What is Android?

- Android is a lightweight OS aimed at mobile devices.
- It is essentially a software stack built on top of the Linux kernel.
- Libraries have been provided to make tasks both simpler and efficient in terms of memory and power.
- Each application gets to run in its own VM on top of the Linux kernel. This provides protection and security.
- Applications are usually written in Java, though native interfaces exist.
- Applications can identify and execute code for the state they are in using callbacks provided by system. Examples are methods like `onCreate()`, `onResume()`, `onStop()` etc.
Android Architecture

**Applications**
- Home
- Contacts
- Phone
- Browser
- ...

**Application Framework**
- Activity Manager
- Window Manager
- Content Providers
- View System
- Package Manager
- Telephony Manager
- Resource Manager
- Location Manager
- Notification Manager

**Libraries**
- Surface Manager
- Media Framework
- SQLite
- OpenGL | ES
- FreeType
- WebKit
- SGL
- SSL
- libc

**Android Runtime**
- Core Libraries
- Dalvik Virtual Machine

**Linux Kernel**
- Display Driver
- Camera Driver
- Flash Memory Driver
- Binder (IPC) Driver
- Keypad Driver
- WiFi Driver
- Audio Drivers
- Power Management
Application Components

- Application Components
  - Activities
  - Services
  - Broadcast Receivers
  - Content Providers
- Intents
- Activities and Tasks
- Process and Threads
  - Remote Procedure Calls
Activities

- An *activity* presents a visual user interface for one focused endeavor the user can undertake.
- Activity is implemented as a subclass of the **Activity** base class.
- The visual content of the window is provided by a hierarchy of views.
  - objects derived from the base **View** class.
- A view hierarchy is placed within an activity's window by the **Activity.setContentView()** method.
public class MainActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.fragment_main);

        Button btn = (Button) findViewById(R.id.button1);
        btn.setOnClickListener((v) -> {
            // TODO Auto-generated method stub
            callSecondIntent();
        });
    }

    public void callSecondIntent() {
        //...
    }

    @Override
    public boolean onCreateOptionsMenuMenu(Menu menu) {
        //...
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        //...
    }

    public static class PlaceholderFragment extends Fragment {
    }
Activity Lifecycle

- An activity has essentially three states
  - Active or Running
  - Paused
  - Stopped

- Activity Lifetime
  - Entire Lifetime [onCreate() to onDestroy()]
  - Visible Lifetime [onStart() to onStop()]
  - Foreground Lifetime [onResume() to onPause()]

- An implementation of any activity lifecycle method should always first call the superclass version
Activity Lifecycle

Activity starts

onCreate()

onStart()

onRestart()

onResume()

Activity is running

Another activity comes in front of the activity

onPause()

The activity comes to the foreground

The activity is no longer visible

onStop()

onDestroy()

Activity is shut down

User navigates back to the activity

Process is killed

Other applications need memory
Services

- A Service does not have a visual interface and runs in the background.
- Each service extends the `Service` base class.
- It's possible to connect to an ongoing service and communicate it through the interface exposed by that service.
- Services run in the main thread of the application process.

Examples
- Network Downloads
- Playing Music
- TCP/UDP Server
Service example

```java
public class serviceEx extends Service {
    public static handler handle = new Handler()

    @Override
    public void handleMessage(Message msg) {
    }

    @Override
    public void onCreate() {
        sendMessageToMainActivity();
    }

    public void sendMessageToMainActivity() {
    }

    @Override
    public IBinder onBind(Intent intent) {
    }
}
```

- **Service definition**
- **Message handler queue**
- **Create Method callback – For init**
Service Lifecycle

- A service can be used in two ways
  - `startService()` - `stopService()`
  - `bindService()` - `unbindService()`

- Service Lifetime
  - Entire Lifetime `[onCreate() to onDestroy()]`
  - Active Lifetime `[onStart()]`

- The `onCreate()` and `onDestroy()` methods are called for all services.
- `onStart()` is called only for services started by `startService()`.
Service Lifecycle

1. **Service is started by `startService()`**
   - `onCreate()`
   - `onStart()`
   - **Service is running**
   - **The service is stopped (no callback)**
   - `onDestroy()`
   - **Service is shut down**

2. **Service is created by `bindService()`**
   - `onCreate()`
   - `onBind()`
   - **Client interacts with the service**
   - `onUnbind()`
   - `onDestroy()`
   - **Service is shut down**

3. **Service is created by `startService()`**
   - `onCreate()`
   - `onStart()`
   - **Service is running**
   - **The service is stopped (no callback)**
   - `onDestroy()`
   - **Service is shut down**

4. **Service is created by `bindService()`**
   - `onCreate()`
   - `onBind()`
   - **Client interacts with the service**
   - `onUnbind()`
   - `onDestroy()`
   - **Service is shut down**

5. **Service is created by `startService()`**
   - `onCreate()`
   - `onStart()`
   - **Service is running**
   - **The service is stopped (no callback)**
   - `onDestroy()`
   - **Service is shut down**

6. **Service is created by `bindService()`**
   - `onCreate()`
   - `onBind()`
   - **Client interacts with the service**
   - `onUnbind()`
   - `onDestroy()`
   - **Service is shut down**
Broadcast Receivers

- A broadcast receiver receive and react to broadcast announcements.
- All receivers extend the BroadcastReceiver base class.
- Many broadcasts originate in system code.
- Broadcast receivers do not display a user interface but they can start an activity or alert user.
Broadcast receiver Lifecycle

- A broadcast receiver has single callback method
  - `onReceive()`
- The lifetime of a broadcast receiver is only during the execution of `onReceive()` method.
- A process with an active broadcast receiver is protected from being killed.
Broadcast receiver example

```java
public class MyReceiver extends BroadcastReceiver{
    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO Auto-generated method stub
        Toast.makeText(context, "intent received", Toast.LENGTH_LONG).show();
    }
}
```
Content Providers

- A *content provider* makes a specific set of the application's data available to other applications.
- It’s the only way to transfer data between applications in Android (no shared files, shared memory, pipes, etc.)
- All content providers extend the `ContentProvider` base class.
- Content Providers are accessed through `ContentResolver` object.
- Content Providers and Content Resolvers enable inter-process communication (IPC)
public class MainActivity extends ActionBarActivity {
    SQLiteDatabase dB;
    EditText patientID;
    EditText Age;
    String patientIDText;
    String ageText;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.fragment_main);

        patientID = (EditText) findViewById(R.id.editText1);
        Age = (EditText) findViewById(R.id.editText2);
        Button btn = (Button) findViewById(R.id.button1);
        btn.setOnClickListener(new View.OnClickListener() {
            // TODO Auto-generated method stub
            try {
                dB = SQLiteDatabase.openOrCreateDatabase(Environment.getExternalStorageDirectory() + "\databaseFolder/myDB", null)
                db.beginTransaction();
                try {
                    //perform your database operations here ...
                    db.execSQL("create table tblPat ("
                        + " recID integer PRIMARY KEY autoincrement, "
                        + " name text, "
                        + " age text ); "");
                }
                catch (SQLException e) {
                    // catch exception
                }
                db.setTransactionSuccessful(); //commit your changes
            } finally {
                if (db != null) {
                    db.close();
                }
            }
        });
    }
}
Intents

- Intents are Asynchronous messages used to convey a request or message.
- An intent is an object of `Intent` class that holds the content of the message.
- Activities, Services and Broadcast Receivers are activated through Intents.
- Intent can contain
  - Component name
  - Action
  - Data
  - Category
  - Extras
  - Flags
Intent Examples

```java
public void callSecondIntent(){
    Intent intSecond = new Intent(this, SecondActivity.class);
    startActivity(intSecond);
}

public void callService(){
    Intent intService = new Intent(this, serviceEx.class);
    startService(intService);
}

public void callBroadcastMessage(){
    Intent intBroad = new Intent();
    intBroad.setAction("com.tutorialspoint.CUSTOM_INTENT");
    sendBroadcast(intBroad);
}
```

Intent to start Activity
Intent to start Service
Intent to start Broadcast listener
Activities and Tasks

- Task is a group of related Activities, arranged in a stack.
- Task is what the user experiences as an application.
- The activity at the top of the stack is one that's currently running.
- All the activities in a task move together as a unit.
Process and Threads

- Android starts a Linux process for each application by default.
- By default, all components of the application run in main thread of application’s process.
- However, you can spawn separate threads for long operations.
- Threads are created in code using standard Java Thread objects.
Threads example

```java
@Override
public void onStart() {
    super.onStart();

    Thread threadRun = new Thread((Runnable) -> {
        // TODO Auto-generated method stub
        try {
            for (int i = 0; i < 10; i++) {
                Random rnd = new Random();
                Message msg = threadHandle.obtainMessage(1, Integer.toString(rnd.nextInt(101)));
                threadHandle.sendMessage(msg);
            }
            myHandler2.post(foeGroundTask);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    });

    threadRun.start();
}

Handler myHandler2 = new Handler();
private Runnable foeGroundTask = new Runnable() {

    @Override
    public void run() {
        // TODO Auto-generated method stub
        ProgressBar pgBar2 = findViewById(R.id.progressBar1);
        pgBar2.setProgress(0);
        pgBar2.setMax(100);
        while (pgBar2.getProgress() != pgBar2.getMax()) {
            try {
                Thread.sleep(10);
                Toast.makeText(MainActivity.this, "Hi from thread 2", Toast.LENGTH_SHORT).show();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            pgBar2.incrementProgressBy(5);
        }
        Toast.makeText(MainActivity.this, "Thread 2 has ended", Toast.LENGTH_SHORT).show();
    }
};
```
Remote Procedure Calls

- Android has a lightweight mechanism for remote procedure calls (RPCs).
- An RPC interface can include only methods.
- RPC Interface id declared using IDL and Java interface definition is generated by `aidl` tool.
Shutting down components

- **Activities**
  - Can terminate itself via `finish();`
  - Can terminate other activities it started via `finishActivity();`

- **Services**
  - Can terminate via `stopSelf();` or `Context.stopService();`

- **Content Providers**
  - Are only active when responding to `ContentResolvers`

- **Broadcast Receivers**
  - Are only active when responding to broadcast
public class MainActivity extends ActionBarActivity {
    String phoneNum;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.fragment_main);

        Button btn = (Button) findViewById(R.id.button1);
        btn.setOnClickListener((v) -> {
            // TODO Auto-generated method stub
            Intent startSenseService = new Intent(MainActivity.this,sensorHandlerClass.class);
            Bundle b = new Bundle();
            b.putString("phone", phoneNum);
            startSenseService.putExtras(b);
            startService(startSenseService);
        });

        EditText edT = (EditText) findViewById(R.id.editText1);
        edT.setOnEditorActionListener((v, actionId, event) -> {
            // TODO Auto-generated method stub
            phoneNum = v.getText().toString();
            return false;
        });
    }
}
Putting it together - Accelerometer eg.

```java
public class sensorHandlerClass extends Service implements SensorEventListener {

    private SensorManager accelManage;
    private Sensor senseAccel;

    float accelValuesX[] = new float[128];
    float accelValuesY[] = new float[128];
    float accelValuesZ[] = new float[128];

    int index = 0;
    int k = 0;

    Bundle b;

    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {...}

    @Override
    public void callFallRecognition(){...}

    @Override
    public void callGestureRecognition(){...}

    @Override
    public void sendSMS() {...}

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {...}

    @Override
    public void onCreate(){...}

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {...}

    @Override
    public IBinder onBind(Intent intent) {...}
}
```
Android Security Model

- Each Android Application runs in its own process which is a secure sandbox.
- Each Android package file installed on the device is given its own unique Linux user ID.
- The permissions required by an application are declared statically in that application.
- All Android applications must be signed.
- Any data stored by an application will be assigned that application's user ID, and not normally accessible to other packages.
Interaction between applications

- Application Sandbox
- Each application runs within its own UID and VM
- Default privilege separation model
- Instant security features
  - Resource sharing
  - CPU, Memory
  - Data protection
  - FS permissions
  - Authenticated IPC
  - Unix domain sockets
- Place access controls close to the resource, not in the VM
Security Awareness

- During installation, apps ask the user to confirm permissions to access some components of the device (Internet, Wifi, Contacts etc)
- Once this permission is granted, the app can use the components throughout its lifecycle.
- This raises concerns over the kind of permissions that an app can ask for and utilize throughout its life
- Classic example is the Facebook app requesting permission to Read text messages, Download files without notification, Read calendar events and confidential info, etc..
Android Manifest

- Its main purpose in life is to declare the components to the system
- Declares the activities and services
- Declares the permissions required
- Defined in xml format

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest>
  <application>
    <activity
      android:name="com.example.project.FreneticActivity"
      android:icon="@drawable/small_pic.png"
      android:label="@string/freneticLabel"
    >
      <!-- Activity content -->
    </activity>
  </application>
</manifest>
```
References

- Android Introduction, Mihail L Sichitiu, Google
- Slides from Victor Matos, CSU
- Android Fundamentals, Kurtis Nusbaum
- Android Development Training, Oiklum
- Android Tutorial, Larry Walters