**Transcript:**

***<Grant Saylor>***

**0 – 3.5**

***SLIDE 1 – The Art of the Architecture:***

Hi everyone, this is Grant speaking. Today myself, Kyle, Anthony and Jiayi will talk about the architecture of our Android application, primarily asynchronous tasks and how we connect it with other services to create a fully featured app. First up is Kyle to talk about the concept of asynchronous tasks.

***Slide 6***

Hi, this is Grant again. I’d like to preview how we use an event listener in our app, Virtual Library. This behaves like an asynchronous task. When our map is being set up, we call the Google Maps API to fill a map fragment (like a map class) with the Google Map view. If we left it like this, the Google Map would just show streets and nothing else, the user wouldn’t be able to manipulate the map or anything at all.

In our code, once a callback to Google Play services is fulfilled, the “OnMapReady” function is triggered, this will allow the map to be manipulated and filled out with markers. You might be more familiar with markers being referred to as “pins”, these are the visual representation of libraries on our map, the colorful book icon. Inside of the OnMapReady method is the event listener we’ve been referring to.

***Slide 7***

Using the event listener we can do various things like allowing users to add additional markers to the map and regenerate the map with existing libraries. Methods are in place to listen for markers being placed around the map, to push and pull the associated data to our database and to add it all to an array to regenerate it for users at a later time.

The event listener is always ready to go and automatically triggers once the map is ready, which to the user should look pretty instant.

To make it a bit clearer, during the loading of the map, all the information to be put onto the map is also being loaded, but it is just waiting to be placed onto the map until the map is ready.

Testing our map without the event listener, or if you placed the event listener in a place where it could trigger before the map is ready would cause a crash. The data is there, but there is nothing to put the data into yet, which is why we wait for the map to finish generating!

Further in the event listener there are calls to a regeneration method that in turn calls the database to retrieve latitude and longitude values amongst others to place the markers onto the map as you see in the screenshots. Every user will see the same markers on their map and if a user places a marker onto the map, the event listener will listen and populate that for every other user as well.

***Slide 8***

There are tons of uses for event listener and async tasks in an android app. In our own app we use it to talk with our Firebase database, whether that is the map screen as you’re all now familiar with or with other screens in our app, such as searching for a book inside our database. All of this is data that we can’t necessarily sequentially program because we don’t have hardcoded values for them, so we do it asynchronously in conjunction with Firebase to prevent a crash scenario. While all of this may seem a bit nested in our code, to the user it all appears seamless and they would never know the intricate dance going on behind the scenes.

I hope this was a useful insight in how asynchronous tasks can make or break your Android application.

I’ll be handing it over to Jiayi so she can close us out with an explanation of Anko and how we use it in our app.

***Slide 6/7/8 Full Script:***

This is Grant again. I’d like to preview how we use an event listener in our app, Virtual Library. This behaves like an asynchronous task. When our map is being set up, we call the Google Maps API to fill a map fragment (like a map class) with the Google Map view. If we left it like this, the Google Map would just show streets and nothing else, the user wouldn’t be able to manipulate the map or anything at all.

In our code, once a callback to Google Play services is fulfilled, the “OnMapReady” function is triggered, this will allow the map to be manipulated and filled out with markers. You might be more familiar with markers being referred to as “pins”, these are the visual representation of libraries on our map, the colorful book icon. Inside of the OnMapReady method is the event listener we’ve been referring to.

Using the event listener we can do various things like allowing users to add additional markers to the map and regenerate the map with existing libraries. Methods are in place to listen for markers being placed around the map, to push and pull the associated data to our database and to add it all to an array to regenerate it for users at a later time.

The event listener is always ready to go and automatically triggers once the map is ready, which to the user should look pretty instant.

To make it a bit clearer, during the loading of the map, all the information to be put onto the map is also being loaded, but it is just waiting to be placed onto the map until the map is ready.

Testing our map without the event listener, or if you placed the event listener in a place where it could trigger before the map is ready would cause a crash. The data is there, but there is nothing to put the data into yet, which is why we wait for the map to finish generating!

Further in the event listener there are calls to a regeneration method that in turn calls the database to retrieve latitude and longitude values amongst others to place the markers onto the map as you see in the screenshots. Every user will see the same markers on their map and if a user places a marker onto the map, the event listener will listen and populate that for every other user as well.

There are tons of uses for event listener and async tasks in an android app. In our own app we use it to talk with our Firebase database, whether that is the map screen as you’re all now familiar with or with other screens in our app, such as searching for a book inside our database. All of this is data that we can’t necessarily sequentially program because we don’t have hardcoded values for them, so we do it asynchronously in conjunction with Firebase to prevent a crash scenario. While all of this may seem a bit nested in our code, to the user it all appears seamless and they would never know the intricate dance going on behind the scenes.

I hope this was a useful insight in how asynchronous tasks can make or break your Android application.

I’ll be handing it over to Jiayi so she can close us out with an explanation of Anko and how we use it in our app.