Intents, Fragments & Toast

Mobile App Development
(Android studio)

WHAT IS AN INTENT?

An intent is a messaging object used to request any action from another app component.

Intents facilitate communication between different components in several ways.

The intent is used to launch an activity, start the services, broadcast receivers, display a web page, dial a phone call, send messages from one activity to another activity, and so on.

Structure of an Intent

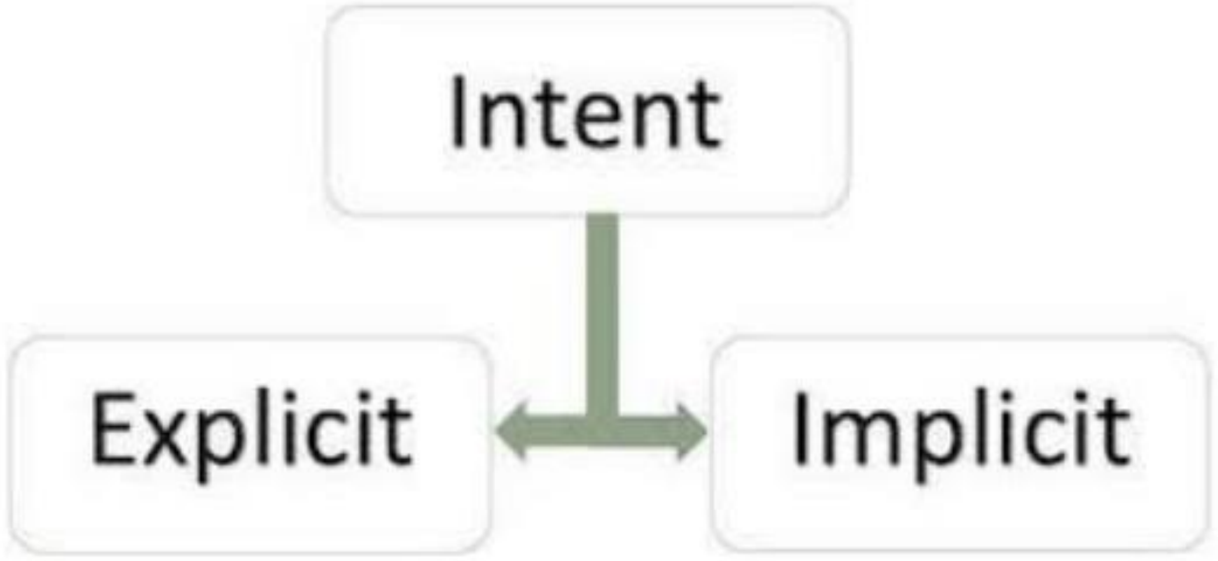
An **Intent** object consists of:

- **Component Name** (only for explicit intents)
- **Action** (Defines what the intent will do)
- **Data** (URI for data to be processed)
- **Category** (Additional information about the action)
- **Extras** (Key-value pairs to send additional data)

Types of Intent

Intents are of two types:

- Explicit intent
- Implicit intent



EXPLICIT INTENT

Explicit intents are communicated between two activities inside the same application. We can use explicit intents when we need to move from one activity to another activity. Example:- when a user wants to start a service to download a file or when a new activity gets started in response to a user action.

```
Intent send = new Intent(FirstActivtiy.this, SecondActivity.class);

//Starts TargetActivity

startActivity(send);
```

Examples

Example: Starting a New Activity (Explicit Intent)

```
Intent intent = new Intent(CurrentActivity.this, NewActivity.class);
startActivity(intent);
```

Example: Passing Data between Activities

```
Intent intent = new Intent(CurrentActivity.this, NewActivity.class);
intent.putExtra("username", "JohnDoe");
// Passing data
startActivity(intent);
```

Receiving Data in New Activity

```
String username = getIntent().getStringExtra("username");
```

IMPLICIT INTENT

Implicit intent is communicated between two activities of an application. One thing to note here is we should not name a specific component. In place of that we can declare a general action to perform which allows a component from another app to handle it. Example:- To show the user a location on a map, we can use an implicit intent.

```
 Intent intent = new Intent();

 intent.setAction(android.content.Intent.ACTION_VIEW);
intent.setData(Contract.Contacts.CONTENT_URL);

 startActivity(intent);
```

Explicit Intent Vs. Implicit Intent

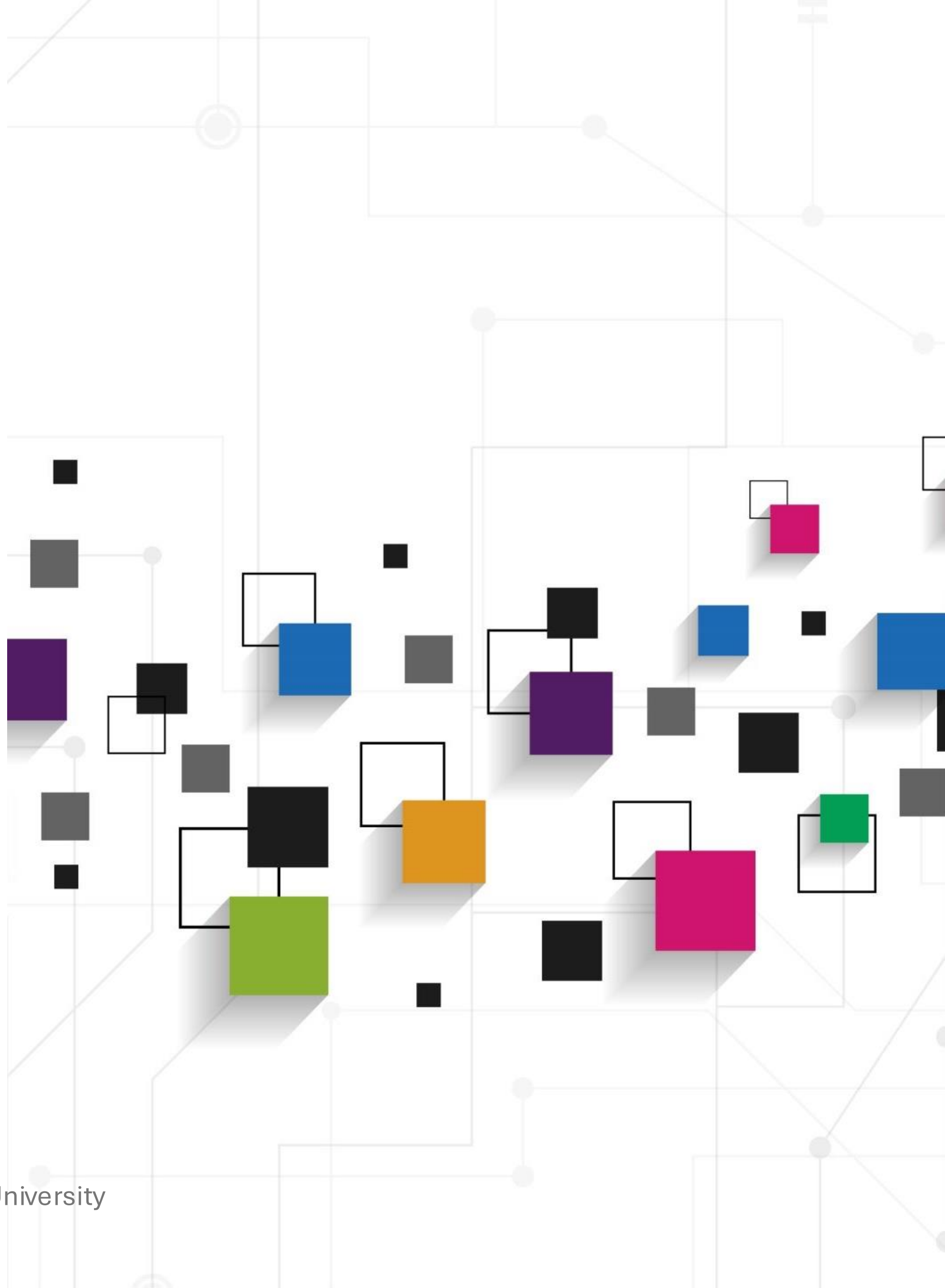
Explicit Intent	Implicit Intent
Explicit intents are those in which the user has a clear vision and knows exactly which activity can handle the requests.	Implicit intents do not name a specific component like explicit intent, instead declare general action to perform, which allows a component from another app to handle.
Explicit intent can do the specific application action which is set by the code like changing activity, downloading the file in the background, etc.	It specifies the only action to be performed and does not directly specify Android Components.
In explicit intent, you can pass data to other activities by using the putExtra method and retrieve by using getIntent().	The action in the intent and OS decides which applications are suitable to handle the task, action across two different applications.
Explicit intents are used for communication inside the application. Like changing activities inside the application.	They are used for communication across two different applications.
In explicit intent, the action target is delivered even the filter is not consulted.	When you make an implicit call with the intent. OS look at the action and then it matches with all the filters intent-filters of all the registered activities of all application using PackageManager and then populates the result as a list, this process is called as intent Resolution.

Introduction to Fragments

A **Fragment** is a modular and reusable UI component that represents a part of an activity. Fragments are used to create flexible and dynamic UIs, especially in large-screen devices like tablets.

Features of Fragments:

- Can be added or removed dynamically.
- Can be reused across multiple activities.
- Have their own lifecycle, similar to activities.
- Allow **multiple UI components** in a single activity.



Why Use Fragments?

Modular Design: Encourages reusability.

Dynamic UI: Supports multiple layouts on different screen sizes.

Navigation Control: Used with ViewPager, TabLayout, and Navigation Component.

Multiple Fragments per Activity: Makes UI more flexible.

Adding Fragments

Fragments can be added to activities in two ways:
Static Fragments – Defined in XML.
Dynamic Fragments – Added/removed at runtime via code.

Adding Static Fragments (XML Approach)

Use <fragment> tag in activity_main.xml:

```
<fragment
   android:id="@+id/fragment_container"
   android:layout_width="match_parent"
   android:layout_height="wrap_content" />
```

Adding Fragments Dynamically

Step 1: Create a Fragment Class

```
public class MyFragment extends Fragment {
   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
      return inflater.inflate(R.layout.fragment_layout, container, false);
   }
}
```

Step 2: Add Fragment Dynamically in Activity

```
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction transaction = fragmentManager.beginTransaction();
MyFragment myFragment = new MyFragment();
transaction.add(R.id.fragment_container, myFragment);
transaction.commit();
```

Adding Fragments Dynamically

Step 3: Remove a Fragment Dynamically

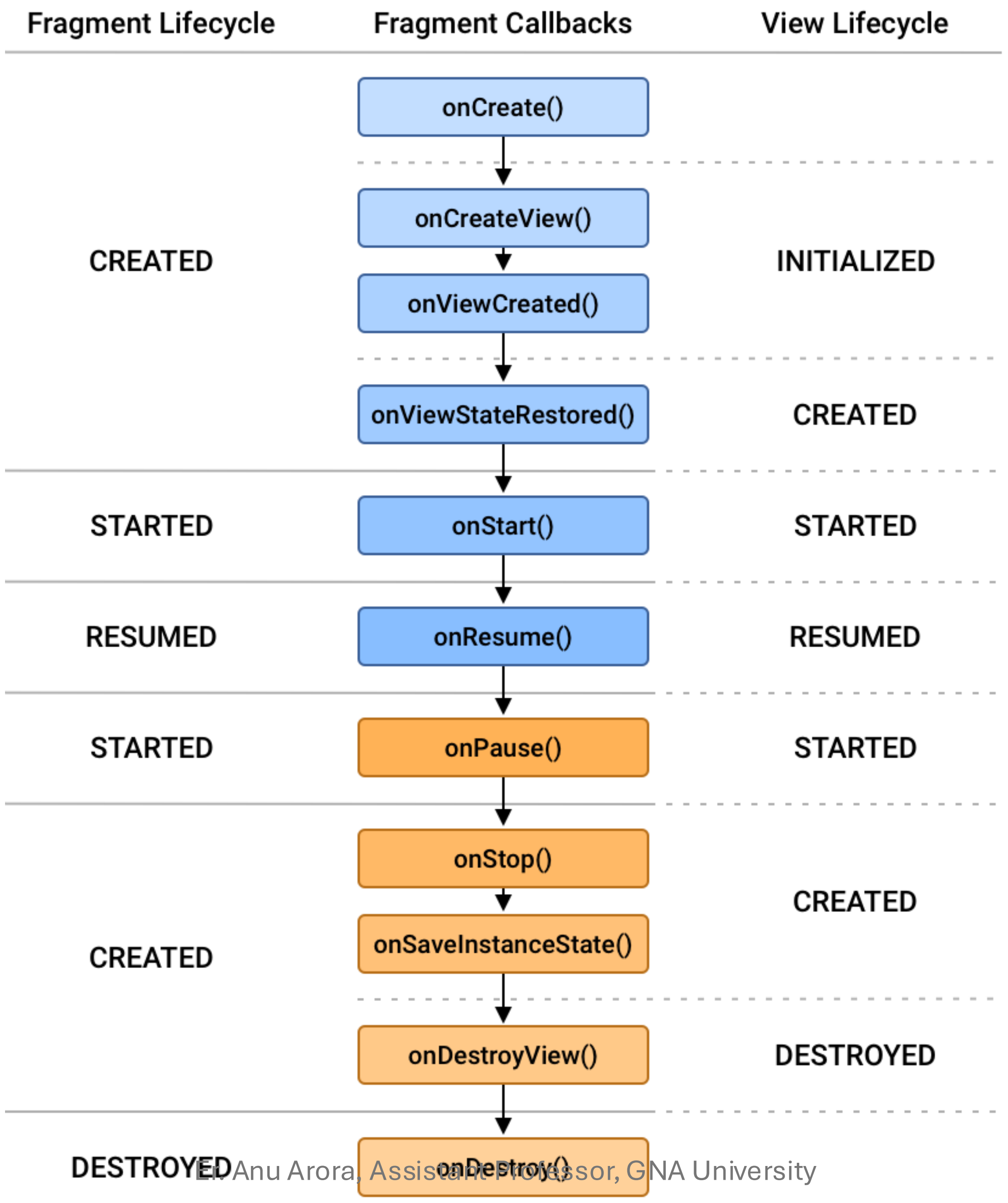
```
FragmentTransaction transaction = fragmentManager.beginTransaction();
transaction.remove(myFragment);
transaction.commit();
```

Step 4: Replace a Fragment

```
transaction.replace(R.id.fragment_container, new AnotherFragment());
transaction.commit();
```

Life Cycle of Fragment

Method	**Description**
onAttach()	Called when the fragment is attached to an activity.
onCreate()	Initializes the fragment (non-UI components).
onCreateView()	Inflates the fragment layout.
onViewCreated()	Called after onCreateView(), used for setting up UI components.
onStart()	Fragment becomes visible to the user.
onResume()	Fragment becomes active and interactive.
onPause()	User is leaving the fragment (partially visible).
onStop()	Fragment is no longer visible.
onDestroyView()	Cleans up view-related resources.
onDestroy()	Final cleanup before the fragment is removed.
onDetach()	Called when the fragment is detached from its activity.


Fragment Lifecycle
Fragment Callbacks
View Lifecycle
onCreate()
onCreateView()
onViewCreated()
onViewStateRestored()
onStart()
onResume()
onPause()
onStop()
onSaveInstanceState()
onDestroyView()
onDestroy()
CREATED
STARTED
RESUMED
STARTED
CREATED
DESTROYED
INITIALIZED
CREATED
STARTED
RESUMED
STARTED
CREATED
DESTROYED

Introduction to Toast in Android

A **Toast** in Android is a short, non-interactive message that appears on the screen and disappears automatically.

It is used for providing **feedback or notifications** to the user **without interrupting their experience**.

- **Duration:** Short-lived (few seconds).
- **Position:** Usually appears at the **bottom of the screen** (can be customized).
- **Use Case:** Display **simple notifications, confirmations, or status messages**.

Creating a Toast in Android

A **basic Toast** can be created using the Toast.makeText() method.

Example: Basic Toast Message

```
Toast.makeText(getApplicationContext(), "Hello, this is a Toast!",
Toast.LENGTH_SHORT).show();
```

Parameters:

1. **getApplicationContext()** – Context of the application.
2. **"Hello, this is a Toast!"** – Message to be displayed.
3. **Toast.LENGTH_SHORT** – Duration (LENGTH_SHORT or LENGTH_LONG).
4. **show()** – Displays the Toast.

Types of Toasts in Android

Simple Toast: Displays a basic message.

```
Toast.makeText(getApplicationContext(), "Simple Toast Message",
Toast.LENGTH_LONG).show();
```

Custom Toast: You can **customize the layout** of a Toast using a custom XML file.

Steps to Create a Custom Toast:

1. Create a **custom layout** (custom_toast.xml).
2. Inflate the layout in Java code.
3. Display the custom Toast.

XML code:

```
<ImageView
    android:layout_width="30dp"
    android:layout_height="30dp"
    android:src="@drawable/ic_info" />
 <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Custom Toast!"
    android:textColor="@android:color/white"
    android:textSize="16sp"
    android:paddingLeft="8dp"/>
```

Java Code:

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_toast,
null);


Toast toast = new Toast(getApplicationContext());
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();
```

Positioning a Toast

You can change the **position** of the Toast using setGravity().

```
Toast toast = Toast.makeText(getApplicationContext(), "Toast at Top!",
Toast.LENGTH_SHORT);
toast.setGravity(Gravity.TOP, 0, 200);
toast.show();
```

Gravity.TOP – Displays the Toast at the top.
0, 200 – Adjusts the x and y offset.