Practice final Exam

1. **How to work with WebView UI and navigating webpage history.**

Answer:

val url = *intent*.getStringExtra("url") ?: ""  
binding.webview.*webViewClient* = WebViewClient()  
binding.webview.*settings*.*javaScriptEnabled* = true  
binding.webview.*settings*.*builtInZoomControls* = true  
binding.webview.loadUrl(url)

binding.btnFwd.setOnClickListener **{** binding.webview.goForward()  
**}**binding.btnBack.setOnClickListener **{** binding.webview.goBack()  
**}**

binding.edtSearch.setOnQueryTextListener(object : androidx.appcompat.widget.SearchView.OnQueryTextListener {  
 override fun onQueryTextSubmit(query: String?): Boolean {  
 val url = query?.let {  
 //if (Patterns.WEB\_URL.matcher(url).matches()) {  
 binding.webview.loadUrl("http://$it")  
 //}  
 } ?: run {  
 binding.webview.loadUrl("http://www.google.com")  
 }  
 return true  
 }  
  
 override fun onQueryTextChange(newText: String): Boolean {  
 return true  
 }  
})

1. **Sensor Programs**

**AccelerometerDemo:**

private lateinit var sensor: Sensor

//help us manage sensor components

private lateinit var sm: SensorManager

sm = getSystemService(SENSOR\_SERVICE) as SensorManager

//select the sensor we wish to use

sensor = sm.getDefaultSensor(Sensor.TYPE\_ACCELEROMETER)

// Set the data source for the media player object

mPlayer = MediaPlayer.create(this, R.raw.iphone)

override fun onResume() {

super.onResume()

sm.registerListener(this, sensor, SensorManager.SENSOR\_DELAY\_NORMAL)

}

// Unregister your Sensor Manager

override fun onPause() {

super.onPause()

sm.unregisterListener(this)

}

override fun onSensorChanged(event: SensorEvent?) {

if (event != null) {

binding.displayReading.text=

"X=${event.values[0]}\nY=${event.values[1]}\nZ=${event.values[2]}"

}

if (event!!.values[0] > 10) {

mPlayer.start()

}

// To release the media player once player complete the audio

mPlayer.setOnCompletionListener(MediaPlayer.OnCompletionListener { mPlayer.release() })

}

override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {

}

**LightMusicDemo:**

// Declare Sensor object to select particular sensor

private var sensor: Sensor? = null

// To manage Sensor components declare SensorManager Object

private var sm: SensorManager? = null

// Declare MediaPlayer object to play music

private lateinit var mp: MediaPlayer

var cuttentLux = 0

private lateinit var binding: ActivityMainBinding

// Initialize object for Sensor service using getSystemService()

sm = getSystemService(Context.SENSOR\_SERVICE) as SensorManager?

// Initialize Sensor object for the Type light using getDefaultSensor() from SensorManager object

sensor = sm!!.getDefaultSensor(Sensor.TYPE\_LIGHT)

// Check device has the requested Sensor or not

if(sensor==null){

Toast.makeText(this, "Your device has no Sensor.TYPE\_LIGHT", Toast.LENGTH\_LONG).show()

}

// Register your Sensor Manager

override fun onResume() {

super.onResume()

sm.also { sm!!.registerListener(this, sensor, SensorManager.SENSOR\_DELAY\_NORMAL)}

// sm!!.registerListener(this, sensor, SensorManager.SENSOR\_DELAY\_NORMAL)

}

// Unregister your Sensor Manager

override fun onPause() {

super.onPause()

sm!!.unregisterListener(this)

}

override fun onSensorChanged(event: SensorEvent?) {

mp = MediaPlayer.create(applicationContext, R.raw.iphone)

if (event!!.values!![0] < 20 )

{ // < 20 use for dark

try

{

// mp.setDataSource("http://www.tamilmp3plus.com/837/vijay-prakash-hits/");

// mp.prepare();

// mp = MediaPlayer.create(applicationContext, R.raw.iphone)

mp.setOnCompletionListener(MediaPlayer.OnCompletionListener { mp.release() })

mp.start()

}

catch (e: Exception) {

e.printStackTrace()

}

}

}

override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {

}

**LightProgressDemo:**

private var sensorManager: SensorManager? = null

private var lightSensor: Sensor?=null

sensorManager = getSystemService(Context.SENSOR\_SERVICE) as SensorManager?

lightSensor = sensorManager!!.getDefaultSensor(Sensor.TYPE\_LIGHT)

if(lightSensor==null){

Toast.makeText(this,"Your device has no Sensor.TYPE\_LIGHT", Toast.LENGTH\_LONG).show()

}

// Register your Sensor Manager

override fun onResume() {

super.onResume()

sensorManager!!.registerListener(this, lightSensor, SensorManager.SENSOR\_DELAY\_NORMAL)

}

// Unregister your Sensor Manager

override fun onPause() {

super.onPause()

sensorManager!!.unregisterListener(this)

}

// get sensor update and reading

override fun onSensorChanged(event: SensorEvent) {

// val ac = event.accuracy

// Toast.makeText(this,event.toString()+" Accuracy : "+ac,Toast.LENGTH\_LONG).show();

if (event.sensor.type == Sensor.TYPE\_LIGHT) {

val currentReading = event.values[0] // Return value is float type

// lightmeter is the id of ProgressBar from the layout

binding.lightmeter.progress = currentReading.toInt() // Convert float to int

binding.reading.text = ("Current Reading: ${(currentReading)} Lux")

}

}

override fun onAccuracyChanged(sensor:Sensor, i:Int) {

}

1. **Able to create Room DB Entity and DAO operations.**

@Entity  
data class MovieItem(// @ColumnInfo(name = "notetitle")  
 val title :String,  
 val note :String  
): Serializable // need to pass Note entity through Fragments. So that entity should be Serializable  
{  
 @PrimaryKey(autoGenerate = true)  
 var id:Int = 0  
}

@Dao  
interface MovieDao {  
 @Insert  
 suspend fun addNote(note:MovieItem)  
 @Query("SELECT \* FROM MOVIEITEM ORDER BY id DESC")  
 suspend fun getAllNotes():List<MovieItem>  
 @Insert  
 suspend fun addMultipleNotes(vararg note: MovieItem)  
 @Update  
 suspend fun updateNote(enote:MovieItem)  
 @Delete  
 suspend fun deleteNote(note: MovieItem)  
}

1. **How to play the Video using VideoView and Media Controller.**

binding.videoView1.setVideoPath("https://www.demonuts.com/Demonuts/smallvideo.mp4")

// To read from raw folder

// binding.videoView1.setVideoPath("android.resource://" + packageName + "/" + R.raw.samplevideo )

/\*VideoView canvas will cause the media controls will appear over the video playback by tapping.

These controls should include a seekbar together with fast forward, rewind and play/pause buttons.

\* \*/

mediaController = MediaController(this)

/\* Set the view that acts as the anchor for the control view.

\* This can for example be a VideoView, or your Activity's main view.

\* When VideoView calls this method, it will use the VideoView's parent(SurfaceView)

\* as the anchor. Parameter view The view to which to anchor the controller when it is visible\*/

//mediaController?.setAnchorView(binding.videoView1)

binding.videoView1.setMediaController(mediaController)

// configure video playback to loop continuously and display the video duration on logs.

binding.videoView1.setOnPreparedListener { mp ->

mp.isLooping = true

Log.i(TAG, "Duration = " + binding.videoView1.duration)

}

// Start Playing

binding.videoView1.start()

1. **Retrofit API Interface, Retrofit Builder and call back using Kotlin Coroutines.**

val api = Retrofit.Builder() // build your Retrofit Object

.baseUrl(base\_url) // hit the base\_url

.addConverterFactory(GsonConverterFactory.create()) // Perform serialization and deserialization

.build() // // Create an instance using the configured values

.create(APIInterface::class.java)

// GlobalScope.launch(Dispatchers.Main) { //This also work

lifecycleScope.launch {

val response = api.getImages()

if( response!!.isSuccessful){ // Check using non null !! operator

response.body().let {

if (it != null) {

var items = it

myAdapter.list = items

myAdapter.notifyDataSetChanged()

}

}

}

else

{

Toast.makeText(applicationContext,"Failed to Retrieve ", Toast.LENGTH\_LONG).show()

}

}

**Jetpack Components Categories**

|  |  |  |  |
| --- | --- | --- | --- |
| **Foundation** | **Architecture** | **Behaviour** | **UI** |
| * **App Compat** | * **Data Binding** | * **Download Manager** | * **Animations & Transitions** |
| * **Android KTX** | * **Life Cycles** | * **Media & Playback** | * **Auto** |
| * **Multidex** | * **Live Data** | * **Notifications** | * **Emoji** |
| * **Test** | * **Navigation** | * **Permissions** | * **Fragments** |
|  | * **Paging** | * **Preferences** | * **Layout** |
|  | * **Room** | * **Sharing** | * **TV** |
|  | * **View Model** | * **Slices** | * **Wear OS by Google** |
|  | * **Work Manager** |  |  |

1. **Various kinds of Android storage types.** 
   * **DataStore Preferences:  Store private, primitive data in key-value pairs.**
   * **Internal Storage: Store data on the device memory. Data will not be accessed by other application.**
   * **External Storage: Store files on the shared external file system. This is usually for shared user files, such as photos. (SD Card)**
   * **Database: Store structured data in a private database using Room.**
   * **Network Connection: Store data on the Web with your own network server.**

**Refer about storage**

1. **Third party JSON Parsing libraries.**

* GSON 🡪 java library
* POJO 🡪 Plain Old JAVA Object
* POKO 🡪 Plain Old kotlin Object

1. implementation 'com.google.code.gson:gson:2.10.1'

* – **SOAP** stands for Simple Object Access Protocol.
* – **REST** stands for REpresentational State Transfer

val api = Retrofit.Builder()  
 .baseUrl("http://www.google.com")  
 .addConverterFactory(GsonConverterFactory.create())  
 .build()  
 .create(ApiInterface::class.*java*)  
  
*lifecycleScope*.*launch* **{** var response = api.getImages()  
 if( response!!.*isSuccessful*){ // Check using non null !! operator  
 response.body().*let* **{** if (**it** != null) {  
 var items = **it** *println*(items.*forEach***{** *println*(**it**) **}**)  
  
 }  
 **}** }  
 else  
 {  
 Toast.makeText(*applicationContext*,"Failed to Retrieve ", Toast.*LENGTH\_LONG*).show()  
 }  
**}**

* Audio playback can be performed using either the **MediaPlayer** or the **AudioTrack** classes.

To Check if it has camera

val takePictureIntent = Intent(MediaStore.ACTION\_IMAGE\_CAPTURE)

if (takePictureIntent.resolveActivity(packageManager) != null) {

// Call launch by passing

startforResultCamera.launch(takePictureIntent) }

getVideo.launch(Intent(MediaStore.ACTION\_VIDEO\_CAPTURE)

* **MediaRecorder** is used to record audio and video.
* Audio playback can be performed using either the **MediaPlayer** or the **AudioTrack** classes.
* The Navigation component consists of three key parts that are described below:

1. Navigation Graph (New XML resource) - This is a resource that contains all navigation-related information in one centralized location. This includes all the places in your app, known as destinations. A destination is any place you can navigate to in your app, usually a fragment or an activity.
2. NavHostFragment (Layout XML view) - This is a special widget you add to your layout. It displays different destinations from your Navigation Graph.
3. NavController (Kotlin/Java object) - This is an object that keeps track of the current position within the navigation graph. It composes swapping destination content in the NavHostFragment as you move through a navigation graph.

Sending data using nav graph

val directions =HomeFragmentDirections.actionHomeFragmentToSplashFragment(

etName.text.toString(),etAge.text.toString().toInt())

// Calling this on a Fragment to navigate your Directions

findNavController().navigate(directions)

private val nargs : SplashFragmentArgs by navArgs()

binding.tvName.text = "Name is ${nargs.pname}“

binding.tvAge.text = "Age is ${nargs.page.toString()}"

**Add Nav up feature**

class MainActivity : AppCompatActivity() {

// Declare Navigation Controller Object to manage navigation within NavHost

private lateinit var mnavController: NavController

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

// Need this support to interact the fragments associated with the Avtivity

val navHostFragment = supportFragmentManager.findFragmentById(R.id.fragmentContainerView) as NavHostFragment

mnavController = navHostFragment.navController

// Code to link the navigation controller to the app bar

NavigationUI.setupActionBarWithNavController(this,mnavController)

}

// override the onSupportNavigateUp() method to call navigateUp() in the navigation controller

override fun onSupportNavigateUp(): Boolean {

return mnavController.navigateUp()

}

}

There are three major components in Room

1. Database
2. DAO (mapping Queries)
3. Entity ( DB table)

**@Entity**  
data class Note(  
**@PrimaryKey(autoGenerate = true)**val id:Int,  
val title :String,  
val note :String  
)

* **@Dao**
* interface NoteDao {
* **@Insert**
* suspend fun addNote(note:Note)
* **@Query("SELECT \* FROM NOTE ORDER BY id DESC")**
* suspend fun getAllNotes():List<Note>
* **@Update**
* suspend fun updateNote(note:Note)
* **@Delete**
* suspend fun deleteNote(note: Note)
* }
* **@Database(**
* **entities = [Note::class],**
* **version = 1**
* **)**
* // If you have multiple tables mention inside [Note::class, Detail::class ]
* // Must Inherit from RoomDatabase
* **abstract class NoteDatabase():RoomDatabase() {**
* **abstract fun getNoteDao() : NoteDao** // need to get the Dao for the entity
* **}**

**Motion Sensors:** These sensors measure acceleration forces along the three axis. This includes accelerometers, gravity sensors, rotation vector, and gyroscopes

[TYPE\_ACCELEROMETER](https://developer.android.com/reference/android/hardware/Sensor.html) 🡪 Motion detection (shake, tilt, etc.).

**Environmental Sensors**: These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity

[TYPE\_LIGHT](https://developer.android.com/reference/android/hardware/Sensor.html) 🡪 Controlling screen brightness.

**Position Sensors**: These sensors measure the physical position of the device which includes orientation changes and magnetometers

[TYPE\_ORIENTATION](https://developer.android.com/reference/android/hardware/Sensor.html) 🡪 Determining device position.

* **Sensor**
  + You can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.
* **SensorEvent** 
  + This class stores information about the sensor type, sensor data, and values measured by the Sensor etc.,
* **SensorEventListener**
  + you acquired a sensor, you can register a SensorEventListener object on it. This listener will get informed, if the sensor data changes.
  + To avoid the unnecessary usage of battery register your listener in the onResume() method and unregister it in the onPause() method.



lateinit var binding: ActivityMain3Binding  
lateinit var lightSensor: Sensor  
lateinit var sm: SensorManager  
  
override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 binding = ActivityMain3Binding.inflate(*layoutInflater*)  
  
  
 sm = getSystemService(Context.*SENSOR\_SERVICE*) as SensorManager  
 lightSensor = sm.getDefaultSensor(Sensor.*TYPE\_LIGHT*)  
  
 if (lightSensor == null) {  
 Toast.makeText(this, "this device don't have light sensor", Toast.*LENGTH\_SHORT*).show()  
 }  
 setContentView(binding.*root*)  
}  
  
override fun onResume() {  
 super.onResume()  
 sm.registerListener(this, lightSensor, SensorManager.*SENSOR\_DELAY\_NORMAL*)  
}  
  
override fun onPause() {  
 super.onPause()  
 sm.unregisterListener(this)  
}  
  
override fun onSensorChanged(p0: SensorEvent) {  
 *TODO*("Not yet implemented")  
 if (p0.sensor.*type* == Sensor.*TYPE\_LIGHT*) {  
 val currentReading = p0.values[0]  
 //lightmeter.progress = currentReading.toInt()  
 //reading.text = ("Current Reading: "  
 // + (currentReading).toString() + " Lux")  
  
 }  
}  
  
override fun onAccuracyChanged(p0: Sensor?, p1: Int) {  
 *TODO*("Not yet implemented")  
  
}