION_PICK to choose an item from a list, ACTION_MAIN to start the Activity as if it were a “main” launching point
  - Data gives more detail about what to run that action on
    - e.g., the Uri to VIEW or the Contact to DIAL
    - Extras then support this data

2. Intents for Another App (Implicit)

- For example, specify a DIAL action: our Intent to be delivered to an application that is capable of dialing a telephone number
  - we're just saying what kind of functionality we need

```
val intent = Intent(Intent.ACTION_DIAL)
intent.data = Uri.parse("tel:206-685-1622")
if (intent.resolveActivity(packageManager) != null) {
    startActivity(intent)
}
```

  - The resolveActivity() method looks up what Activity is going to receive our action.

3. Intents for a Response

- We create Intents in the same way, but use a different method to launch them: startActivityForResult().
  - The launched Activity will send *another* Intent back to us, which we can handle the result

- We'll launch the Camera to take a picture, then get the picture and show it in an ImageView we have.

3. Intents for a Response

```
private val REQUEST_IMAGE_CAPTURE = 1

fun dispatchTakePictureIntent() {
    val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
    if (takePictureIntent.resolveActivity(packageManager) != null) {
        startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE)
    }
}
```

- The MediaStore.ACTION_IMAGE_CAPTURE action (the action for “take a still picture and return it”)
- A second argument: a “request code”
  - used to distinguish this Intent from others we may send (kind of like a tag)
- Could pass an Extra for where we want to save the large picture file to

3. Intents for a Response

- To handle the “response” Intent, we provide a callback that will get executed when that Intent arrives: onActivityResult()

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent) {
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == Activity.RESULT_OK) {
        val extras = data.extras?.apply {
            val imageBitmap = get("data") as Bitmap
            val imageView = findViewById<ImageView>(R.id.img_thumbnail)
            imageView.setImageBitmap(imageBitmap)
        }
    }
}
```

  - which Intent led to the result (the requestCode) and the status of that request (the *result*Code)
  - access to the returned data (e.g., the image) by getting the "data" value from the extras

3. Intents for a Response

- Three “parts” of a filter:
  - a <action android:name="action"> filter: the Action we can respond to
  - a <data …> filter: aspects of the data we accept
    - e.g., only respond to Uri’s that look like telephone numbers
  - a <category android:name="category"> filter: a “more information” piece
- You must include the DEFAULT category to receive Implicit Intents
  - used by startActivity() and startActivityForResult()
- You can include multiple actions, data, and category tags
  - multiple types are combined with an “or”, not an “and”

3. Intents for a Response

- Responding to that dial command

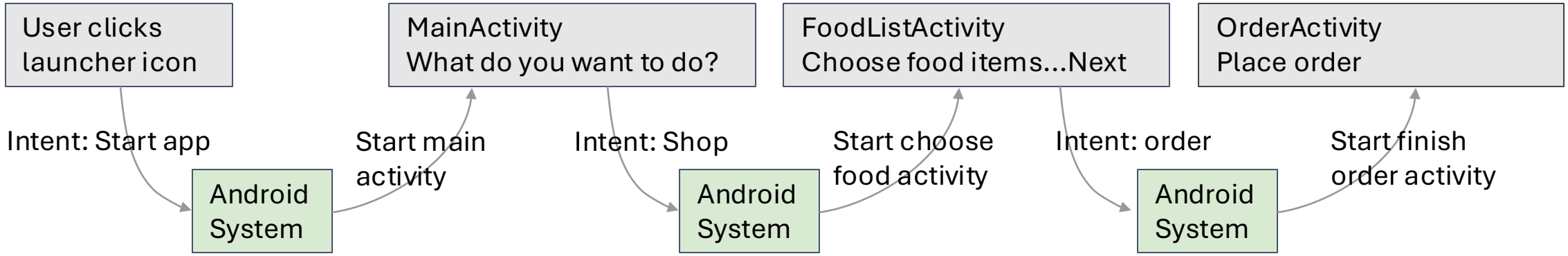
```
<activity android:name=".SecondActivity">
  <intent-filter>
      <action android:name="android.intent.action.DIAL"/>
      <category android:name="android.intent.category.DEFAULT" />
      <data android:scheme="tel" /> <!-- telephone Uris -->
  </intent-filter>
</activity>
```

4. Listening for Intents

- In order to receive an Implicit Intent, we need to declare that our Activity is able to handle that request.
  - in the Manifest using an <intent-filter>
  - We're "hearing" all the intents, and we're "filtering" for the ones that are relevant to us
- An <intent-filter> tag is nested inside the element that it applies to (e.g., the <activity>).
  - responds to the MAIN action sent with the LAUNCHER category (meaning that it responds to Intents from the app launcher)

How Activities Run

- All activities are managed by the Android runtime
- Started by an "intent", a message to the Android runtime to run an activity

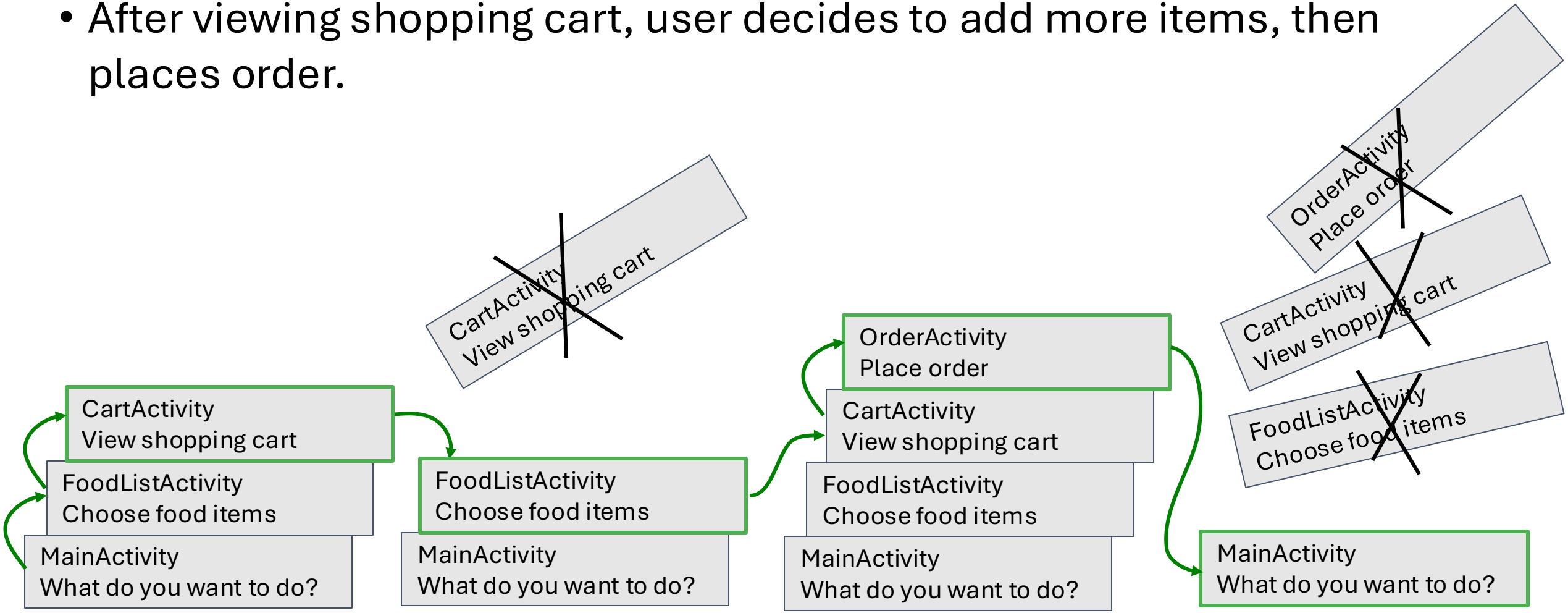


Activity stack

- When a new activity is started, the previous activity is stopped and pushed on the activity back stack
- Last-in-first-out-stack—when the current activity ends, or the user presses the Back button, it is popped from the stack and the previous activity resumes

Activity Stack

- After viewing shopping cart, user decides to add more items, then places order.



Two forms of navigation

- Temporal or back navigation
  - provided by the device's back button
  - controlled by the Android system's back stack

- Ancestral or up navigation
  - provided by the app's action bar
  - controlled by defining parent-child relationships between activities in the Android manifest

Back navigation

- Back stack preserves history of recently viewed screens
- Back stack contains all the activities that have been launched by the user in reverse order for the current task
- Each task has its own back stack
- Switching between tasks activates that task's back stack
- Launching an activity from the home screen starts a new task
- Navigate between tasks with the overview or recent tasks screen

Up navigation

- Goes to parent of current activity
- Define an activity's parent in Android manifest
- Set parentActivityName

```
<activity
   android:name=".ShowDinnerActivity"
   android:parentActivityName=".MainActivity" >
</activity>
```