ANDROID PRACTICAL FILE

1. Installation and setup of java development kit (JDK), setup android SDK, setup eclipse IDE, setup android development tools (ADT) plugins, create android virtual device.

1. Java Development Kit (JDK) Installation and Setup
The JDK is essential for developing Android apps using Java. Here are the steps to install and set up the JDK:
- **Download JDK:** Visit the Oracle website and download the latest version of the JDK suitable for your system.
- **Install JDK:** Run the downloaded installer and follow the prompts to install the JDK. Typically, it installs in a directory like `C:\Program Files\Java\jdk-`.
- **Verify Installation:** Open a Command Prompt and type `javac -version` to verify that the JDK is installed correctly.

2. Android SDK Setup
The Android SDK provides the necessary tools and libraries for Android app development. Here's how to set it up:
- **Download Android Studio or SDK:** While Android Studio includes the SDK, you can also download the SDK separately. However, using Android Studio is recommended as it simplifies the setup process.
- **Install SDK Components:** Launch Android Studio, and it will prompt you to install essential SDK components like Platform-tools, Build-tools, and SDK Tools through the SDK Manager.
- **Verify SDK Installation:** Use the SDK Manager to ensure all required packages are installed and up-to-date.

3. Eclipse IDE Setup
Eclipse is another popular IDE for Android development, especially before Android Studio became the standard. Here's how to set it up:
- **Download Eclipse: Get** the Eclipse IDE for Java EE Developers or the Classic version.
- **Install Eclipse:** Run the installer and follow the prompts.
- **Launch Eclipse:** Choose a workspace directory when prompted.

4. Android Development Tools (ADT) Plugins Setup
To integrate Android development capabilities into Eclipse, you need the ADT plugins:
- **Install ADT Plugin:**
  - Open Eclipse and go to `Help` > `Install New Software`.
  - Add the ADT repository URL (`https://dl-ssl.google.com/android/eclipse/`) and select the Developer Tools.
  - Install the plugin and restart Eclipse.
- **Configure ADT Plugin:**
  - Go to `Window` > `Preferences` > `Android`.
  - Specify the path to your Android SDK installation.

5. Creating an Android Virtual Device (AVD)
AVDs are used to test and debug Android apps without a physical device. Here's how to create one:
- **Using Android Studio:**
  - Open Android Studio and go to `Tools` > `Android` > `AVD Manager`.
  - Click `Create Virtual Device` and select a device definition.
  - Choose a system image and configure other settings like RAM and storage.

- Click `Finish` to create the AVD.

- Using Eclipse:

- Go to `Window` > `Android Virtual Device Manager`.
- Click `New` and enter AVD details like name, device, target, and hardware settings.
- Click `OK` to create the AVD.

---

2. Create “Hello World” application. That will display “Hello World” in the middle of the screen using TextView Widget in the red color.

1. Project Setup: Theoretically, you would start by setting up a new Android project. This involves creating a new project in Android Studio and selecting the appropriate project template, such as "Empty Activity."

2. User Interface Design:

- Layout File: You would design the user interface by editing the layout file (`activity_main.xml`).

- TextView Widget: Add a `TextView` widget to the layout. This widget is used to display text on the screen.

- Centering the Text: To center the text both horizontally and vertically, you would use layout attributes like `android:layout_centerHorizontal="true"` and `android:layout_centerVertical="true"` within a `RelativeLayout`.

- Text Color: Change the text color to red by setting `android:textColor` to a red color code (e.g., `#FF0000`).

- Text Content: Set the text content using `android:text`, referencing a string resource defined in `strings.xml`.

3. String Resource:

- Define a string resource named `hello_world` in `strings.xml` with the value "Hello World!".

4. Activity Code:

- In the main activity class (`MainActivity.java`), ensure that the layout is set using `setContentView(R.layout.activity_main)` in the `onCreate` method.

5. Running the Application:

- Theoretically, once the project is set up and the layout and code are configured, you would run the application on an emulator or a physical device to see the "Hello World!" text displayed in red, centered on the screen.

---

3. Create Registration page to demonstration of Basic widgets available in android.

1. Planning the Layout

- Choose a Layout: Decide on a layout structure that can accommodate all the necessary widgets. Common layouts include `LinearLayout`, `RelativeLayout`, and `ConstraintLayout`. Each has its advantages depending on the complexity and desired positioning of elements.

- Identify Required Widgets: Determine which basic widgets are needed for a registration form, such as `EditText` for input fields, `TextView` for labels, `Button` for submission, and possibly `CheckBox` or `RadioButton` for additional options.

2. Designing the UI
- **Add Input Fields:** Use `EditText` widgets to create fields for user input, such as username, email, password, and confirm password. You can customize these fields with hints, input types (e.g., email, password), and other attributes.
- **Add Labels:** Use `TextView` widgets to provide labels for each input field, making it clear what information is required.
- **Add Buttons:** Include a `Button` for submitting the registration form. You might also include a "Cancel" or "Back" button depending on your app's flow.
- **Optional Widgets:** Consider adding `CheckBox` or `RadioButton` widgets if your registration form requires additional options, such as agreeing to terms or selecting a user type.

3. Implementing Validation and Logic
- **Input Validation:** Implement checks to ensure that users enter valid data, such as checking for email format or ensuring passwords match.
- **Submission Logic:** Handle the button click event to process the registration data. This might involve sending data to a server, storing it locally, or triggering other actions within your app.

4. Enhancing User Experience
- **Responsive Design:** Ensure that your layout adapts well to different screen sizes and orientations.
- **Accessibility:** Consider accessibility features like text size adjustment and screen reader compatibility.
- **Feedback**: Provide visual or auditory feedback when users interact with widgets, such as highlighting fields with errors or displaying a progress bar during submission.

5. Testing and Iteration
- **Test on Various Devices:** Verify that your registration page works as expected across different devices and Android versions.
- **Gather Feedback:** Collect user feedback to identify areas for improvement and refine the design and functionality accordingly.

---

4. Create sample application with login module.(Check username and password) On successful login, Change TextView "Login Successful". And on failing login, alert user using Toast "Login fail".

1. Setting Up the Project
- **Launch Android Studio:** Start by opening Android Studio to create a new Android project.
- **Create a New Project:** Choose the "Empty Activity" template to create a basic project structure.
- **Configure Project Settings:** Enter an application name, choose a company domain for the package name, select the programming language (Java or Kotlin), and set the minimum SDK to ensure compatibility with various Android versions.

2. Designing the User Interface
- **Open the Layout File:** Modify the `activity_main.xml` file to design the UI. This includes adding:
  - **EditText Widgets:** For entering the username and password. Ensure the password field has an input type set to `textPassword`.
  - **TextView Widget:** To display the login status.
  - **Button Widget:** For submitting the login credentials.

3. Implementing Login Logic
- **Handle Button Click:** Set an `OnClickListener` for the login button. When clicked, retrieve the text from the `EditText` widgets.

- **Validate Credentials**: Compare the entered username and password with predefined or stored credentials. If they match, update the `TextView` to display "Login Successful".
- **Display Error Message:** If the credentials do not match, use a `Toast` message to alert the user with "Login fail".

4. Enhancing Security and User Experience
- **Input Validation:** Implement checks to ensure that users enter valid data (e.g., checking for empty fields).
- **Security Measures:** Consider using secure methods to store and compare passwords, such as hashing.
- **Feedback Mechanism:** Provide immediate feedback to the user upon login attempts, such as changing the text color or displaying a progress bar.

5. Testing and Iteration
- **Test on Various Devices:** Verify that the login functionality works correctly across different devices and Android versions.
- **Gather Feedback:** Collect user feedback to identify areas for improvement and refine the design and functionality accordingly.

---

5. Create an application for demonstration of Scroll view in android.

1. Understanding ScrollView
- **Purpose**: A `ScrollView` is a view group that allows its child views to be scrolled when the content exceeds the layout size. It supports vertical scrolling by default.
- **Limitations:** A `ScrollView` can only have one direct child. To include multiple views, you must wrap them in a `ViewGroup` like `LinearLayout`.

2. Designing the UI
- **Add a ScrollView:** Insert a `ScrollView` into your layout file (`activity_main.xml`).
- **Add a ViewGroup:** Place a `LinearLayout` (or another `ViewGroup`) inside the `ScrollView` to hold multiple views.
- **Add Views:** Include several views (e.g., `TextView`, `Button`) inside the `LinearLayout` to demonstrate scrolling.

3. Configuring the Layout
- **Orientation:** Set the `LinearLayout` orientation to vertical for vertical scrolling.
- **Content Size:** Ensure the content is larger than the screen size to enable scrolling.

4. Implementing Horizontal Scrolling (Optional)
- **Use HorizontalScrollView:** If you want to demonstrate horizontal scrolling, replace `ScrollView` with `HorizontalScrollView`.
- **Adjust Layout:** Change the `LinearLayout` orientation to horizontal for horizontal scrolling.

5. Testing the Application
- **Run the App:** Deploy the app on an emulator or physical device.
- **Verify Scrolling:** Test that the content scrolls as expected when the user interacts with the screen.

6. Enhancing the Application
- **Customization:** Explore additional attributes of `ScrollView`, such as hiding the scrollbar or styling it.

- **Complex Layouts:** Consider integrating `ScrollView` with other scrollable views, though this can introduce complexity due to touch event handling.

---

6. Create login application where you will have to validate username and passwords till the username and password is not validated, login button should remain disabled.

1. Designing the User Interface
- **EditText Widgets:** Use two `EditText` widgets to input the username and password. Ensure the password field has an input type set to `textPassword`.
- **TextView Widgets:** Add labels for the username and password fields.
- **Button Widget:** Include a login button that will be initially disabled.

2. Implementing Validation Logic
- **Text Watchers:** Use `TextWatcher` to monitor changes in the `EditText` fields. This allows you to validate the input as the user types.
- **Validation Rules:** Define rules for validating the username and password. For example, the username might need to be at least 4 characters long, and the password might require a mix of uppercase, lowercase, and numbers.
- **Enable/Disable Button:** Based on the validation results, enable the login button only when both fields meet the validation criteria.

3. Handling Login Button Click
- **OnClickListener:** Set an `OnClickListener` for the login button. When clicked, retrieve the text from the `EditText` widgets and compare it with stored or predefined credentials.
- **Authentication Logic:** Implement logic to authenticate the user. This could involve checking against a local database or making a network request to a server.

4. Providing Feedback
- **Error Messages:** Display error messages if the validation fails or if the login attempt is unsuccessful. This can be done using `Toast` messages or by setting error text directly on the `EditText` fields.
- **Success Feedback**: Provide feedback when the login is successful, such as displaying a success message or navigating to a new activity.

5. Security Considerations
- **Password Storage:** Ensure that passwords are stored securely, ideally using hashing and salting techniques.
- **Input Validation:** Always validate user input to prevent potential security vulnerabilities like SQL injection or cross-site scripting (XSS).

6. Testing and Iteration
- **Test on Various Devices**: Verify that the login functionality works correctly across different devices and Android versions.
- **Gather Feedback:** Collect user feedback to identify areas for improvement and refine the design and functionality accordingly.

---

7. Create an application for calculator.

1. Designing the User Interface
- **Layout Structure:** Use a layout like `LinearLayout` or `GridLayout` to organize the calculator buttons and display.
- **Buttons:** Include buttons for digits (0-9), operators (+, -, *, /), and special functions (clear, equals).
- **TextView or EditText:** Use a `TextView` or `EditText` to display the current calculation and result.

2. Implementing Button Click Logic
- **OnClickListener:** Set an `OnClickListener` for each button to handle clicks.
- **Digit Buttons:** When a digit button is clicked, append the digit to the current calculation string.
- **Operator Buttons:** When an operator button is clicked, append the operator to the calculation string.
- **Equals Button**: When the equals button is clicked, evaluate the calculation string to display the result.
- **Clear Button:** When the clear button is clicked, reset the calculation string and display.

3. Handling Calculations
- **Expression Evaluation:** Use a method to evaluate the mathematical expression. This could involve parsing the string and performing operations based on operator precedence.
- **Error Handling:** Implement checks to handle invalid expressions or division by zero.

4. Enhancing Functionality
- **Advanced Operations**: Consider adding buttons for advanced operations like square root, logarithm, or trigonometric functions.
- **Memory Functions:** Add buttons for memory operations (e.g., storing and recalling values).

5. Improving User Experience
- **Responsive Design:** Ensure the layout adapts well to different screen sizes and orientations.
- **Feedback Mechanism:** Provide visual or auditory feedback when buttons are pressed.

6. Testing and Iteration
- **Test on Various Devices:** Verify that the calculator works correctly across different devices and Android versions.
- **Gather Feedback:** Collect user feedback to identify areas for improvement and refine the design and functionality accordingly.

---

8. Demonstrate use of scroll view.

1. Understanding ScrollView
- **Purpose:** A `ScrollView` is a view group that allows its child views to be scrolled when the content exceeds the screen's dimensions. It supports vertical scrolling by default.
- **Limitations**: A `ScrollView` can only have one direct child. To include multiple views, you must wrap them in a `ViewGroup` like `LinearLayout`.

2. Designing the UI
- **Add a ScrollView:** Insert a `ScrollView` into your layout file (`activity_main.xml`).
- **Add a ViewGroup:** Place a `LinearLayout` (or another `ViewGroup`) inside the `ScrollView` to hold multiple views.

- **Add Views:** Include several views (e.g., `TextView`, `Button`) inside the `LinearLayout` to demonstrate scrolling.

3. Configuring the Layout
- **Orientation:** Set the `LinearLayout` orientation to vertical for vertical scrolling.
- **Content Size:** Ensure the content is larger than the screen size to enable scrolling.

4. Customizing ScrollView
- **Attributes:** Use attributes like `android:layout_gravity`, `android:padding`, and `android:background` to customize the appearance and positioning of the `ScrollView`.
- **Accessibility:** Provide accessibility information using `android:contentDescription` for screen readers.

5. Handling Nested Scrollable Views
- **Challenge:** When using a scrollable view inside another scrollable view, touch events can be intercepted by the parent view, preventing the child view from scrolling.
- **Solution:** Implement custom touch event handling to ensure both views scroll as expected.

6. Testing and Iteration
- **Run the App:** Deploy the app on an emulator or physical device.
- **Verify Scrolling:** Test that the content scrolls as expected when the user interacts with the screen.

---

9. Demonstrate use of intent in android.

1. Understanding Intents
- **Purpose:** An intent is a messaging object used to request actions from other app components, such as activities, services, or broadcast receivers. It facilitates communication between different parts of an application or even between different applications.
- **Types of Intents:**

There are two main types of intents:
  - **Explicit Intents:** These specify the exact component to handle the action, typically used for internal communication within the same application.
  - **Implicit Intents:** These declare the action to be performed without specifying a target component. The Android system resolves the intent and finds the most suitable component to handle it based on intent filters defined in the manifest.

2. Using Explicit Intents
- **Launching Activities:** Use explicit intents to start activities within your own application. For example, moving from one activity to another.
- **Example:** An explicit intent can be created by specifying the target activity class, such as `Intent intent = new Intent(FirstActivity.this, SecondActivity.class);`.

3. Using Implicit Intents
- **Performing Actions:** Use implicit intents to perform actions like opening a web page, sending an email, or making a phone call. The system will prompt the user to choose an app if multiple apps can handle the action.
- **Example:** An implicit intent can be created by specifying an action like `Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("https://www.example.com"));`.

4. Passing Data with Intents
- **Extras:** Intents can be used to pass data between components by adding extras to the intent. This is useful for sending information from one activity to another.
- **Example:** You can add a string extra to an intent using `intent.putExtra("key", "value");`.

5. Handling Intent Results
- **startActivityForResult:** Use this method to start an activity and receive a result back. This is useful for scenarios like selecting a contact or taking a photo.
- **onActivityResult**: Implement this method in your activity to handle the result returned from another activity.

6. Security Considerations
- **Intent Filters:** Ensure that intent filters are properly defined in your app's manifest to control which actions your components can handle.
- **Data Validation:** Always validate data received from intents to prevent potential security vulnerabilities.

---

10. Create application to demonstrate menu option.

1. Understanding Menu Types
- **Options Menu:** This is the primary menu type used for actions relevant to the current activity context, such as settings or search.
- **Context Menu:** A floating menu that appears when a view is long-pressed, typically used for actions specific to the selected item.
- **Submenu:** A menu nested within another menu item, useful for organizing related actions.

2. Designing the Menu
- **Create a Menu Resource**: Define your menu in an XML file within the `res/menu` directory. Use `` tags to specify menu items, including attributes like `android:id`, `android:title`, and `android:icon`.
- **Inflate the Menu:** Override the `onCreateOptionsMenu` method in your activity to inflate the menu resource using `MenuInflater`.

3. Handling Menu Item Selection
- **Override onOptionsItemSelected:** Implement this method to handle menu item clicks. Use the `MenuItem` object to identify which item was selected and perform the corresponding action.
- **Launch Activities or Intents**: Use menu items to launch activities or send intents to other apps. For example, a "Share" menu item might use an intent to open a sharing dialog.

4. Dynamic Menu Items
- **Add Items Based on Intents:** Use `Menu.addIntentOptions` to dynamically add menu items based on available activities that can handle specific intents. This ensures that only functional menu items are displayed.

5. Customizing Menu Appearance
- **Themes and Styles**: Apply themes or styles to customize the appearance of your menu, ensuring it aligns with your app's design.
- **Icons and Text:** Use icons and text to make menu items more intuitive and visually appealing.

6. Testing and Iteration
- Test on Various Devices: Verify that the menu works correctly across different devices and Android versions.
- Gather Feedback: Collect user feedback to identify areas for improvement and refine the design and functionality accordingly.

---

11. Create application to demonstrate progress bar.

1. Understanding ProgressBar
- Purpose: A `ProgressBar` is a graphical view indicator that shows the progress of an operation, such as downloading or uploading files.
- Types:
There are two main types of progress bars:
 - Horizontal ProgressBar: Displays a horizontal bar representing the progress.
 - Circular ProgressBar: Displays a spinning wheel indicating progress without a specific percentage.

2. Designing the UI
- Add a ProgressBar: Insert a `ProgressBar` into your layout file (`activity_main.xml`).
- Configure Attributes: Set attributes like `android:max` to define the maximum progress value (default is 100), `android:progress` to set the initial progress, and `android:indeterminate` to choose between determinate and indeterminate modes.
- Style: Use styles like `?android:attr/progressBarStyleHorizontal` for a horizontal bar or the default style for a circular bar.

3. Implementing Progress Updates
- Thread or AsyncTask: Use a background thread or `AsyncTask` to perform operations that require progress updates, such as downloading files.
- Update Progress: Use the `setProgress` method to update the progress bar's value as the operation progresses.

4. Using ProgressDialog
- Alternative Approach: Instead of a standalone `ProgressBar`, you can use a `ProgressDialog` to display a progress bar within a dialog. This is useful for operations like downloading or uploading files.
- ProgressDialog Attributes: Set properties like `setMessage`, `setProgressStyle`, and `setIndeterminate` to customize the dialog's appearance and behavior.

5. Enhancing User Experience
- Feedback Mechanism: Provide visual feedback to the user as the progress updates, ensuring they are informed about the operation's status.
- Responsive Design: Ensure the layout adapts well to different screen sizes and orientations.

6. Testing and Iteration
- Test on Various Devices: Verify that the progress bar works correctly across different devices and Android versions.
- Gather Feedback: Collect user feedback to identify areas for improvement and refine the design and functionality accordingly.
