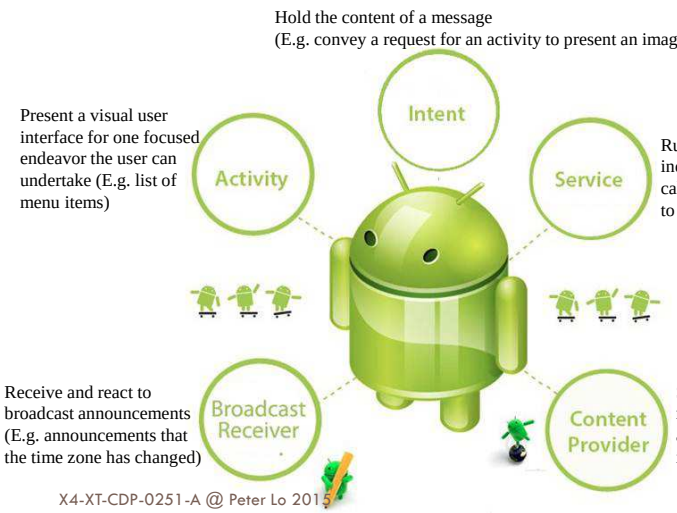
ANDROID APPS DEVELOPMENT FOR MOBILE AND TABLET DEVICE (LEVEL I)

Lecture 2: Android Programming Foundation

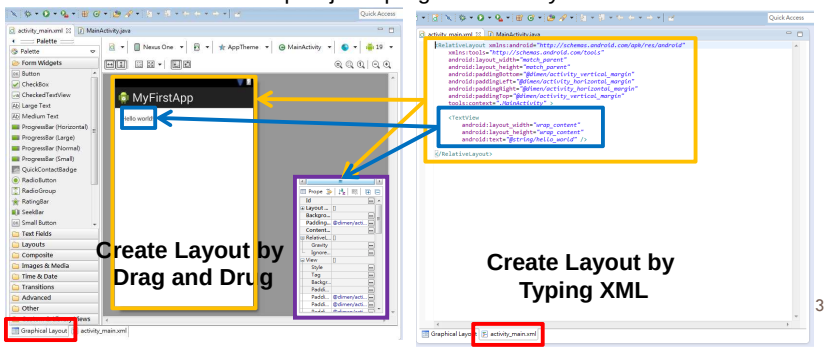
Peter Lo

Application Components



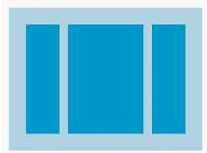
Simple Android App Layout

- A layout defines the visual structure for a user interface:
  - **Declare UI elements in XML.** Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
  - **Instantiate layout elements at runtime.** Your application can create View and ViewGroup objects programmatically.

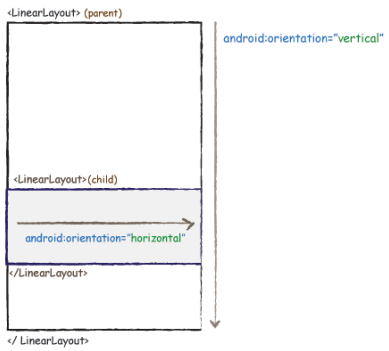


Linear Layout

- In a linear layout, all the elements are displayed in a linear fashion, either Horizontally or Vertically
- The properties is set in ***android:orientation*** which is an attribute of the node ***LinearLayout***.

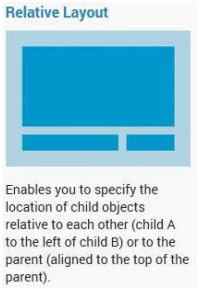


A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.



Relative Layout

- In a relative layout every element arranges itself relative to other elements or a parent element.
  - E.g. The “Cancel” button is placed relatively, to the right of the “Login” button parallel. Here is the code snippet that achieves the mentioned alignment



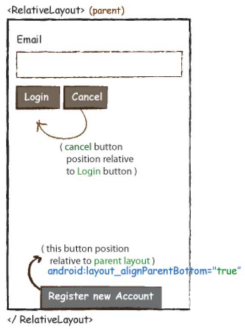
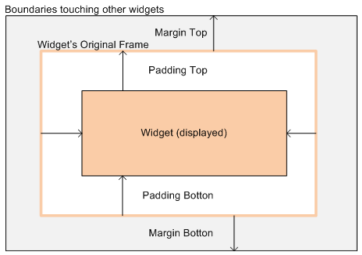
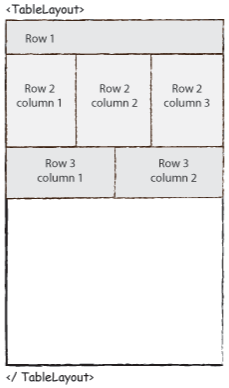


Table Layout

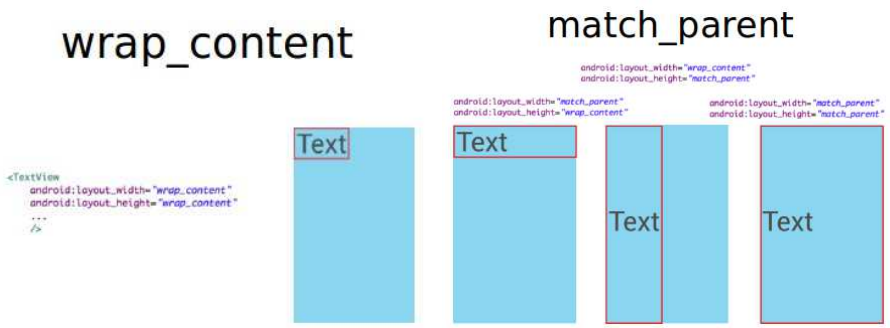
- Table layouts in Android works in the same way HTML table layouts work.
- You can divide your layouts into rows and columns.





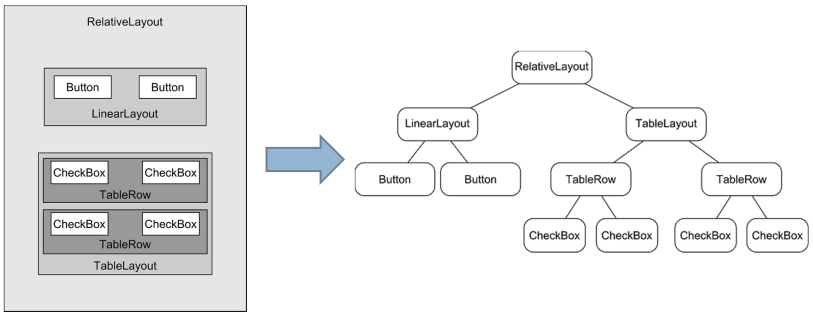
Layout Attributes

- **FILL_PARENT / MATCH_PARENT**, which means that the view wants to be as big as its parent (minus padding)
- **WRAP_CONTENT**, which means that the view wants to be just big enough to enclose its content (plus padding)



The View Hierarchy

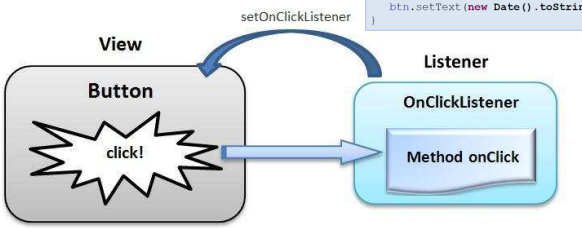
- The view hierarchy diagram gives probably the clearest overview of the relationship between the various views that make up the user interface shown. When a user interface is displayed to the user, the Android runtime walks the view hierarchy, starting at the root view and working down the tree as it renders each view.



Event Listener

- An object cannot process event on its own, it needs a listener for this.
  - E.g. When a button is clicked, listener reacts and runs the code from **onClick** method.

```
btn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        updateTime();
    }
});

private void updateTime() {
    btn.setText(new Date().toString());
}
```



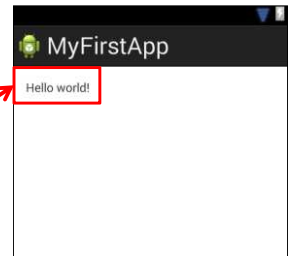
Common Event

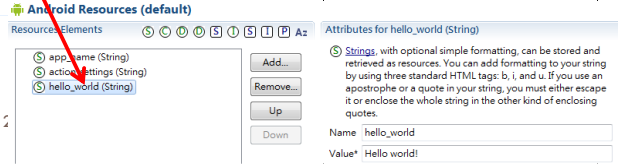
- onClick()
- onLongClick()
- onFocusChange()
- onKey()
- onTouch()
- onCreateContextMenu()

TextView (Label)

- A label is called in android a TextView.
- TextViews are not editable, typically used for output to display a caption.

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```





Button

- A Button widget allows the simulation of a clicking action on a GUI. Button is a subclass of TextView, it styled using the system's default button background.

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

        setContentView(R.layout.content_layout_id);

        final Button button = (Button) findViewById(R.id.button_id);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Perform action on click
            }
        });
    }
}
```

Request a reference to the button from the activity by calling *findViewById*.

Create a class implementing "OnClickListener" and set it as the on click listener for the button.

EditText (Text Field)

- EditText allows the user to type text (single text or multi-line) into your app.
- Touching a text field places the cursor and automatically displays the keyboard.

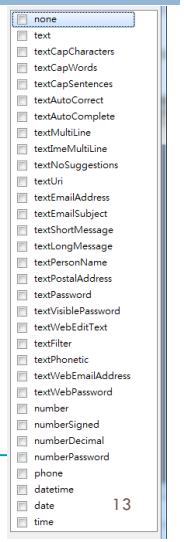
```
public class MainActivity extends Activity {
    EditText UserInput;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        UserInput = (EditText) findViewById(R.id.editText1);
        UserInput.setOnFocusChangeListener(new OnFocusChangeListener() {
            public void onFocusChange(View v, boolean hasFocus) {
                String strValue = UserInput.getText().toString();
            }
        });
    }
}
```

Request a reference to the button from the activity by calling *findViewById*.

Retrieve the EditText value and convert to string

Common InputType for EditText

- text hello
- number 123456
- phone (852) 2766-5111
- textMultiLine first row second row
- textCapCharacters HELLO
- textPassword ••••••••d
- textAutoCorrect teh ⇨ the



Radio Button

- Radio buttons allow the user to select one option from a set. You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side.
- To create each radio button option, create a ***RadioButton*** in your layout. However, because radio buttons are mutually exclusive, you must group them together inside a ***RadioGroup***. By grouping them together, the system ensures that only one radio button can be selected at a time

(●) RadioButton1
( ) RadioButton2
( ) RadioButton3

Handling Events in Radio Button

```
RadioButton radio0 = (RadioButton) findViewById(R.id.radio0);
RadioButton radio1 = (RadioButton) findViewById(R.id.radio1);

RadioGroup radioGroup1 = (RadioGroup) findViewById(R.id.radioGroup1);
radioGroup1.setOnCheckedChangeListener( new RadioGroup.OnCheckedChangeListener() {
        public void onCheckedChanged ( RadioGroup group, int checkedId ) {
                if ( checkedId == radio0.getId() ) {
                        Your Action ...
                } else if ( checkedId == radio1.getId() ) {
                        Your Action ...
                }
        }
} );
```


Checkbox

- Checkboxes allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.
- To create each checkbox option, create a ***CheckBox*** in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.

☐ CheckBox1
☐ CheckBox2

Handling Events in Checkbox

```
CheckBox checkbox1 = (CheckBox) findViewById(R.id.checkBox1);
if (checkbox1.isChecked()) {
                    Your Action ...
} else {
                    Your Action ...
}

CheckBox checkbox2 = (CheckBox) findViewById(R.id.checkBox2);
if (checkbox2.isChecked()) {
                    Your Action ...
} else {
                    Your Action ...
}
```


Toast (Message)

- Toast messages is a simple pop-up message for alerting users.
- The following code would generate a toast message:
  - ***Toast.makeText(Context, Text, Duration).show();***

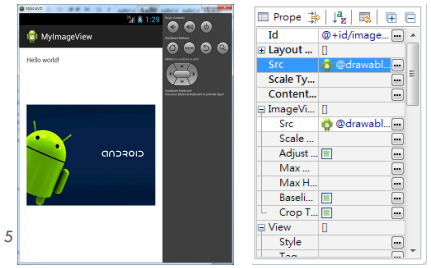
The context to use, usually your Application or Activity object by method *getApplicationContext()*

The text to show on screen

Duration for the message display:
- *LENGTH_SHORT* (2 seconds)
- *LENGTH_LONG* (3.5 seconds)

ImageView and ImageButton

- ImageView and ImageButton are two Android widgets that allow embedding of images in your applications.
- Both are image-based widgets analogue to TextView and Button, respectively.
- Each widget takes an android:src or android:background attribute (in an XML layout) to specify what picture to use.
- Pictures are usually a reference to a drawable resource.



Appendix 1: Variables

- A variable is a reserved location in your computer's memory that will allow the application to store data.
- It is often referred to as a container that we can pass data in to hold and then reuse the data throughout the application.
- Before using a variable we need to both **Declare** and **Instantiate** the variable

Declaring a variable

- When declaring a variable the Android system will reserve a portion of the system's memory so that it can be used to store data.
- Variables can vary in size and type, so when declaring a variable we also state which data type to use.
- The system can then reserve an appropriate amount of memory and expect a certain type of data to fill that memory.

```
// Declare a integer variable to store total quantity
int TotalQuantity;
```


Instantiating a Variable

- In addition to declaring a variable we must also instantiate the variable.
  - This means that we assign data to the variable.
- As with most programming languages, we use the equal sign to assign a value to a variable.
- On the left of the equal sign will be the variable name and on the right of the equal sign will be the value that is assigned to the variable.

```
// Set the total quantity to 100
TotalQuantity = 100;
```


Primitive Data Types

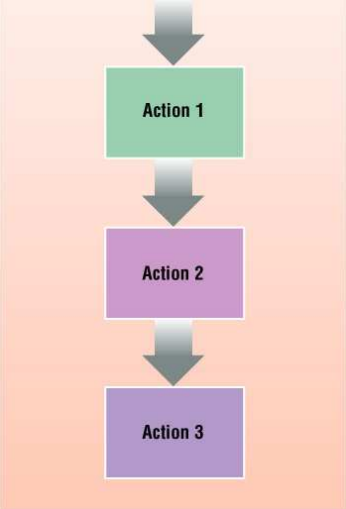
Type	Description	Memory Size
boolean	True or False	Varies 1bit to 4 bytes
char	Unicode characters	2 byte (16 bit)
byte	Integers -128 to 127	1 byte (8 bit)
short	Integers -32,768 to 32,767	2 byte (16 bit)
int	Integers -2,147,483,648 to 2,147,483,647	4 byte (32 bit)
long	Integers -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8 byte (64 bit)
float	Decimals 3.4e-038 to 3.4e+038	4 byte (32 bit)
double	Decimals 1.7e-308 to 1.7e+308	8 byte (64 bit)

Appendix 2: Programming Control Structure

- Three types of control structure, derived from structured programming:
  - Sequences of instructions
  - Selection of alternative instructions
  - Iteration (repetition) of instructions

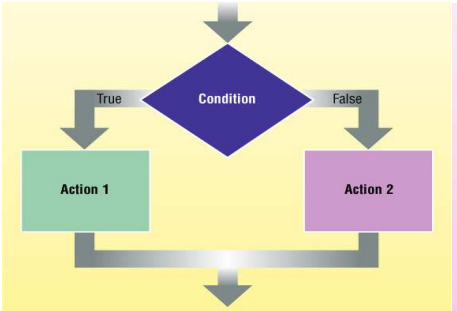
The Sequence Structure

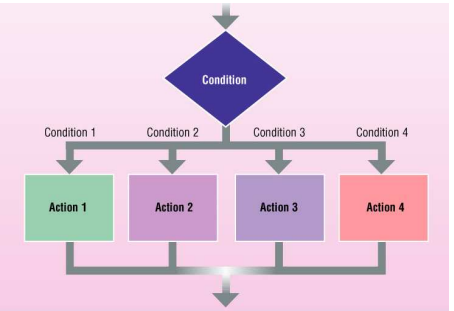
- Directs computer to process program instructions in a particular order
- Set of step-by-step instructions that accomplish a task



The Selection Structure

- Also called the **Decision Structure**, Makes a decision and then takes appropriate action based on that decision.





The if-then-else Statement

- The if-then statement is the most basic of all the control flow statements.
- It tells your program to execute a certain section of code only if a particular test evaluates to true.

```
if ( condition is true ) {
        execute statement(s) if condition is true
} else {
        execute statement(s) if condition is false
}
```

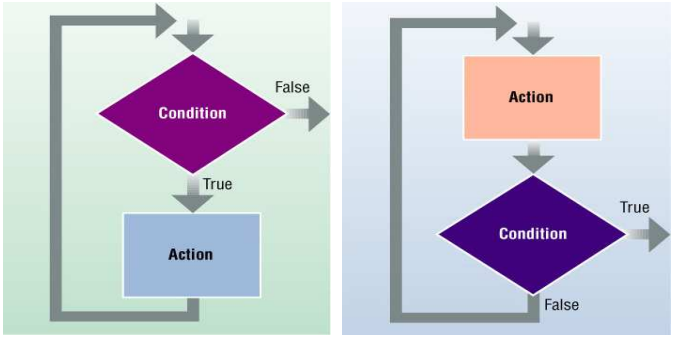

Switch Statement

- A switch statement is a common conditional decision structure.
- It passes in a variable to evaluate and compares it to each case.
- If the case value matches the test variable then the code within the case executes.
- A break will end the case.
- You can code as many cases as needed.

```
switch ( expression ) {
  case 1:
     execute statement(s);
     break;
  case 2:
     execute statement(s);
     break;
  default:
      execute statement(s);
      break;
}
```

The Iteration Structure

- Directs computer to repeat one or more instructions until some condition is met
- Also referred to as a **Loop**, **Repeating** or **Iteration**



The while Loop

- The while loop is one of the most common constructs in programming.
- A while loop is a control structure that allows you to repeat a task a certain number of times.

```
while ( condition ) {
        execute statement(s)
}
```


The do…while Loop

- There is one crucial difference between the Do-While and While loop.
  - In the While loop, the action is performed only if the condition is true.
  - A Do-While loop, on the other hand, will run at least once because the body (Do) is executed before the condition (While) is tested.

```
do {
        execute statement(s)
} while ( condition )
```


The for Loop

- A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
- Here is the flow of control in for loop:
  1. The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
  2. Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.
  3. After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables.
  4. The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself. After the Boolean expression is false, the loop terminates.

```
For (initialization; condition is true; update) {
        execute statement(s)
}
```