# 0 – Introduction

Hey guys and welcome to the tutorial. My name is Umer and my group mates are Elena, Shukry, and Poulad. In this video tutorial we are going to talk about a simple android application we wrote.

We will be using Xamarin framework that enables us to create Android apps using C#.

If you are experiencing problems with installing or running Xamarin, consider watching the tutorial videos that Poulad made or search for the error online. The tutorials are sent to the group members through email and also are uploaded to the shared folders.

We have sent you the project’s zip file with this video. During this tutorial we go over the code we wrote and try to explain it. Our application is a simple Rock Paper Scissors game. Extract the project’s zip file and open the solution.

We have organized each part into a region. We are going to open them in order and explain those parts.

Here you see an overview of the topics we are going to cover. First, we quickly create a simple layout and then implements the app’s functionality.

After that will come the parts for activity life cycle and intents. Next thing will be enhancing the layout by using two more attributes.

And finally, the assignments. We hope you will follow this tutorial and will do the assignments. The solution to the assignments will be posted on the group next Tuesday so if you managed to do any of those assignments, please send your code as an email to the group.

Alright, let’s get started. The first part would be presented by Elena.

# 1 – Layout 1

## Presentation – Part 1

Hey Everyone. I am going to talk about Layouts in an Android application.

Layouts control the visual structure of elements on the screen. You can see some examples of common layouts here.

In our game, we are going to use some linear layouts together. Linear layouts organize its children in a horizontal or vertical row.

This is how our design is going to look like by the end of this tutorial. We have a bunch of Images and Buttons on the screen.

These 3 Image Buttons are put together using a Linear Layout with Vertical orientation.

These layouts are also nested with other layouts shown by the red boxes and then finally everything is nested within a vertical layout shown by the yellow box.

Blue Boxes here represent Horizontal Linear Layouts

There are other Horizontal linear layouts containing elements. Eventually everything is in a Vertical linear layout.

Now, I open visual studio to begin creating the layout

## Design Mode – Part 1

In the Resources folder go into the Layout folder and open the Main Layout.

We already have the layout finished but for the tutorial we will do it again.

Just like HTML, the visual structure of a layout in Android, is controlled by an XML-based language. Let’s take a look at the source.

I remove lines 7 to 108.

Now we are left with a single Linear Layout with vertical orientation.

I’ll pin this toolbox here in design mode for easy access.

In the start we only have the outer Vertical layout which was the yellow box.

Now, we will place 3 horizontal layouts within the outer vertical layout.

We will also place two more horizontal layouts in the first horizontal layout from the top.

Make sure to drop them carefully within the box.

We can also drag and drop view elements the same way as layouts.

Filled box designates that it will be place within the container.

In the middle horizontal layout we have a nested vertical layout with 3 ImageButtons.

The bottom Horizaontal Layout contains 2 Buttons.

## Presentation – Part 2

Each view element has a type such as Button or TextView. They also have a unique ID. We assign them IDs so we can work with them; later in the code.

We also will be adding the images wherever is needed.

We have added some images to the Drawable folder.

## Design Mode – Part 2

I go to ImageViews first. They have an ID and a source attribute.

Assign the ID to iv (for the type ImageView) and then Player.

And set the image using the src attribute.

I do the same thing for the rest of the View elements on the layout.

Now we have a layout to show to the user so Shukry is going to go over the app’s logic.

# 2 – Play

In this part, I am going to explain the logic behind the game. First, let’s have a look at the game in the emulator.

When the user clicks on one of the image buttons, the game is started and CPU makes its own choice.

The image for CPU’s choice is changed and then the winner’s score is incremented.

If the result was draw, a Toast message is shown

## Class fields

I expand region 0: Class fields’ declaration.

Here, the variables I need to work with are declared as class fields. They are in class scope so will be accessible in every method of this activity.

Before going over the next region, let’s have a quick look at the slides.

The xml code on the right side is just an example of a layout that contains a button with id bReset.

What we have on the left side is the code for that activity. We can use method FindViewById to create an object from an element on the layout.

We need to provide the type of that element and its unique id as an argument.

Xamarin framework takes care of constructing this Resource.Id class every time we edit and save our layout.

Now the code in this part should make sense. You also may find this method similar to the FindElementById function in JavaScript.

## Event Handlers

Having all the variables initialized, I go to region Play (event handlers). I simply have chained the Play method to the Click event handlers of those 3 image buttons.

## Play Method

Now, I go to the region for Play method.

Since the Play method is supposed to handle the click event, it must accept two parameters. First one is of type object which represents the sender or the image button that user clicked on. The other parameter is for the Event Arguments.

In the method, we use that Random number generator to have a random choice for CPU. Then we check for the winner inside the switch statement.

For example if CPU’s choice was rock, we show it to the user by changing the CPU’s choice image to rock.

After that, we simply check which image button user clicked on.

In the case of a draw, we show a Toast message to the user.

If we had a winner, we increment the winner’s score by one and change the TextView in the layout.

The logic for cases 1 and 2, Paper and Scissors respectively, are similar to the case 0.

## Lambda Expressions

Now, go ahead and open the region for Lambda expressions.

Lambda expressions are just a shorthand way for writing anonymous methods.

Whenever the Button reset is clicked, all the score counters and Text Views are reset to 0.

As I mentioned before, every method that handles that Click event, must receive two parameters. I put those two inside round brackets.

The Close button has only one task: Calling the method Finish to close the app.

Since the compiler expects two parameters of type object and EventArgs, I can just write their names.

Now, this is the end of my part and Umer is going to continue by talking about the life cycle of an activity.

# 3 - Activity Life Cycle

## Slide (Part 1)

In this section, I’ll cover the activity life cycle in Android. Android OS calls some specific methods from an activity when the activity goes in different states. We can put some code in those methods to be processed. The methods are called Life Cycle Methods.

When the user clicks on an application, the app starts from the first activity (Launcher Activity). Android OS calls the activity’s OnCreate method. OnCreate is the only method that is mandatory for every single activity. We can use it to initialize variables and set display layout.

Then come the OnStart and OnResume methods. When the OnResume is called, activity is visible and has the focus and user can interact with the app.

OnPause method is executed when the activity is partially visible. For example a dialog box for entering a date can cover a part of the activity.

If the user navigates away from our app to another app and our activity becomes completely covered, the OnPause and then OnStop methods will be called. In this state, the app is still alive but it is not visible.

After that, when user opens up our app again, the methods OnRestart, OnStart, and OnResume are called. We can write some code in OnRestart method to allocate resources again.

And lastly, we have OnDestroy. The app is shut down when user hits back button from the launcher activity, Android OS needs resources such as Memory, or the device is rotated.

## Code (Part 1)

Back to the code, expand the region 3 (Activity Life Cycle).

I have overwritten 3 methods: OnStart, OnStop, and OnDestroy. This will help to understand the life cycle of an activity.

Just like any other life cycle method, I start by calling the base class’s method and then simply showing a toast message that says the name of the method.

Toast messages are going to be shown whenever these methods are called so let’s launch the app to see the life cycle methods in action.

The OnStart() was called at the start of the activity as expected. I press the Home button and OnStop() is going to be called.

I encourage you to override other life cycle methods such as OnPause(), this will help you better understand the order of these method calls.

Now, the app is in running state and I am going to rotate the screen by pressing the keys Ctrl + F11.

As toast messages told us, OnStop, OnDestroy and again OnStart() were called one after the other so the activity was initialized again. This means the layout was destroyed and recreated. As a result, the information we had such as scores is now lost.

## Slide (Part 2)

There are two other life cycle methods that can address this issue. We use those methods to maintain the state of the application during the life cycle of the activity.

With the use of the method OnSaveInstanceState we can save some data we want to maintain. This method is called just before OnPause.

On the other side, we get all the data we saved in the method OnRestoreInstanceState.

## Code (Part 2)

Let’s expand the region 4 - Maintaining State.

I have overwritten the methods OnSaveInstanceState and OnRestoreInstanceState.

Both of the methods receive a Bundle object as their parameter. This Bundle is just a key-value data structure.

First, for saving the state, I just need to keep track of the scores. Scores are simply integers so I used the method PutInt.

I store the integer value of playerScore for the key player\_score. And the same thing with CPU’s score.

Notice here I’ve called the base class’s method after manipulating the bundle. So this method basically receives the Bundle, adds two other things to it and passes it to the base class.

That Bundle object from saved instance would be passed to the OnRestoreInstanceState at some other point.

This time I just get those integers back using the same keys I used before in saving the state.

In addition, the TextViews that show the scores need to be updated as well.

Let’s check our code using the emulator.

Everything seems ok and this wraps up my part about activity life cycle. Poulad is going to present the next part.

# 4 – Implicit Intent

## Slide

In this section, I will be talking about the implicit intents.

In our application, when user clicks on the Player’s image, he can take a picture using the camera application. The picture will be set as the player’s image.

Since I am running the application on an Android Virtual Device, I see an emulated camera but it’ll be just like a normal camera on any actual device.

We can achieve that feature in an Android app using Intents. We use Intents to open components.

I used an Implicit Intent in this case. Implicit Intents request an action to be performed. The request (Capturing an image in this case) is sent to the Android OS.

The OS redirects that request to an installed application, the Camera app.

If user captures the image, an intent containing image’s thumbnail would be passed back to Android OS and then ­­to our activity.

Android OS handles these all for us and we only need to use two methods in our activity.

For sending the request, We need to create an Intent object and pass it to the method StartActivityForResult().

Eventually the activity from Camera application would be started and the result of capturing the image would be sent back to our activity.

Method OnActivityResult() is where we receive the result. Let’s take a look at the code.

## Code

I expand the region 5 (sending a request). I have hooked up a lambda expression to the click event handler of the player’s image.

When user clicks that Image View, An Intent is created. Notice that the argument I have passed to its constructor is a simple constant string and here you can see its value. This string defines the action.

Then I call the method StartActivityForResult() and pass it that Intent object. I’ll talk about this 0 here in a few seconds.

In the other region for Implicit Intents, I have overwritten OnActivityResultMethod().

This method is called whenever the Android OS passes back the result of an action we requested to be performed.

The requestCode parameter in this method corresponds to this 0 here. Having a requestCode helps us to keep track of our intents.

Android applications are more complex than this example. The result of all the implicit intents we send come here in different states, different times so we say requestCode number 0 is for the Player’s image.

In a simple if statement I check if the requestCode was 0 and the result of action was Ok, I use this statement to extract a thumbnail of the captured image as a bitmap.

Then I set the player’s image to that bitmap.

In the next part of this tutorial we are going to enhance the layout.

# 5 – Layout 2

## Slide

The app’s implementation is done. In this part, I’ll try to enhance the design.

The first thing I am going to talk about is Weight.

The weight of a view element specifies its importance in the terms of how much space it should take.

Look for the weight attribute for these buttons.

The attribute weightSum is for the parent. In this case, the linear layout.

Each child element has its layout\_weight and the sum of them should be equal to the weightSum of the parent.

The other attribute is gravity. Here is a Vertical Linear Layout that contains a Button.

When we set the gravity to center\_vertical, the content inside the layout is centered vertically.

The layout\_gravity changes the gravity of the element inside its parent.

As you can see, here we have a Vertical Linear Layout with a Button inside.

By setting the layout\_gravity to right, the Button tries to put itself on the right side of its parent.

I encourage you to change the gravity and weight of the elements in the game’s layout to understand their behavior better.

## Design Mode

Click on this button to be able to select the containers or layouts.

Select the root layout that contains everything else and change the weightSum to 100.

Now set the layout\_weight for its children.

I give 20% to the layout on top, 20% to the bottom layout and 60% to the layout in the middle.

There are 2 layouts in the middle. I change their gravity attribute so the elements inside them will be centered.

Here I set the weight of 1 for both layouts inside the top layout.

Notice that I didn’t specify a weightSum of 2 for their parent. Fortunately the Android OS is smart enough to understand the sum should be 2.

The TextViews look better with their contents centered.

Lastly, I don’t like to see the buttons stretched; so I set their height to wrap\_content.

And this concludes my part. Poulad will be presenting next.

# 6 – Assignments

Alright, here come your assignments. Now, it’s your turn to work on this project.

Write 2 lambdas for reset and close buttons to do the specified tasks.

When the user rotates the screen, the CPU’s choice is lost. Keep track of that using the life cycle methods.

We decided to design a simple layout that gives us a better control on the attributes we wanted to introduce.

You can change the layout or redesign it based on your own ideas.

For example, I have added another TextView here that shows the result of each round of the game.

Keep in mind that you also have to update the text based the events that happen.

I will send you the way I solved these a few days later.

There are a lot more about Android OS to cover but we hope we have helped you to understand some of the fundamentals.

Alright. This is the end of this tutorial. Thank you for watching and happy coding.