**Intro to Mobile Platforms:**

-Apple: Programmed with Swift/Objective C

-Android: Programmed with Java/Kotlin

-Windows(lol): Programmed with C#/C++/Basic

Tablets/Watches/Phones.

**Market share:**

Global: Android 74%, IOS 25%, other <1%

US: Android 45%, iOS 55%, other <1%

**Android open source project:**

Created by Andy Rubin, acquired by Google in 2005. Google provides online documentation, tools, forums, SDK.

Pros:

-Java

-Cheaper developer license (25$ once, Apple charges 99$ a year)

-More open app store

-Open source (source.android.com)

-Widely available development environment

-Larger global community

-Many different manufacturers, first was HTC under T-Mobile in 2008

-Made popular by apps such as email, browsers, and google play store (formerly Android Market)

Licensed under GNU GPLv2 “strong copyleft” anyone who creates a fork and modifies Android must keep their versions in the public domain, as they stay under the same license.

Framework distributed under the Apache Software License (ASL/Apache2)

**Android as a System:**

-Built on linux

-Each app runs in its own VM known as Dalvik, and each app gets its own linux user id and file space.

-Android Programming Language Choices:

Java

Kotlin

C/C++

WebView(HTML/CSS/JS)

\*Review latest android version names\*

\*Event dispatcher class: “Looper”, used in a AsyncTask\*

-Layers:

Application code

Java Runtime Environment

Native C Libraries/Android

HAL (Hardware abstraction layer)

Linux Kernel

-Can use Google APIs

**8/23/22**

**Java review:**

A class that extends another inherits everything from that parent class, but not the constructors.

Up-casting: Initializing a new parent class object with the value of an existing child class object.

Down-casting: Initializing a new child class object with the value of an existing parent class object. Not always legal, and always requires an explicit type casting {Car c = (Car) vehicleObject;}.

Enumeration over a list:

for(Shape s : shapes){

doStuff();

}

An interface can extend other interfaces.

Nested classes: inner class/static nested class. An inner class has access to the methods of the enclosing class, but the static nested class does not. Inner classes can be private.

Anonymous class: a class that is defined and instantiated at the same time, such as event listeners.

**Graphical User Interfaces:**

Event-driven system: software system designed around responding to different events, like key presses, mouse movements, a timer progressing, or an arrival of a message.

Event handler: Fragment of code that defines a response to an event.

Event dispatcher: Control software for an event-driven system. Runs a continuous loop which accepts events and invokes associated event-handlers. Exists in procedure-driven system code.

GUIs contain at least: Controls, events, and event handlers.

**8/25/22**

When an activity needs to be started, Android calls the onCreate() method for it. Once the activity is started up, Android places it at the top of the Android activity stack. When the activity is done running, it is removed from the stack and the next activity on the stack is resumed.

Android activity lifecycle methods: onCreate(), onRestart(), onStart(), onResume(), onPause(), onStop(), onDestroy(). **Should override at least onCreate().** OnSaveInstanceState() is used to save the UI state into a **Bundle** when the app might be restarted. **OnSaveInstanceState()** is used right after onPause(), at times such as when the user presses the home button, shifts orientation, etc. The Bundle object can be accessed by **onRestoreInstanceState()**, which is called right after onStart().

**Android View class:** Base class for most components in an android application.

**R class:** giant class created at compilation by Android for an application.

**Android logging:** Logging user interface available in the Log class of the android.util package.

Example:

Import android.util.log;

Private static final String DEBUG\_TAG = “Go dogs”;

Log.i(DEBUG\_TAG, “MainActivity.onCreate()”);

Log.e = log errors, .w = warning, .i = info, .d = debug, .v = verbose.

**8/30/22**

The DEBUG\_TAG in passed to the logging method is a string.

Verbose logging tag includes all other logging tag messages.

**Button:** To add a listener/eventHandler to a button,

**8/31/22**

**DP units:** Density-independent pixels/device-independent pixels. Used as a resolution-independent unit. 1 DP = 1/160 of an inch

**Desktop publishing point:** 1/72 of an inch

**Printing press point:**

**DPI:** Dots/pixel per square inch. Used to measure pixel density.

**SP:** Scale-independent pixels

**9/6/22**

**Margin:** space outside of a frame

**Padding:** Space inside of a frame.

**Raw resources** folder:

Located in “app > res > raw” folder

Accessed using:

“Resources resources = getResources();

InputStream is = resources.openRawResource(R.raw.fileNameNoExtension);”

**Spinner component:** A searchable dropdown menu of sorts, must have labels, but not a TextEdit.

Each item in a Spinner is a TextView when selected. Used by:

“Spinner spinner = (Spinner) findViewById(R.id.Spinner1);

TextView selectedTextView = (TextView) spinner.getSelectedView();

String selectedText = selectedTextView.getText().toString();”

**ImageView component:**

-Used to display an image/GIF, will support most formats but designed to work with png.

-scaleType attribute controls where it is displayed (e.g. fitStart, fitCenter, matrix, fitXY, centerCrop, etc.)

**Other types of Android controls:** Switch, Button, CheckBox, ToggleButton, RadioButton, DatePicker, NumberPicker, TimePicker.

**More components:** ProgressBar (usable with bar.setProgress(int) method), Slider, SeekBar (Slider, like a progress bar but adjustable y the user), DigitalClock, TextClock, AnalogClock, RatingBar, Chronometer,

**9/8/22**

**Activity Transitions with Intents**: Tell Android “I would like to do yada-yada, can you do that for me?”

Handling permanent Activity transitions: startActivity(), finish().

-”startActivity(new Intent(getApplicationContext(),MyTargetActivity.class));”

Intents can contain Extras contained in a Bundle object.

Handling temporary transitions: registerForActivityResult(), ActivityResultLauncher()

**9/14/22**

**XML (Extensible Markup Language)**

-An evolution of SGML (Standard generalized markup language)

-Managed by W3C, XML 1.0 recommended by them in ?1998?

-Uses semi-structured data.

PCDATA = Parsed Character Data

Elements/Tags contain data, including recursively containing other elements/tags.

Attributes provide extra information about elements. Attributes are passed within the opening element tag: <note title=”Bob’s note”>

-Uses .dtd files to define types, or xml schema as an alternative:

<xs:attribute name=”name” type=”type”/>

**XML Parsers:** Stream-oriented parsers/APIs accessible from a programming language, like SAX and StAX; Tree-traversal APIs accessible from a programming language, for example DOM (Document Object Model); XML data binding, which provides an automated translation between XML documents and programming-language objects.

RDF: Resource Description Framework.

Pubmed: A massive index of all scientific medical documents.

**9/15/22 Positioning with Layouts**

Types of layouts: FrameLayout, LinearLayout, TableLayout, RelativeLayout, ConstraintLayout.

**ViewGroup:** A special kind of view used to format/contain multiple views simultaneously. Child views inside the ViewGroup inherit attributes from the parent ViewGroup unless explicitly overridden Examples: ListView, RecycleView, …

**9/20/22**

**RelativeLayout:** An older layout, used to be the layout of choice but has been replaced by ConstraintLayout. Works by giving boolean attributes to child views within the layout container that dictate its position within its parent layout. Examples of these attributes include: layout\_centerInParent, layout\_leftOf, etc.

**FrameLayout:** Displays a stack of child view items. The most recently added child will be drawn on top. The size of the frame layout is the size of the largest child. Attributes include: foreground (Parent), foregroundGravity (Parent), measureAllChildren (Parent), layout\_gravity (Child). You can attach fragments to a FrameLayout.

**TableLayout:** Organizes children into rows. A TableLayout should be the parent of some TableRow children.

**GridLayout:** Kind of like CSS Flex. Supposed to be superceded by the ConstraintLayout.

**USING CONTAINER CONTROL CLASSES:**

ViewGroup container types: Lists and grids, ScrollView, HorizontalScrollView, ViewFlipper, etc.

**ListView:**

**GridView:**

**AdapterView:** do things like providing values for a ListView.

ArrayAdapter: Provides values from a list or array of values.

“ArrayAdapter<String> itemsAdapter = new ArrayAdapter<String>(v.getContext(), android.R.simple\_list\_item\_1, states);//States is an array of the values you want to list

listView.setAdapter(itemsAdapter);”

You can use an onItemClick() within your ListView fragment to listen for if a specific item on a list is clicked.

CursorAdapter: Provides values from a cursor, such as in a database.

**ConstraintLayout:** Slower than a relative layout, but preferred by our professor. Does not use nesting. Shares some features with LinearLayout. Attributes such as “layout\_constraintLeft\_toRightOf, layout\_constraintTop\_toBottomOf” etc.

A **constraint** defines how to place a view constrained relative to other stuff. At least one vertical and one horizontal constraint must be specified for each view.

**Alignment constraints:** align edges of views together with attributes such as “layout\_constraintTop\_toTopOf, layout\_constraintRight\_toRightOf” etc.

**Guidelines:** Used to manage constraints.

**Barrier:** An invisible guideline that moves in response to view sizing. It moves in conjunction with the most extreme/dominant view.

**Chains**: A linear group of views. A chain can be spread (evenly distributed), packed(condensed inside), spread inside (First and last views are fixed to constraints), or weighted. To put views in a chain, select them all, right click, go to “Chains”, and create the type of chain needed.

**9/22/22**

**Flutter**

**Cross-platform development**: Used to make an application that works on Android and iOS, as well as web deployment.

-Created in 2017 by Google, looking to pass React Native as an option for non-native development.

-Uses the Dart language, which is object oriented.

-Flutter development can be done on Xcode for Mac, Android Studio for Windows, or Visual Studio Code (or cmd).

-Verify flutter installation with the “flutter doctor” tool.

-Unlike Android development, layouts in Flutter are mixed in with the logic/code.

-Flutter uses **widgets** as children of a layout to build a UI.

-StatelessWidget does not have a mutable state, it stays intact after it has been created.

-StatefulWidget has a state

-Widget types: Icon, Image, Text, TextField, Buttons of different types, CheckBoxRadioBox, Slider, Date/Time pickers, etc.

-Flutter **layouts**: Row, Column, GridView, Flow, ListView, Table, Stack. Can be nested, like in Android.

**Dart Programming Language:**

Object Oriented language with single-inheritance, interfaces and abstract classes are available. Dart also uses packages.

-Dart classes have constructors

-Classes have instance variables and methods (which can be static).

Methods: Parameters are positional, OR named; named parameters are required or optional:

**method (int positional, {required int namedRequired, String? NamedOptional})**

**Example call: method(11, name: ‘Jim’, id: 12345)**

optional positional parameters: method(int a, [String name, int id])

In “\_Thing” the underscore makes it private.

The => operator is a function that executes the expression to its right and returns its value.

**9/28/22**

Every Activity has an Android-defined FrameLayout with the identifier @+id/content.

Use the androidx Fragment import

Fragment “commit” method

**Example Widget:**

class PushCounter extends StatefulWidget{

@override

Widget build(BuildContext context){

return MaterialApp(

title:’Push Counter’,

theme: ThemeData(

primarySwatch: Colors.blue,

),

home: Scaffold(

appBar: AppBar(

title: const Text(‘Push Counter’),

),

body: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

PushCounterWidget(),

], ) ), ); } }

**9/27/22 Fragments**

-Cannot exist on their own, always hosted within an activity.

-Implemented by an Android class or a custom class inheriting from the Android class.

-Fragment class must be created, and then connected to an activity using transactions.

-A **FragmentTransaction** is used to perform updates or modifications to a fragment.

-Fragment constructors can’t take in variables, because Android must be able to reconstruct the fragment with the default parameter-less constructor.

-Nesting can be used with fragments

**Fragment lifecycle:** onCreate, onAttach, onCreateView, onViewCreated, onViewStateRestored, onStart, onResume, onPause, onStop, onSaveInstanceState, onDestroyView, onDestroy, onDettach.

**FragmentManager:** Can be retrieved with the getFragmentManager() method, available from the Activity and Fragment classes.

-Fragment is loaded into an activity using the setContentView() method in the onCreate() method.

-FrameLayout can be used as a placeholder for a Fragment.

Two side by side fragments updating each other:

ListFragment:

onCreate() = non-graphical initializations

onCreateView() = create the fragment views in here

**Inflate method:** Called

Fragment constructors do not have parameters, parameters can only be passed using a sent Bundle with setArguments() and getArguments(). Any use of setArguments() must occur before the fragment is attached to an activity.

**Android Navigation scenarios:** Entry, lateral, descendant, back, ancestral, or external navigation.

Entry navigation: How a user navigates into your application (from home, app list, etc.)

Lateral navigation: Transitioning between screens at the same hierarchical level. (Like between activities with Intents

Descendant navigation: Navigating to a different hierarchical level. (Usually done by creating a new activity and referencing parent activity)

Back navigation: Navigates the user to the immediately previous activity or fragment from the back stack. To override this behavior, call onBackPressed() from within your activity. For a fragment to be placed on the backstack, it must be added manually with addToBackStack().

Ancestral navigation: Going up one level. App must indicate the parentActivityName. (Different from back button because it skips previously visited lateral activities until it gets to a parent).

External navigation: Used when navigating out of an activity and into a different application. To return a resulting value from an external application, use ActivityResultLauncher and an overridden callback onActivityResult().

**Navigational Design Patterns:** Targets, swipe views, tabs, navigational drawer, or

Targets:

Swipe views: Always connected to fragments. Implemented with ViewPager/ViewPager2

Tabs:

Navigational Drawer: “Hamburger view” pop up side nav menu from button in top left corner. Implemented with DrawerLayout from the AndroidX JetPack library.

**ActionBars:**

Using getSupportActionBar() will return the ActionBar that is default for every activity.

Toast.makeText **????**

A **Toolbar** can be used in place of an ActionBar with “setSupportActionBar(toolbar);”

-Toolbars can be placed anywhere on the screen.

**Persistent storage:** Can save in the Android file space, or to an external service.

**Floating Action Button:** Uses an on click listener like normal buttons

**“Navigation Component”:** Introduced component intended to provide a consistent feel while navigating around activities and fragments. Android Studio features a graph editor to design navigation graphs with

**Dialogues:** Used when the application needs to confirm or acknowledge an action a user has taken that will permanently alter the user’s data. All DiaglogueComponents use show(), dismiss(), and onDismiss(). Create an AlertDialogueFragmetn class which extends DialogueFragment, and use setTitle(), setPositiveButton(), setNegativeButton(), setIcon(), and setMessage() to customize your dialogue from the onCreate method. Then use the below code to create an alert:

“DialougFragment dFragment = AlertDialogueFragment.newInstance();

showDialogueFragment(dFragment);”

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Accessing Android Files and Directories**

Android gives each user/application its own file space.

Files can be used in an app’s internal private area by the app while it is running, but not before.

Apps have internal/external cache storage, which store temporary data.

Structured data can be stored locally or externally: SQLite database, Remote database, etc.

Reading/writing from disk can be a blocking operation that causes an application to hang/freeze; so database operations should not be done on the main (UI) thread of the application).

**Good file management:** Release resources when done (close streams), check resource availability before using them,

**MIDTERM REVIEW**

~20 questions

Review:

-Attributes and what they do.

-Go through review sheet on eLC.

The **synchronized** java method header ensures that a method is only running on one thread at any given time.

**Using SQL Lite:**

In onResume(), open the db.

In onPause(), close the db.

Longer operations should take place asynchronously as a background thread.

Cursor cursor = null;

cursor = db.query(db, allColumns, null,null,null,null,null);

Must extend SQLiteOpenHelper and override

public void onCreate(SQLiteDatabase db){

db.execSQL(CREATE\_JOBLEADS);

}

and

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){//Runs whenever the //database has been updated

db.execSQL(“drop table if exists ”+TABLE\_JOBLEADS);

onCreate(db);

}

JobLeadObjectDBHelper: only one instance.

**POJO** object: Plain Old Java Object: a java class used to represent an object in a database, like a “JobLead” class in a jobleads table. You can use another class “JobLeadsData” to use for abstracting storage when transfering between the obect and the database, usually contains a constructor, open(), close(), retrieveAllJobLEads(), and storeJobLead(JobLead jobLead) methods.

**Sharing project on github: Import with VCS, VCS menu → share project with github**

**Async task:**

package edu.uga.cs.jobstrackersqlite;  
  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
import android.os.Handler;  
import android.os.Looper;  
  
  
*/\*\* This is a replacement class for the Android's deprecated AsyncTask class.  
 \* However, handling of the task's progress has not been implemented.  
 \* It uses Java's standard concurrency framework.  
 \** ***@param*** <*Param*> *type of the input parameter for doInBackground  
 \** ***@param*** <*Result*> *type of the result value returned by doInBackground  
 \*/*public abstract class AsyncTask<Param,Result> {  
  
 // An internal method to execute something in background  
 private void executeInBackground( Param... params ) {  
  
 // Get en executor service -- it will serve to run the task  
 // in the background, i.e., in this executor. This executor service  
 // has its own thread, which is different than the main UI thread.  
 // As a result, anything running on this thread will not block the UI.  
 ExecutorService executor = Executors.*newSingleThreadExecutor*();  
  
 // Execute a Runnable with the task on the executor service.  
 // Runnable is a Java interface to be implemented by a class to execute in  
 // a thread. The run() method is called automatically when a Thread is given  
 // a Runnable object.  
 //  
 // The anonymous Runnable class below will execute the method body (doInBackground)  
 // in the executor service, which uses a different thread than the main UI thread.  
 // Once the result is obtained, the Runnable class will add another Runnable  
 // with the call to onPostExecute with the Result argument to be added to the  
 // main UI thread. The main UI thread will update the UI accordingly.  
 // Since the Runnable below will execute in a different thread, the main UI thread  
 // will not be blocked.  
 executor.execute( () -> {  
  
 // Run the method body (doInBackground)  
 Result result = doInBackground( params );  
  
 // Now, pass the result to the main UI thread  
 //  
 // Get the looper of the UI thread (the main UI event dispatcher's loop)  
 // A Looper is simply a message queue within the Android OS.  
 // If you are curious, you can examine the source code of the Looper class here:  
 // https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/os/Looper.java  
 Looper looper = Looper.*getMainLooper*();  
  
 // Create a Handler using the main UI's looper.  
 // A Handler is used to interact with a Looper, for example,  
 // for posting messages on the main Looper's queue.  
 // If you are curious, you can examine the source code of the Handler class here:  
 // https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/os/Handler.java  
 Handler handler = new Handler( looper );  
  
 // Post the processing of the result of the doInBackground method  
 // on the main UI thread's looper.  
 handler.post( new Runnable() {  
 @Override  
 public void run() {  
 // handle the method result in the main UI threa  
 onPostExecute( result );  
 }  
 });  
 });  
 }  
  
 // This method is just like in the AsyncTask class  
 public void execute( Param... arguments ){  
 executeInBackground( arguments );  
 }  
  
 // These abstract methods are just like in the AsyncTask class  
 protected abstract Result doInBackground( Param... arguments );  
 protected abstract void onPostExecute( Result result );  
}

**Web Services:**

In our example the JobLeads will be stored remotely and accessed using an API.

RPC (Remote Procedure Call): RPCRuntime intakes/outputs a packed request and sends it to or receives it from another RPCRuntime over the network.

Distributed Objects: RPC but built around remote objects: CORBA, DCOM, Java RMI

Marshalling: Packing procedure (function) parameters into a message packet.

Binding: Process of connecting a client to a server.

SOAP (Simple Object Access Protocol): predecessor of RESTful services. 3 part messages: envelope, header, and body.

REST:

HTTP methods: GET, POST, PUT, DELETE, HEAD (GET but no response body).

WSDL (Web Services Description Language): Used to describe web services.

UDDI: Used to find what web services are available.

**REST Services/APIs**: Requests limited to GET, POST, PUT, and DELETE.

REST = Representational State Transfer. Named by Roy Fielding.

Resources = things/entities (products, customers, shopping carts, university courses, etc.)

Safe method: Does not change the state at all (like a println statement)

Idempotent: Methods can still be repeated, but it changes the state (like x=41)

Non-idempotent: Methods cannot be repeated without getting a different result, misuse can cause issues (like x++)

GET = safe, return all customers

POST = not idempotent, create a new customer and return new customer URI

PUT = idempotent, update what you already have

DELETE = unsafe, delete what you already have.

**Software Engineering Review:**

Verification: Testing that the software meets specifications provided by the users

Validation: Making sure the software fills the needs of the users

**Google Firebase: Realtime database**

Firebase:

-All data is stored as JSON objects

FirebaseDatabase database = FirebaseDatabase.getInstance();

DatabaseReference myRef = database.getReference(“message”);

myRef.setValue(“Hello :)”);

“onDataChange()” called any time the value of your reference changes.

“addOnCompleteListener(…)” runs when firebase completes the action this method is called on.

Types of NoSQL databases:

-document stores (like Firebase)

-Graph database: Joe is a node, his car is a node, an edge connects them that shows he owns the car.

**Final project:**

App 1: Shopping list

Require

Use at least 2 lists on firebase: needed items, purchased items

App 2: Ride sharing

//Adding an item to a database list

database.getReference().push().setValue(Object).addOnSuccessListener(new onSuccessListener<Void>(){

public void onSuccess(Void void){ … }

}).addOnFailureListener( … );

//Getting all objects of a query from the database

reference.addValueEventListener(new ValueEventListener(){

public void onDataChange(@NonNull DataSnapshot snapshot){

jobLeadsList.clear();//inefficient

for(DataSnapshot postSnapshot: snapshot.getChildren()){

JobLead jobLead = postSnapshot.getValue(JobLead.class);

jobLead.setKey(postSnapshot.getKey()); //every item has a unique key created by firebase

jobLeadsList.add(jobLead);

Log.d( … );

}

Log.d(…);

recyclerAdapter.notifyDataSetChanged();//notify the recyclerView that the list has been updated so it can reload

}

public void onCancelled(@NonNull DatabaseError databaseError){ … }

});

//In recyclerAdapter onBindViewHolder:

holder.itemView.setOnClickListener(new View.OnClickListener() {

public void onClick(View v){

EditJobLeadDialogFragment editJobFragment = ...

}

}

**Communicating from dependent fragment back to parent activity with a listener:**

EditJobLeadDialogFragment contains example with listener.updateJobLead(), the listener is implemented in the ReviewJobLeadsActivity class in updateJobLead.

**UI Storyboards:** Mockups of the user interface, used to plan the layout, look, and feel of your app.

**:** Used to map user flows and the links between pages/activities of an app.

**Resources:**

res/drawable (Graphics): drawable-hdpi = high density, -mdpi = medium, -ldpi = low, -xhdpi = extra high, -xxhdpi = 480dpi, -xxxhdpi = 640dpi.

res/layout (UI resources)

res/menu (menu resources for showing options/actions in your activities)

res/mipmap (app launcher icon resources): Add new with New→Image Asset

res/values (simple data like strings, objects): Used to store primitive resource types. XML files names are arbitrary, but best practice is to store data types in corresponding files by name like “strings.xml” or “bright\_colors.xml”. Can hold bool, integer, string-array, string, etc.

-You can use some html tags in strings, such as <i></i>, <b></b>, and <u></u>.

-String quantity handling words: zero, one, two, few, many, and other.

**More resource types:**

res/animator

res/anim

res/color

**Review:**

**Application Components:**

-Activities transition from state to state depending on what the user does.

-(onCreate(), onStart(), onResume(), →( onPause() onStop(), onRestart(), onStart(), onResume() )OR onPause(), onStop(), onDestroy().

-Transition between activities with Intents.

-XML: know tags, elements, and attributes.

Example namespaces: android, app

-DTD:

Padding: space inside a box.

Margin: space around the outside of a box.

Device independent pixels (dpi): 1dpi = 1inch/160

Scale-independent Pixels for fonts (sp):

-**Review attributes of common views we covered in the course/projects.**

LinearLayout:

ConstraintLayout: Each child view must have at least one horizontal constraint and vertical constraint.

Bias: used to adjust which constraints have the biggest pull on a view. Change bias ratios to change the position of a view being constrained.

Guidelines:

Barriers:

FrameLayout: Placeholder for a fragment;

TableLayout:

-wrap\_content = adjust layout using gravity to influence children; match\_parent = inherit layout from parent and fill available space.

ArrayAdapter:

RecyclerAdapter:

Fragments: Built to be reusable pieces of layouts. Used for orientation changes, variations of pages, etc.

-lifecycle: onCreate(), onViewCreate(), onViewCreated(),

-Fragments must be inflated.

**Navigational patterns:**

Distinguish between back and up navigation:

Understand and describe basic navigational patterns:

-Target navigation: lateral, descendant, ancestral

-Swipe views:

-Tabs: Similar to swipes, useful when you have a small number of choices.

-Navigational drawer: Pick a choice on where to go.

-Master detail flow: List of choices on the left, details of selected choice displayed on the right.

-Action bar, menus, action buttons:

-Floating action button:

-Dialogues fragments:

**Android files and directories:**

Every app is a separate linux user with its own filespace, as well as its own private cache space.

Filespace reservations are maintained until the app is uninstalled.

Private cached files are kept until the temporarily until the app closes, the app removes them, or android needs the space.

\*Review file permissions\*

\*Review assets folder\*

**Relational databases and SQLite**

SQLite database included in Android as an embedded database.

-Contains 4 data types as well as null: (INTEGER, REAL, TEXT, and BLOB)

-SELECT, INSERT, UPDATE, and DELETE.

Primary key: Unique value or composition of values that identifies each row in a table.

Object to Relational Mapping (ORM): Each cell will only hold one value,

SQLiteOpenHelper: Define the database schema and update it.

Understand Asynchronous method calls.

Review RecyclerView.

**Web Services:**

Know the use of GET, POST, PUT, and DELETE.

Understand JSON notation

**System Development:**

Review user stories and their confirmation criteria.

Understand the software development process.

Understand software development methodologies, specifically the Scrum methodology.

Review personas, screen maps, screen layouts, UI sketches/wireframes, and UI storyboards.

**Themes and styles:**

Understand the difference between a theme and a style: A theme is a collection of styling design changes that are applied as a whole to an application or activity. A style is applied to a specific view type.