**Assignment-3**

1. **What is Intent? Explain Explicit vs. Implicit intents with Example?**

* **In the context of Android app development, "intent"** refers to a messaging object that's used to request an action or communicate between different components of an Android application or between different Android applications. Intents are a fundamental part of the Android operating system and are used to trigger various activities, services, and broadcasts.There are two main types of intents in Android: Explicit Intents and Implicit Intents.
* **Explicit Intents:** Explicit intents are used to start a specific component (usually an activity or service) within the same application. They explicitly specify the target component by its class name. They are typically used for navigation within an app. You provide the exact component name to launch, and Android knows exactly which component to start.

|  |
| --- |
| * **Example of an Explicit Intent:**   Intent explicitIntent = new Intent(this, SecondActivity.class);  startActivity(explicitIntent); |

* **Implicit Intents:** Implicit intents do not specify the exact component to be executed; instead, they declare a general action to be performed, and Android system determines which component is capable of fulfilling that action. They are often used when you want to perform an action that might involve components from different apps, such as sharing content or opening a web URL. Implicit intents include an action, data (optional), and category (optional) to describe the operation to be performed.

|  |
| --- |
| * **Example of an Implicit Intent to open a web page:**   Intent implicitIntent = new Intent(Intent.ACTION\_VIEW, Uri.parse("https://www.example.com"));  startActivity(implicitIntent); |

1. **Compare implicit and explicit intents?**

* **Implicit intents and explicit intents** are two types of intents used in Android development. They differ in how they specify the component that should handle the intent.
* **Explicit intents** explicitly specify the component that should handle the intent. This means that the developer knows the exact name of the class that implements the component and includes it in the intent object. Explicit intents are typically used when the developer wants to launch a specific activity or service within their own app, or when they need to start an activity in another app that has the appropriate intent filter.
* **Implicit** intents do not explicitly specify the component that should handle the intent. Instead, it specifies the action to be performed, the data to be acted upon, and the type of component that should handle the intent. The Android system will then broadcast the intent to all registered components that match the intent filter. If multiple components match the intent filter, the user will be prompted to choose which component to handle the intent.
* **Here is a table that summarizes the key differences between implicit and explicit intents:**

|  |  |  |
| --- | --- | --- |
| Feature | Implicit Intent | Explicit Intent |
| Specificity | Does not explicitly specify the component to handle the intent | Explicitly specifies the component to handle the intent |
| Flexibility | More flexible, as it can launch any component that matches the intent filter | Less flexible, as it can only launch a specific component |
| Use Cases | Typically used when the developer wants to launch a component without knowing the exact implementation | Typically used when the developer knows the exact component to launch |
| Performance | More efficient, as the Android system does not need to broadcast the intent to all registered components | Less efficient, as the Android system needs to broadcast the intent to all registered components |
| Security | Less secure, as the developer does not have control over which component handles the intent | More secure, as the developer has control over which component handles the intent |

1. **Write a code to send data from one activity to another activity using implicit intent?**

* **To send data** from one activity to another using an implicit intent, you'll need to create an intent object and specify the action, data, and type of component to handle the intent. The receiving activity will then need to retrieve the data from the intent object. Here's an example of how to send data from one activity to another using an implicit intent:
* **Sending Activity:**

|  |
| --- |
| Intent intent = new Intent(Intent.ACTION\_SEND);  intent.setType("text/plain");  intent.putExtra(Intent.EXTRA\_TEXT, "This is the data to be sent");  startActivity(intent); |

* **Receiving Activity:**

|  |
| --- |
| Intent intent = getIntent();  String data = intent.getStringExtra(Intent.EXTRA\_TEXT);  textView.setText(data); |

In this example, the sending activity creates an intent with the ACTION\_SEND action and the text/plain type. The EXTRA\_TEXT extra is used to store the data to be sent. The receiving activity retrieves the data from the intent using the getStringExtra() method.

1. **Write code to send data from one activity to other activity using Explicit Intent?**

* **To send data from one activity to another using an explicit intent,** you'll need to create an intent object, specify the target activity class, and add the data you want to send using the putExtra() method. The receiving activity will then need to retrieve the data using the getIntent() method and the getStringExtra() method. Here's an example of how to send data from one activity to another using an explicit intent:
* **Sending Activity (MainActivity):**

|  |
| --- |
| Intent intent = new Intent(this, SecondActivity.class);  intent.putExtra("message", "This is the data to be sent");  startActivity(intent); |

* **Receiving Activity (SecondActivity):**

|  |
| --- |
| Intent intent = getIntent();  String message = intent.getStringExtra("message");  textView.setText(message); |

In this example, the sending activity (MainActivity) creates an intent and specifies the target activity class (SecondActivity). It then adds the data to be sent using the putExtra() method. The receiving activity (SecondActivity) retrieves the data using the getIntent() method and the getStringExtra() method.

1. **Write a Program move one page to another page using Intent?**

* **Here is an example of how to move from one page to another page using an Intent in Android:**

|  |
| --- |
| Intent intent = new Intent(this, SecondActivity.class);  startActivity(intent);   * This code will create an Intent object specifying the target activity class (SecondActivity) and then start the activity. |

* **Here is an example of how to send data from one page to another page using an Intent:**

|  |
| --- |
| Intent intent = new Intent(this, SecondActivity.class);  intent.putExtra("message", "This is the data to be sent");  startActivity(intent);   * This code will create an Intent object specifying the target activity class (SecondActivity) and then add the data to be sent using the putExtra() method. The receiving activity can then retrieve the data using the getIntent() method and the getStringExtra() method. |

* **Here is an example of how to start a service from an activity using an Intent:**

|  |
| --- |
| Intent intent = new Intent(this, MyService.class);  startService(intent);   * This code will create an Intent object specifying the target service class (MyService) and then start the service. |

* **Here is an example of how to stop a service from an activity using an Intent:**

|  |
| --- |
| Intent intent = new Intent(this, MyService.class);  stopService(intent);   * This code will create an Intent object specifying the target service class (MyService) and then stop the service. |

* **Here is an example of how to send a broadcast using an Intent:**

|  |
| --- |
| Intent intent = new Intent("com.example.mybroadcast");  intent.putExtra("message", "This is the broadcast message");  sendBroadcast(intent);   * This code will create an Intent object specifying the broadcast action ("com.example.mybroadcast") and then send the broadcast. Any receiver that is registered for the broadcast action will receive the broadcast message. |

1. **Describe /Write a short note on Shared preference?**

* **SharedPreferences** is a lightweight data storage mechanism provided by the Android framework. It allows applications to store and retrieve small amounts of primitive data, such as integers, strings, and booleans, as key-value pairs. The data is stored in a private file on the device, so it is not accessible to other applications.
* **Advantages of using SharedPreferences:**
* **Simple and easy to use:** SharedPreferences provides a simple API for storing and retrieving data.
* **Lightweight:** SharedPreferences is a lightweight data storage mechanism, so it does not consume a lot of memory or storage space.
* **Private:** SharedPreferences data is stored in a private file on the device, so it is not accessible to other applications.
* **Disadvantages of using SharedPreferences:**
* **Not suitable for large amounts of data:** SharedPreferences is not suitable for storing large amounts of data, as it can slow down your application.
* **Not secure:** SharedPreferences data is not encrypted, so it is not secure against unauthorized access.
* **Common uses for SharedPreferences:**
* **Storing user preferences:** SharedPreferences can be used to store user preferences, such as language settings, font size, and notification settings.
* **Storing app state:** SharedPreferences can be used to store app state, such as the current page in a pager or the last position in a list.
* **Caching data:** SharedPreferences can be used to cache data, such as frequently accessed web content or images.
* **Here is an example of how to use SharedPreferences to store a user preference:**

|  |
| --- |
| SharedPreferences sharedPreferences = getSharedPreferences("myPreferences", MODE\_PRIVATE);  SharedPreferences.Editor editor = sharedPreferences.edit();  editor.putBoolean("darkModeEnabled", true);  editor.apply();   * This code will store a boolean value (true) in SharedPreferences under the key "darkModeEnabled". The MODE\_PRIVATE mode indicates that the shared preferences file should be private to the application. |

* **Here is an example of how to retrieve a user preference from SharedPreferences:**

|  |
| --- |
| SharedPreferences sharedPreferences = getSharedPreferences("myPreferences", MODE\_PRIVATE);  boolean darkModeEnabled = sharedPreferences.getBoolean("darkModeEnabled", false);   * This code will retrieve the value stored under the key "darkModeEnabled" from SharedPreferences. If the key does not exist, the default value of false will be returned. |

1. **What is Intent Filter? Explain key points about intent filters with Android Mainifest.xml file code?**

* **An intent filter** is an element in an Android application's manifest file that defines the types of intents that an app component can receive and respond to. It specifies the actions, categories, and data that an intent must match in order for the component to be considered a potential recipient.
* **Key Points about Intent Filters:**
* **Components:** Intent filters can be associated with activities, services, and broadcast receivers. Each component's intent filter defines the intents it can handle.
* **Elements:** An intent filter consists of one or more <action>, <category>, and <data> elements. These elements define the criteria that an intent must meet to match the filter.
* **Action:** The <action> element specifies the action that an intent must declare. For example, ACTION\_VIEW indicates the intent is meant to view content.
* **Category:** The <category> element defines additional classifications for the intent. For instance, CATEGORY\_BROWSABLE suggests the intent is related to browsing the web.
* **Data:** The <data> element specifies the data associated with the intent, such as a URI or MIME type. It filters intents based on the data they carry.
* **Priority:** Intent filters can have a priority attribute that influences the Android system's selection process. Higher priority filters are preferred over lower ones.
* **Android Manifest.xml:** Intent filters are declared within the <activity>, <service>, or <receiver> elements of the Android Manifest.xml file. They are nested within the corresponding component element.
* **Example of Intent Filter in Android Manifest.xml:**

|  |
| --- |
| <activity android:name=".MainActivity">  <intent-filter>  <action android:name="android.intent.action.MAIN" />  <category android:name="android.intent.category.LAUNCHER" />  </intent-filter>  </activity>   * This example defines an intent filter for an activity named MainActivity. It indicates that the activity can handle the android.intent.action.MAIN action and belongs to the android.intent.category.LAUNCHER category, making it eligible to be launched from the device's home screen. |

1. **Explain about:**
2. **DatePicker:** A DatePicker is a UI component that allows the user to select a date from a calendar.It's often used for tasks that require the user to choose a specific date, such as setting reminders, selecting birthdates, or scheduling appointments.In Android, you can use the DatePickerDialog to create a DatePicker and display it to the user.
3. **TimePicker:** A TimePicker is a UI component that enables the user to select a specific time, including hours and minutes.It's used for various tasks like setting alarms, specifying event times, or choosing time-based preferences.You can use the TimePickerDialog to create and display a TimePicker in your Android app.
4. **DateTimePicker:** A DateTimePicker is a combination of both a DatePicker and a TimePicker, allowing the user to select a specific date and time. This is useful for events and tasks that require both date and time information, such as scheduling meetings or creating event entries.
5. **ListView:** A ListView is a UI component used to display a list of items in a scrollable vertical list.It's a fundamental component for displaying dynamic or static lists of data, such as contacts, messages, or items in an online store. You can use an ArrayAdapter or a custom adapter to populate the ListView with data.
6. **Spinner (SpinnerView):** A Spinner is a UI component that displays a drop-down list of items for the user to choose from. It's commonly used for selecting one option from a set of options, like choosing a country, sorting options, or selecting a category. In Android, you can use the Spinner widget and populate it with data using an ArrayAdapter or a custom adapter.

These UI components are essential for creating user-friendly Android applications, and they provide the means to input, display, and interact with various types of data. Depending on your app's requirements, you can use one or more of these components to enhance the user experience and make your app more intuitive and functional.

1. **Explain about saving data to internal and external storage?**

* **In Android,** we can save data to both internal and external storage, and each has its own characteristics and use cases. Here's an explanation of saving data to internal and external storage:
* **Internal Storage:** Internal storage is a private storage area that's associated with your app. It's not accessible by other apps, making it a secure place to store sensitive data. Data saved in internal storage is automatically removed when the user uninstalls your app. You can use internal storage for storing app-specific data, such as user preferences, configurationfiles, and cached data.
* **Example of Saving Data to Internal Storage:**

|  |
| --- |
| String data = "This is some internal storage data.";  try {  FileOutputStream fos = openFileOutput("myfile.txt", Context.MODE\_PRIVATE);  fos.write(data.getBytes());  fos.close();  } catch (IOException e) {  e.printStackTrace();  } |

* **External Storage:** External storage refers to the shared storage area that can be accessed by both your app and other apps. It's typically used for storing files that are meant to be shared or accessed by other apps. Data saved to external storage remains available even if the app is uninstalled, providing a way to store data more permanently. You should request the appropriate permissions to write to external storage and be mindful of the user's storage preferences and available space.
* **Example of Saving Data to External Storage:**

|  |
| --- |
| String data = "This is some external storage data.";  String fileName = "myfile.txt";  File externalDir = Environment.getExternalStorageDirectory();  File file = new File(externalDir, fileName);  try {  FileOutputStream fos = new FileOutputStream(file);  fos.write(data.getBytes());  fos.close();  } catch (IOException e) {  e.printStackTrace();  } |

It's important to note that starting with Android 10 (API level 29), external storage access is restricted for privacy and security reasons. You should use the recommended alternatives like the MediaStore for media files and scoped storage for app-specific data. Scoped storage provides a way to access shared storage while maintaining user privacy.

1. **Discuss about saving and loading user preferences?**

* **Saving and loading user preferences** is an essential part of developing user-friendly applications. It allows you to personalize the user experience and adapt to their individual needs and choices. Here's a comprehensive discussion on saving and loading user preferences:
* **Saving user preferences offers several advantages:**
* **Personalized Experience:** It enables tailored experiences by allowing users to adjust the application to their liking, fostering a sense of ownership and satisfaction.
* **Convenience:** By remembering user preferences, the application saves users the hassle of repeatedly setting their choices each time they use the app.
* **Enhanced User Engagement:** Personalized experiences can lead to increased user engagement, as users are more likely to interact with an application that adapts to their preferences.
* **Improved Accessibility:** Saving preferences can make applications more accessible for users with disabilities by allowing them to customize settings to suit their needs.
* **Methods for saving user preferences:**
* **SharedPreferences:** SharedPreferences is a lightweight key-value storage mechanism provided by the Android framework, ideal for storing small amounts of primitive data like integers, strings, and booleans.
* **Database:** For structured data like lists or complex objects, a database like SQLite is a more suitable option. It provides efficient storage and retrieval of structured data.
* **Cloud Storage:** For preferences that need to be synchronized across multiple devices or backed up, cloud storage services like Firebase or Google Cloud can be utilized.
* **Loading User Preferences:** Once preferences are saved, they need to be loaded when the application starts or when a user logs in to restore their customized settings. This process typically involves retrieving the stored preference values and applying them to the appropriate application elements.
* **Best Practices for Saving and Loading Preferences**
* **Minimize Data Storage:** Avoid storing excessive data, especially for infrequently accessed preferences, to conserve storage space and optimize app performance.
* **Handle Preference Updates**: Implement mechanisms to handle preference updates, such as notifying affected components or updating UI elements when preferences change.
* **Provide User Control:** Allow users to review, modify, or reset their preferences easily to maintain transparency and user control over their experience.
* **Consider Privacy Implications:** Be mindful of the sensitivity of user preferences and implement appropriate security measures to protect user privacy.
* **Utilize Preference Libraries:** Leverage existing preference libraries or frameworks to simplify the process of saving, loading, and managing user preferences.

1. **Discuss about:**
2. **Accessing Preferences:** To access user preferences, the application needs to obtain the SharedPreferences object associated with the preference file. This can be done using the getSharedPreferences() method, which takes the name of the preference file as a parameter. The following code snippet demonstrates accessing preferences:

|  |
| --- |
| SharedPreferences sharedPreferences = getSharedPreferences("myPreferences", MODE\_PRIVATE);   * This code retrieves the SharedPreferences object associated with the preference file named "myPreferences". The MODE\_PRIVATE parameter indicates that the preference file should be private to the application. |

1. **Retrieving Preferences:** Once the SharedPreferences object is obtained, you can retrieve stored preference values using the appropriate method for the data type. For instance, to retrieve a stored integer value:

|  |
| --- |
| int savedValue = sharedPreferences.getInt("savedValueKey", 0);   * This code retrieves the integer value associated with the key "savedValueKey" and stores it in the savedValue variable. If the key does not exist, the default value of 0 is returned. |

1. **Modifying Preferences:** To modify preferences, you need to use the edit() method of the SharedPreferences object to obtain an Editor object. The Editor object provides methods for adding, modifying, or removing preference values. For example, to modify an existing integer value:

|  |
| --- |
| SharedPreferences.Editor editor = sharedPreferences.edit();  editor.putInt("savedValueKey", newSavedValue);  editor.apply();   * This code updates the integer value associated with the key "savedValueKey" to the new value newSavedValue. The apply() method commits the changes to the SharedPreferences object. |

1. **Explain Pending Intents in details?**

* **A PendingIntent** is a special type of Intent that represents an action that will be executed at a future time. It is a token that allows one application to request another application to perform an action on its behalf. Pending Intents are commonly used to start activities, services, or broadcast receivers.
* **How Pending Intents Work:** When an application creates a PendingIntent, it is actually creating a reference to a token managed by the Android system. This token contains all of the information necessary to execute the action, such as the component to launch, the data to pass to the component, and the flags that control how the action is executed. When the PendingIntent is sent to another application, the Android system takes care of delivering the token to the appropriate component. The component then executes the action specified by the token.
* **Benefits of Using Pending Intents:**
* **Decoupling:** Pending Intents allow applications to decouple their code from other applications. This makes it easier to develop and maintain applications, as each application can focus on its own functionality without worrying about the details of how other applications work.
* **Flexibility:** Pending Intents can be used to execute a wide variety of actions, including starting activities, services, and broadcast receivers. This makes them a very versatile tool for developing applications.
* **Security:** Pending Intents can be used to enforce security policies. For example, an application can use a PendingIntent to start an activity that requires certain permissions. This ensures that only applications with the appropriate permissions can start the activity.
* **Types of Pending Intents:**
* **Mutable Pending Intents:** Mutable Pending Intents can be modified after they have been created. This means that the action associated with the PendingIntent can be changed, or the data that is passed to the component can be changed.
* **Immutable Pending Intents:** Immutable Pending Intents cannot be modified after they have been created. This means that the action associated with the PendingIntent cannot be changed, and the data that is passed to the component cannot be changed.
* **Common Uses of Pending Intents:**
* **Starting activities:** Pending Intents can be used to start activities from other applications, such as notifications or shortcuts.
* **Starting services:** Pending Intents can be used to start services from other applications, such as background tasks or timers.
* **Sending broadcasts:** Pending Intents can be used to send broadcasts from other applications, such as system events or notifications.
* **Example of Using a Pending Intent:**

|  |
| --- |
| Intent intent = new Intent(this, SecondActivity.class);  PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);  // Send the PendingIntent to another application |

|  |
| --- |
| https://miro.medium.com/v2/resize:fit:700/1*HzvWzmyx6LJMGgNEI1WQqg.jpeg |

1. **Write short notes on Broadcast Receivers?**

* **Broadcast receiver** are one of the four core android components. Simply, Broadcast Receivers can send or receive messages from other applications or from the system itself. These messages can be events or intents. For instance, Android system sends broadcasts when system events occur such as system boots up, device starts charging, connetivity chaning, date chaning, low battery. Furthermore, apps can send custom broadcasts to notify other apps(data download completed).
* **Types of Broadcasts:**
* **System Broadcasts:** Generated by the Android system itself, indicating system-level events like device boot, connectivity changes, or battery level updates.
* **Custom Broadcasts:** Sent by applications to notify other applications about internal events, such as new messages received or tasks completed.

1. **What is mean by Adapter? List its Types? Explain any 2 Adapter?**

* **An adapter** is a crucial component used to bind data to user interface elements, such as ListView, RecyclerView, and Spinner. Adapters act as a bridge between the data source and the UI elements, allowing you to display data in a user-friendly format. There are several types of adapters in Android, with ArrayAdapter, BaseAdapter, and RecyclerView.Adapter being some of the most commonly used ones.
* **Types of Adapter:**
* **ArrayAdapter:** ArrayAdapter is used to bind data from an array or a List to a ListView or Spinner. It's a simple and straightforward adapter for basic use cases. You provide an array or a List of data to the ArrayAdapter, and it takes care of displaying that data in the UI.
* **BaseAdapter:** BaseAdapter is an abstract class that serves as the base for creating custom adapters. It is more flexible than ArrayAdapter and allows you to create custom adapters for complex UI elements. You need to implement methods like getCount(), getItem(), and getView() when extending BaseAdapter.
* **CursorAdapter:** CursorAdapter is used when you want to display data from a SQLite database cursor in a ListView. It's suitable for working with data retrieved from a database query.
* **RecyclerView.Adapter:** RecyclerView is a more modern and flexible replacement for ListView. The RecyclerView.Adapter is used to bind data to a RecyclerView. It requires implementing methods like onCreateViewHolder(), onBindViewHolder(), and getItemCount() to manage the view recycling and data binding.
* **PagerAdapter:** PagerAdapter is specifically designed for ViewPager, which is used to implement swipeable views, such as image galleries or tabbed interfaces. It manages the content of individual pages.

1. **What is mean by Notification in android? Write its code?**

* **A notification** is a message that appears on an Android device to alert the user about an event or update. It typically includes a title, a message, and an icon. Users can tap on a notification to open the associated app or take an action directly from the notification, such as replying to a message or dismissing an alarm.
* **Purpose of Notifications:**
* **Alert Users:** They inform users about new messages, events, or updates from apps, allowing them to stay informed and engaged.
* **Prioritize Information:** Notifications can be prioritized based on their importance, ensuring that critical information is brought to the user's attention promptly.
* **Components of a Notification:**
* **Icon:** A small icon that represents the app or the type of notification.
* **Title:** A brief description of the notification's content.
* **Message:** More detailed information about the notification.
* **Actions:** Buttons that allow users to take immediate actions without opening the app.
* **Priority:** Notifications can be prioritized based on their importance, affecting their visibility, sound, and vibration behavior.
* **Creating a Notification:**

|  |
| --- |
| NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL\_ID)  .setSmallIcon(R.drawable.ic\_notification)  .setContentTitle("My Notification")  .setContentText("This is a simple notification.")  .setPriority(NotificationCompat.PRIORITY\_DEFAULT);  NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);  notificationManager.notify(1, builder.build()); |

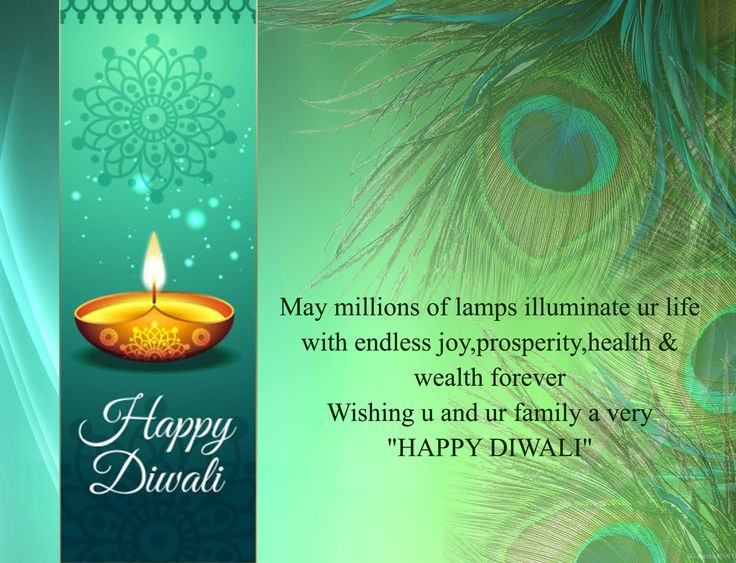
1. **Explain Dialogs with its types?**

|  |
| --- |
|  |
|  |
|  |

* **A dialog** is a small window that appears on top of the current activity to provide information, gather input, or prompt the user for a decision. Dialogs are a fundamental UI component used for various purposes in Android apps. There are different types of dialogs available, each suited for specific use cases. Here are the common types of dialogs in Android:

1. **Alert dialog:** An alert dialog is a simple pop-up window that can display a title, a message, and one or more buttons. It's often used for presenting information or getting user confirmation. To create and show an alert dialog: **One Button, Two Button, Three Button.**
2. **Date pickers and time pickers:** Date pickers and time pickers allow users to select a date or time. You can use DatePickerDialog and TimePickerDialog in your app to display these pickers and retrieve the selected date or time.
3. **Custom dialogs:** Custom dialogs allow you to create pop-up windows with a customized layout. You can design the dialog's layout in an XML file and then inflate it in your Java code.

**(PPT Give you more understanding than PDF)** The material for the PDF has been compiled from various sources such as books, tutorials (offline and online), lecture notes, several resources available on Internet. The information contained in this PDF is for general information and education purpose only. While we endeavor to keep the information up to date and correct, **we make no representation of any kind about the completeness and accuracy of the material.** The information shared through this PDF material should be used for educational purpose only.



**Thank you…**