

EMBEDDED SYSTEMS PROGRAMMING 2016-17

UI Specification: Approaches

UIS: APPROACHES

- **Programmatic approach:** UI elements are created inside the application code
- **Declarative approach:** UI elements are listed in a data structure that is external to the code, albeit linked to it in some way.
The data structure can be usually accessed with a visual editor
- A mixed approach is possible

PROGRAMMATIC APPROACH: PROS AND CONS

- ⊕ Flexibility
- ⊕ UI can be built at run time
- ⊖ Not clear where/what to change to modify the UI
- ⊖ Modifications imply recompilation
- ⊖ Difficult to support multiple languages and/or multiple screen sizes

DECLARATIVE APPROACH: PROS AND CONS

- ⊕ Better design: the presentation of the application is well separated from the code that controls its behavior
- ⊕ Modifications concentrated in one point.
And no need to recompile!
- ⊕ Easy to support multiple languages and/or multiple screen sizes

Bottom line: go declarative

DECLARATIVE APPROACH: ANDROID

- UI data stored in **XML** files
- The **XML** vocabulary corresponds to the names of the `View` class/methods and its subclasses/methods

1. Write the **XML** code

2. The **XML** is compiled into a **resource**

3. Load the resource from your **Java** code

DECLARATIVE HELLOWITHBUTTON (1/4)

- Project file `app/src/main/res/layout/activity_hello_with_button.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:orientation="horizontal"
    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="it.unipd.dei.esp1516.hellowithbuttond.HelloWithButton">

    <Button android:id="@+id/bu"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pressme" />
    <TextView android:id="@+id/tv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pressplease" />

</LinearLayout>
```

DECLARATIVE HELLOWITHBUTTON (2/4)

- Project file

app/src/main/res/values/strings.xml

```
<resources>  
  <string name="app_name">HelloWithButtonD</string>  
  <string name="pressme">Press me</string>  
  <string name="pressplease">Press the button, please</string>  
  <string name="goodjob">Good job!</string>  
</resources>
```

DECLARATIVE HELLOWITHBUTTON (3/4)

- **Source file** `app/src/main/java/it/unipd/dei/esp1516/hellowithbuttond/HelloWithButton.java (1/2)`

```
package it.unipd.dei.esp1516.hellowithbuttond;

import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class HelloWithButton extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        // Display the layout
        setContentView(R.layout.activity_hello_with_button);
    }
    ...
}
```

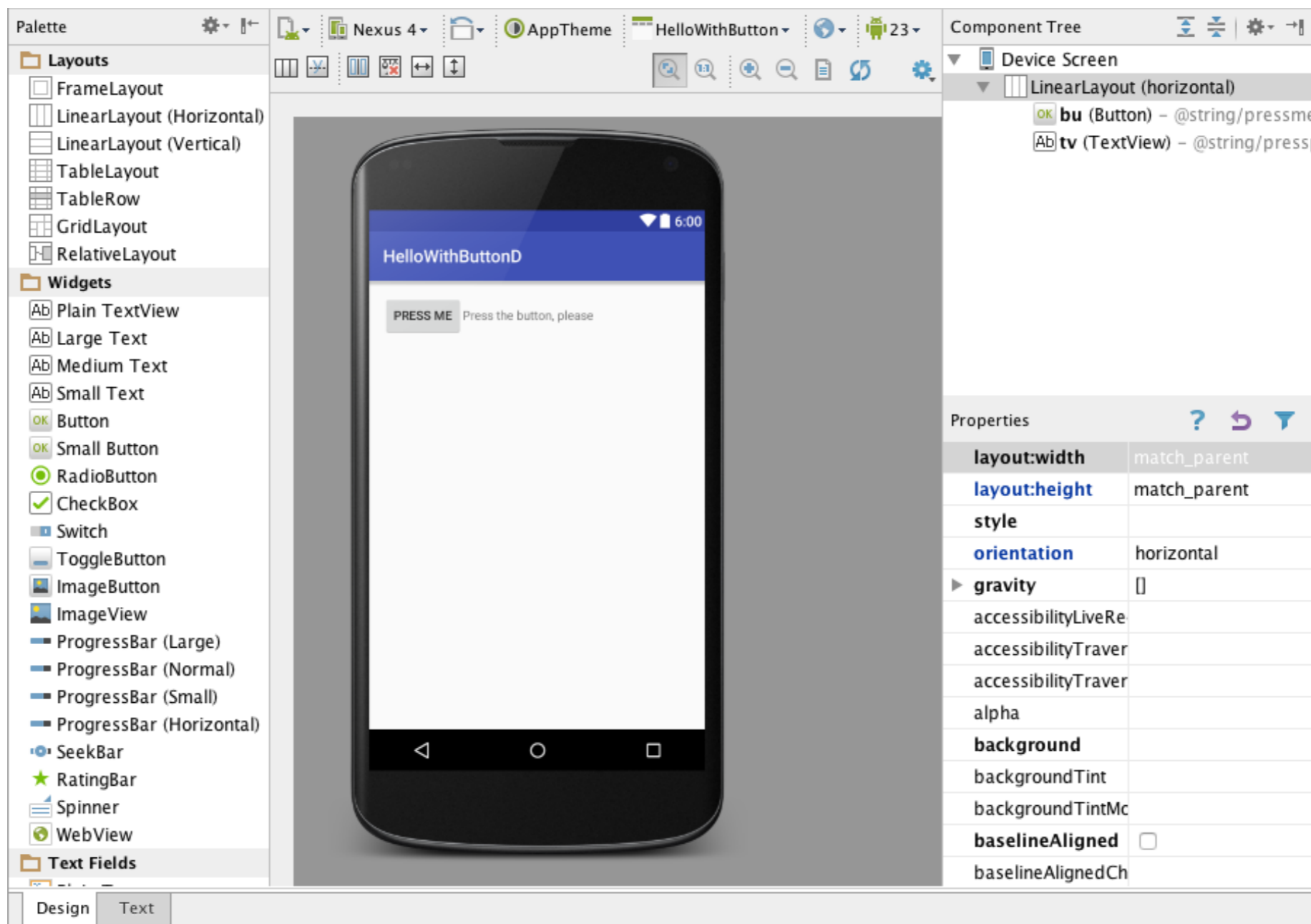

DECLARATIVE HELLOWITHBUTTON (4/4)

- **Source file** `app/src/main/java/it/unipd/dei/esp1516/hellowithbuttond/HelloWithButton.java (2/2)`

```
...  
    // Get references to the TextView and the button.  
    // Do it AFTER setContentView()! Before setContentView()  
    // the objects have not been instantiated yet  
    final TextView tv = (TextView)findViewById(R.id.tv);  
    Button bu = (Button)findViewById(R.id.bu);  
  
    // Set the action to be performed when the button is pressed  
    bu.setOnClickListener  
    (  
        new View.OnClickListener() {  
            public void onClick(View v) {  
                // Perform action on click  
                tv.setText(getString(R.string.goodjob));  
            }  
        }  
    );  
}
```

ANDROID: IDE SUPPORT (1/3)

- Visual editing of the XML layout file




ANDROID: IDE SUPPORT (2/3)

- Visual editing of `strings.xml`

The screenshot shows the visual editor for `strings.xml` in Android Studio. At the top, there is a toolbar with a green plus icon, a globe icon, and a checkbox labeled "Show only keys needing translations". To the right of the toolbar is a link that says "Order a translation...". Below the toolbar is a table with the following columns: "Key", "Default Value", and "Untransl...". The table contains the following rows:

Key	Default Value	Untransl...
app_name	HelloWithButtonD	<input type="checkbox"/>
goodjob	Good job!	<input type="checkbox"/>
pressme	Press me	<input type="checkbox"/>
pressplease	Press the button, please	<input type="checkbox"/>

Below the table, there are three input fields for editing the selected key:

- Key:
- Default Value: 
- Translation: 

ANDROID: IDE SUPPORT (3/3)

- Fragment of autogenerated R.java source file

```
package it.unipd.dei.esp1516.hellowithbuttond;

public final class R {
    public static final class id {
        ...
        public static final int bu=0x7f0c0050;
        public static final int tv=0x7f0c0051;
    }
    public static final class layout {
        ...
        public static final int activity_hello_with_button=0x7f040019;
    }
    public static final class string {
        ...
        public static final int app_name=0x7f060014;
        public static final int goodjob=0x7f060015;
        public static final int pressme=0x7f060016;
        public static final int pressplease=0x7f060017;
    }
    ...
}
```

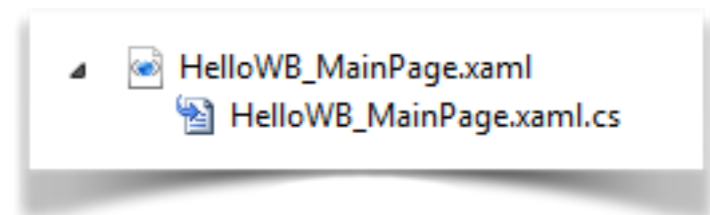
DECLARATIVE APPROACH: IOS

- UI data stored in **XIB** files or **Storyboard** files.
- Such files contain **XML** code but are not easy to read
- Objective-C code can refer to programmatically-defined UI objects via special instance variables called **outlets**.
UI objects can invoke Objective-C code via special methods called **actions**

1. Create the UI with Interface Builder (inside Xcode)
2. Create outlets and actions in the source code
3. Establish the connections with Interface Builder

DECLARATIVE APPROACH: WINDOWS PHONE (1/2)

- Let us consider XAML applications
- Each page is associated with
 - one **.xaml file**
that specifies the visual appearance of the page,
 - one **code-behind file** (written in C# or VB)
that specifies the control logic of the page



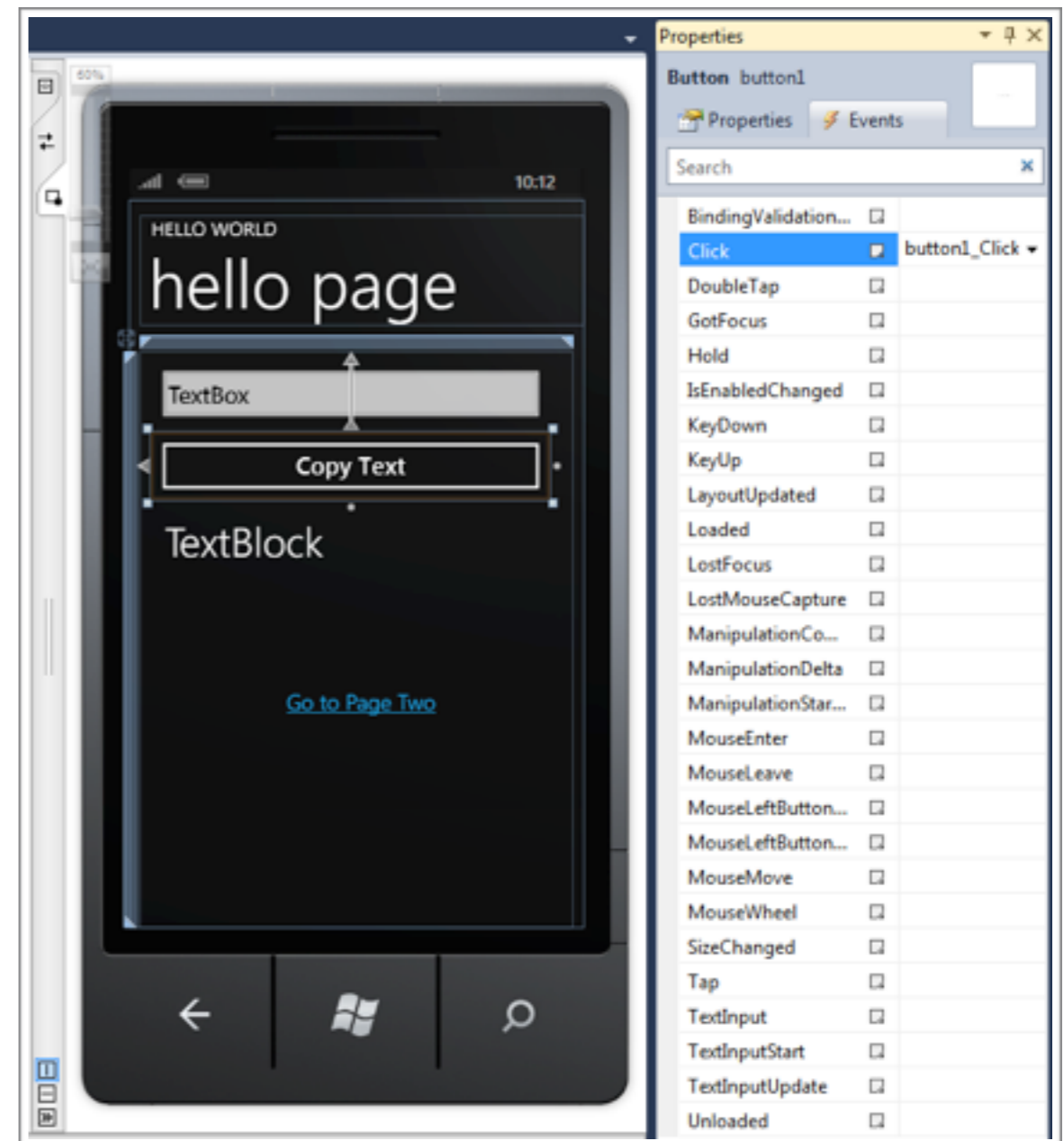
DECLARATIVE APPROACH: WINDOWS PHONE (2/2)

- Visual Studio allows to specify connections between UI events and methods in the code-behind file

```
namespace HelloWithButton
{
    public partial class MainPage : PhoneApplicationPage
    {
        ...

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            textBlock1.Text = "Good job!";
        }

        ...
    }
}
```



LAST MODIFIED: MARCH 20, 2017

COPYRIGHT HOLDER: CARLO FANTOZZI (FANTOZZI@DEI.UNIPD.IT)
LICENSE: CREATIVE COMMONS ATTRIBUTION SHARE-Alike 3.0