FINAL NOTES

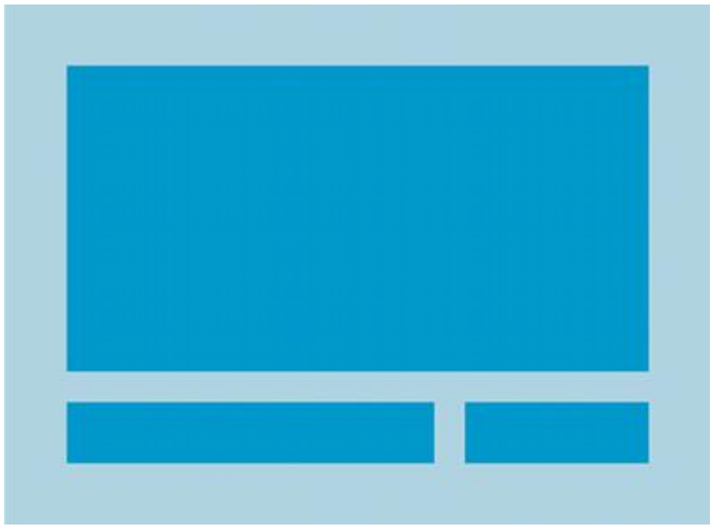
COURSE TITLE: MOBILE APP DEVELOPMENT

COURSE CODE: CS-512

FROM: MASK

RelativeLayout:

- view group
- displays child views in **relative positions**.
  - The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent RelativeLayout area (such as aligned to the bottom, left or center).



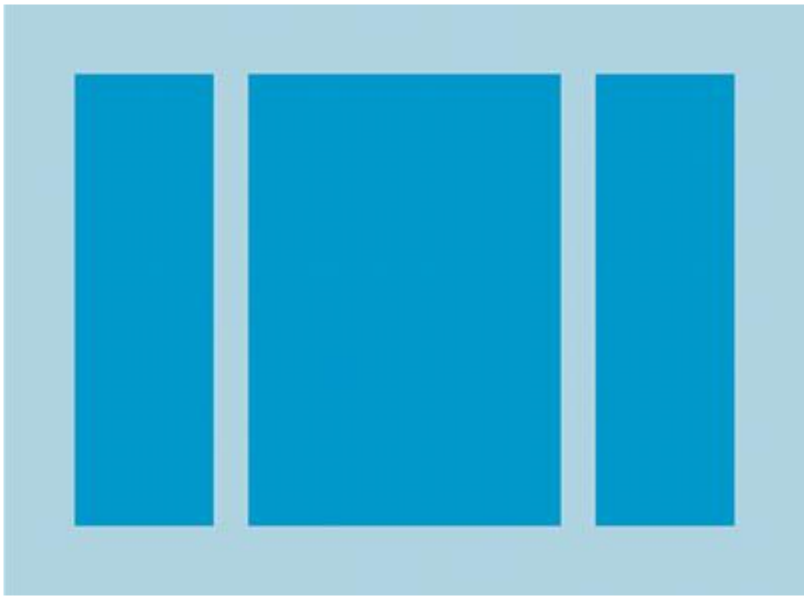
- powerful utility for designing a **user interface** & **improve performance**
  - because it can eliminate nested view groups and keep your layout hierarchy flat

RelativeLayout Properties:

- android:layout_alignParentTop
- android:layout_centerVertical
- android:layout_below
- android:layout_toRightOf

LinearLayout:

- view group
- aligns all children in a **single direction**, vertically or horizontally
- specify layout direction with the **android:orientation** attribute



- All children of a LinearLayout are **stacked** one after the other
    - A vertical list will only have one child per row, no matter how wide they are.
    - A horizontal list will only be one row high (the height of the tallest child, plus padding).
- Allow **margin** between children and the **gravity** (right, center, or left alignment) of each child.

EditText:

- **UI Element**
- Extends **TextView**
- For **entering** and **modifying text**
- Choose **input type** in order to **configures the keyboard type** that is shown, acceptable characters, and appearance of the edit text.
    - You can set **inputType** to “numericPassword” if you want to accept a secret number, like a unique pin or serial number.
      An inputType of “numericPassword” results in an edit text that accepts numbers only, shows a numeric keyboard when focused, and masks the text that is entered for privacy.
- Receive **callbacks as user changes text** by adding a **TextWatcher** to the edit text.
    - Useful for adding **auto-save functionality** as changes are made, or **validate** the format of user input.

- Does not support auto-sizing text

Button:

- **UI Element**
- User can tap or click to perform an **action**
- **Extends TextView**
- To specify an action when the button is pressed, set a **click listener** on the button object in the corresponding activity code:

```
public class MyActivity extends Activity {
     protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);

          setContentView(R.layout.content_layout_id);

          final Button button = findViewById(R.id.button_id);
          button.setOnClickListener(new View.OnClickListener() {
              public void onClick(View v) {
                  // Code here executes on main thread after user
presses button
              }
          });
     }
 }
```

  - The system executes the code written in **onClick(View)** after the user presses the button.

RadioButton:

- **UI Element**
- Extends **CompoundButton**
- **Two-states button**
  - Either checked or uncheck
- Unlike CheckBox, a radio button **cannot** be **unchecked** by the user once checked.
- Normally **used** together in a **RadioGroup**
  - When several radio buttons live inside a radio group, checking one radio button unchecks all the others.
- **Usage:**

  - **XML CODE:**

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

- **JAVA CODE:**

```
public void onRadioButtonClicked(View view) {
  // Is the button now checked?
  boolean checked = ((RadioButton) view).isChecked();

  // Check which radio button was clicked
  switch(view.getId()) {
    case R.id.radio_pirates:
      if (checked)
        // Pirates are the best
      break;
    case R.id.radio_ninjas:
      if (checked)
        // Ninjas rule
      break;
  }
}
```

CheckBox:

- **UI Element**
- Extends **CompoundButton**
- **Two-states button**
  - Either checked or unchecked
- **Usage:**

```
public class MyActivity extends Activity {
  protected void onCreate(Bundle icicle) {
    super.onCreate(icicle);

    setContentView(R.layout.content_layout_id);

    final CheckBox checkBox = (CheckBox) findViewById(R.id.checkbox_id);
    if (checkBox.isChecked()) {
      checkBox.setChecked(false);
    }
  }
}
```

View.OnClickListener:

- It is an **Interface**
- It is a **callback** to be executed when a view is clicked.
- It consists of a **public method "onClick"** which is called when a view has been clicked.
  - **onClick** method have a parameter of type **View** (The view that was clicked)
- **Example:**

```
public class MyActivity extends Activity {
     protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);

         setContentView(R.layout.content_layout_id);

         final Button button = findViewById(R.id.button_id);
         button.setOnClickListener(new View.OnClickListener() {
             public void onClick(View v) {
                 // Code here executes on main thread after user
presses button
             }
         });
     }
 }
```

Type Casting:

- Type casting is when you assign a value of one primitive data type to another type.
- In Java, there are two types of casting:

  - **Widening Casting** (automatically) - converting a smaller type to a larger type size
    byte -> short -> char -> int -> long -> float -> double

    Example:

```
public class Main {

  public static void main(String[] args) {

    int myInt = 9;

    double myDouble = myInt; // Automatic casting: int to
double


    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
  double -> float -> long -> int -> char -> short -> byte

  Example:

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to
int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

Exception Handling:

- **Exception** is **technical term** for **error**.
  - When an error occurs, Java program will normally stop and generate an error message.
- **try**
  - define a block of code to be tested for errors while it is being executed.
- **catch**
  - define a block of code to be executed, if an error occurs in the try block.
- **finally**
  - lets you execute code, after try...catch, regardless of the result

- **Example:**

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    } finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

- **throw**
  - allows you to create a custom error.
  - used together with an **exception type**.
  - **Exception types:** ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc.
  - **Example:**

```
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }

  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

Intents:

- A **messaging object** you can use to **request** an action from another app component.
- Intents facilitate communication between components in several ways, there are three fundamental **use cases**:
  - **Starting an activity**
  - **Starting a service**
  - **Delivering a broadcast**
- Intent **types:**
  - **Explicit Intents**

    Specify which application will satisfy the intent, by supplying either the **target app's package name** or a fully-qualified component **class name**.

    You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start.

    Example:
    You might start a new activity within your app in response to a user action, or start a service to download a file in the background.

    Code Example:

    ```
    // Executed in an Activity, so 'this' is the Context
    // The fileUrl is a string URL, such as "http://www.example.com/image.png"
    Intent downloadIntent = new Intent(this, DownloadService.class);
    downloadIntent.setData(Uri.parse(fileUrl));
    startService(downloadIntent);
    ```

  - **Implicit Intents**

    Do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.

    Example:

    if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Code Example:

```
// Create the text message with a string.
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Try to invoke the intent.
try {
   startActivity(sendIntent);
} catch (ActivityNotFoundException e) {
   // Define what your app should do if no activity can handle the intent.
}
```

- **Intent Filters:**

  - An intent filter is an **expression** in an **app's manifest file** that specifies the type of intents that the component would like to receive.
  - By declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent.
  - if you do *not* declare any intent filters for an activity, then it can be started only with an explicit intent.

- **Intent (Practical Example):**
  Starting a new activity and passing values using intent.

```
// This intent will transition user from MainActivity to SecondActivity
Intent intent = new Intent(MainActivity.this, SecondActivity.class);

//Passing values in intent so that we can access them in SecondActivity
intent.putExtra("DEGREE", "BSCS");
intent.putExtra("SEMESTER",6);

// Starting activity
startActivity(intent);
```

FIREBASE:

- Google-backed application development software
- Enables developers to develop iOS, Android and Web apps
- Provides tools for:
  - Tracking analytics

    - Reporting
    - Fixing app crashes
    - Creating marketing and product experiment
- Offers number of services:
    - Analytics
    - Authentication
    - Cloud Messaging
    - Realtime database
    - Crashlytics
    - Performance
    - Test lab

JSON:

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight format for storing and transporting data
- JSON is often used when data is sent from a server to a web page
- JSON is "self-describing" and easy to understand

Online Database:

- Firebase Database
- Back4app

Offline Database:

- SQLite
- ROOM DATABASE

FIREBASE DATABASE:

Firebase Realtime Database is a cloud-hosted database that supports multiple platforms Android, iOS and Web. All the data is stored in JSON format and any changes in the data, reflects immediately by performing sync across all the platforms & devices. This allows us to build more flexible realtime apps easily with minimal effort.