# CSIT242 Mobile Application Development

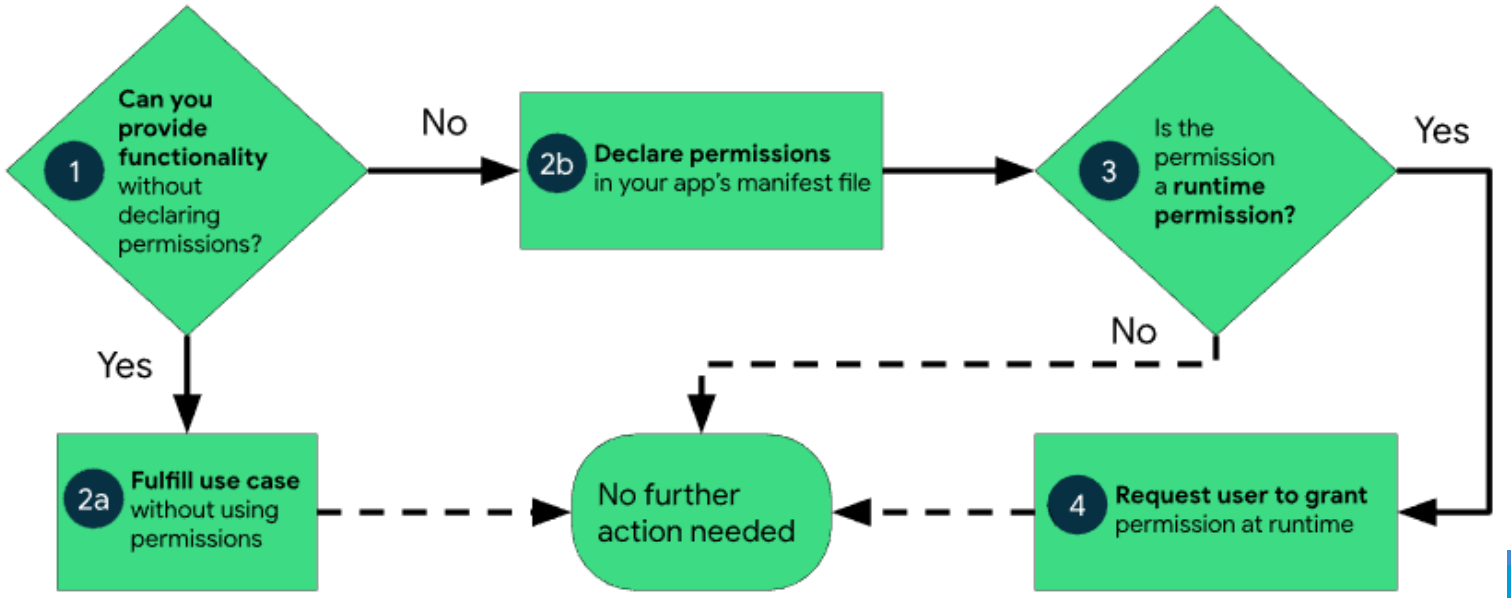
## LECTURE 6 – LOCATION MAP, SENSORS, FRAMEWORKS AND NETWORKING

# Outline

- Location maps and sensors
  - Android – Location & Google Map API
  - Android supported sensors
  - iOS – Location & Maps
  - iOS supported sensors
- Frameworks, networking, apps Testing & distribution
  - iOS Frameworks
  - Android - Connectivity
  - Networking
  - App testing and distribution

# Android – Requesting Runtime Permissions

- The purpose of a permission is to protect the privacy of an Android user
- Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet)
  - Depending on the feature, the system might grant the permission automatically (install-time permissions) or might prompt the user to approve the request (runtime permissions)



1 Can you provide functionality without declaring permissions?
No
2b Declare permissions in your app's manifest file
3 Is the permission a runtime permission?
Yes
No
Yes
2a Fulfill use case without using permissions
No further action needed
4 Request user to grant permission at runtime

https://developer.android.com/guide/topics/permissions/overview#normal

# Android – Requesting Runtime Permissions

- Install-time permissions - give limited access to restricted data or restricted
    - data or resources with very little risk to the user's privacy or the operation of other apps
    - app store presents an install-time permission notice to the user in app's details page
    - automatically granted at install time
  - Normal permissions - access to data and actions that extend beyond your app's sandbox
  - Signature permissions - grants permission to an app only when the app is signed by the same certificate as the app or the OS that defines the permission

- Run-time (Dangerous) permissions - the app wants data or resources that involve the user's sensitive information, or could potentially affect the operation of other apps
  - The user has to explicitly grant the permission to the app
  - The app must prompt the user to grant permission at runtime
  - These permissions can be revoked later on

- Special permissions

- Permission groups - consist of a set of logically related permissions

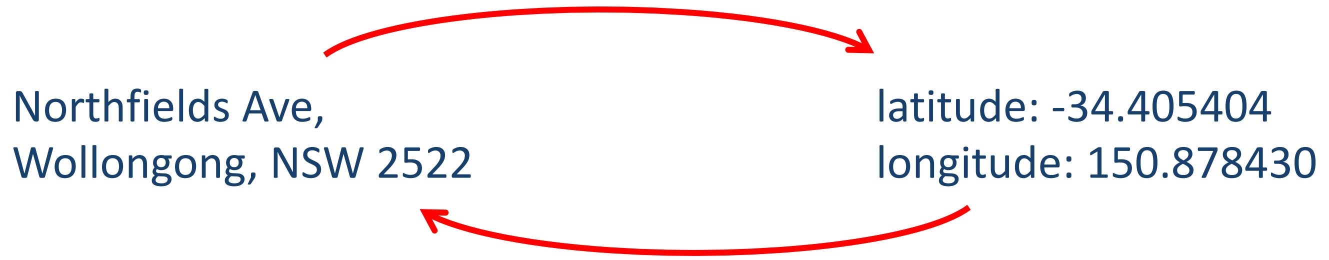
# Android – Requesting Runtime Permissions

- Request App permissions
  - declare that the app needs a permission by putting <uses-permission> element in the app manifest file

    ```
    <uses-permission android:name="android.permission.INTERNET"/>
    ```
  - check whether the permission is granted (before performing an operation that requires that permission) – use `ContextCompat.checkSelfPermission(…)` method
  - Provide an explanation if the user has already denied that permission request (`shouldShowRequestPermissionRationale()` returns true/false if the user has previously denied the request and/or selected the "Don't ask again option")
  - Request permission - use `requestPermissions(…)` method
  - When the user responds to the app's permission request, the system invokes the app's `onRequestPermissionsResult(…)` method

# Geocoding and Reverse-geocoding

- Latitude and Longitude are the units that represent the coordinates at geographic coordinate system, so we can specify virtually any point on earth
  - Just like every actual house has its address (which includes the number, the name of the street, city, etc), every single point on the surface of earth can be specified by the latitude and longitude coordinates
- Geocoding (forward-geocoding) is the process of converting a textual based geographical location (such as a street address) into geographical coordinates expressed in terms of longitude and latitude
- Reverse-geocoding involves the translation of geographical coordinates into a human readable address string



# Android – Location & Context

- Location and maps-based apps offer a compelling experience on mobile devices
- Applications that have "location-related" features are using the classes of the `android.location` package, Google Location Services API, or the Google Maps Android API
- Main component of the location framework (`android.location` package) is the `LocationManager` system service, which provides APIs to determine location and bearing of the underlying device (if available)
- The Google Location Services API, part of Google Play services, is the preferred API
  - a simpler API, higher accuracy, automated location tracking, low-power geofencing, and activity recognition
  - new features such as activity detection that are not available in the location framework API

# Android – Location permissions

- Apps that use location services must request location permissions
  - Android offers three location permissions:
    - `ACCESS_COARSE_LOCATION`
    - `ACCESS_FINE_LOCATION`
    - `ACCESS_BACKGROUND_LOCATION` (on Android 10+ / API level 29+)
  - Chosen permission determines the accuracy of the location returned by the API or access on the device location while the app is in the background

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package= ". . . " >
 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
 . . .
</manifest>
```

*If the app targets Android 10 or higher, must be declared the ACCESS_BACKGROUND_LOCATION permission in the app's manifest file and receive user permission in order to receive regular location updates while the app is in the background.*
*If the app doesn't require location access while running in the background, it's highly recommended not to request ACCESS_BACKGROUND_LOCATION.*

# Android – Getting the Last Known Location

- Using the Google Play services location APIs, the app can request the last known location of the user's device
- The **fused location provider** (location APIs in Google Play services) can be used for retrieving the device's last known location
- Fused Location Provider
  - manages the underlying location technology and provides a simple API so can be specified requirements at a high level (like high accuracy or low power)
  - optimizes the device's use of battery power

# Android – Getting the Last Known Location

- To get the last known location of the device, an instance of the `Fused Location Provider Client` needs to be created
- The `getLastLocation()` method can be used to retrieve the device location
  - The precision of the location returned by this call is determined by the permission setting in the app manifest
- The `getLastLocation()` method returns a *Task* that can be used to get a `Location` object with the latitude and longitude coordinates of a geographic location

```
fusedLocationClient.getLastLocation()
        .addOnSuccessListener(this, new OnSuccessListener<Location>() {
            @Override
            public void onSuccess(Location location) {
             //Got last known location. In some situations this can be null.
                if (location != null) {
                    // Logic to handle location object
                }
            }
        });
```

## Android – Getting the Last Known Location

- The `location` object may be null in the following situations:
  - Location is turned off in the device settings (disabling location also clears the cache)
  - The device never recorded its location (new device or restoring to factory settings)
  - Google Play services on the device has restarted - there is no active Fused Location Provider client that has requested location after the services restarted

- The applications that use location, besides the geographical location (latitude and longitude), may require to get:
  - the bearing (horizontal direction of travel), altitude, or velocity of the device
- These information are also available in the `Location` object (retrieved from the `fused location provider)`

# Android – Change Location Settings

- `LocationRequest` objects are used to request a quality of service for location updates from the FusedLocationProvider API

- `LocationRequest` include methods like:
  - setIntervalMillis(long) - sets the interval (milliseconds) for location updates
  - setMinUpdateIntervalMillis(long) – sets the minimal interval of location updates
  - setPriority(int) - sets the priority of the request:
    - Priority.PRIORITY_BALANCED_POWER_ACCURACY - request for location precision is balanced between location accuracy and power usage
    - Priority.PRIORITY_HIGH_ACCURACY – request for the most precise location possible
    - Priority.PRIORITY_LOW_POWER - request that favours low power usage at the possible expense of location accuracy
    - Priority.PRIORITY_PASSIVE - ensures that no extra power will be used to derive locations (will only receive locations calculated on behalf of other clients)

# Android – Receiving Location Updates

- Regular location updates can be started by calling `requestLocationUpdates()`
- Based on the form of the request, the `fused location provider` can
  - invoke the `LocationCallback.onLocationResult()` callback method and passes it a list of Location objects, or
  - issues a `PendingIntent` that contains the location in its extended data
- The location updates can be stopped when the activity is no longer in focus
  - This will reduce power consumption
- To stop location updates, can be called `removeLocationUpdates()` method

## Android – Displaying a Location Address

- To get the location from the address and address from location `Geocoder` class (from `android.location` package) can be used
  - The `getFromLocation()` method - accepts a latitude and longitude, and returns a list of addresses
  - The `getFromLocationName()` method – accept the name of a place ("Wollongong", " Wollongong, Australia”, "Northfields Avenue, Wollongong, Australia",…) and returns an array of `Address`es that are known to describe the named location
  - The returned `address` object will be localized for the locale (specific geographical, political, or cultural region), and can be used to extract the latitude and longitude of the address

# Android – Google Maps Android API

- The Google Maps Android API allows the inclusion of maps and customized mapping information in the app
- Using Google Maps Android API maps can be embedded into an activity as a fragment with a simple XML snippet
- Key developer features
  - 3D maps; indoor, satellite, terrain, and hybrid maps; vector-based tiles for efficient caching and drawing; animated transitions
  - add markers onto the map to indicate special points of interest for the users; define custom colours or icons for the map markers
  - draw polylines and polygons to indicate paths or regions; provide complete image overlays
  - ability to control the rotation, tilt, zoom, and pan properties of the "camera" perspective of the map
  - add Street View to the app

# Android – Google Maps Android API

- Maps, add to the app with the Google Maps Android API, are based on Google Maps data
  - The API automatically handles access to Google Maps servers, data downloading, map display, and touch gestures on the map
- The key class in the Google Maps Android API is MapView.
- A MapView displays a map with data obtained from the Google Maps service
- MapView advantages
  - provides all of the UI elements necessary for users to control the map
  - captures keypresses and touch gestures to pan and zoom the map automatically
  - the app can also use MapView class methods to control the map programmatically and draw a number of overlays on top of the map
  - can handle network requests for additional maps tiles

## Android – Sensors

- Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions
- These sensors are capable of providing raw data with high precision and accuracy, and are useful for monitoring three-dimensional device movement or positioning, or changes in the ambient environment near a device
  - For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing
  - A weather application might use a device's temperature sensor and humidity sensor to calculate and report the dewpoint

# Android – Sensors

- The Android platform supports three broad categories of sensors: Motion, Position and Environmental

- Motion sensors
  - This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors
  - Motion sensors are useful for monitoring device movement, such as tilt, shake, rotation, or swing
  - These sensors measure acceleration forces and rotational forces along three axes
  - The movement is usually a reflection of
    - direct user input (monitoring motion relative to the device's frame of reference or the application's frame of reference), or
    - a reflection of the physical environment in which the device is sitting (monitoring motion relative to the world's frame of reference)

## Android – Sensors

- Position sensors
  - This category includes orientation sensors and magnetometers
  - Geomagnetic field sensor and the accelerometer sensors measure the physical position of a device (in the world's frame of reference)
  - The Android platform also provides a sensor that  determines how close the face of a device is to an object (proximity sensor)
- Environmental sensors
  - This category includes barometers, photometers, and thermometers
  - These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity
  - With the exception of the light sensor, which most device manufacturers use to control screen brightness, environment sensors are not always available on devices

# Android – Sensors

- Some of the sensors are hardware-based and some are software-based
- Hardware-based sensors are physical components built into a handset or tablet device
  - They derive their data by directly measuring specific environmental properties, such as acceleration, geomagnetic field strength, or angular change
- Software-based sensors are not physical components, although they mimic hardware-based sensors
  - They derive their data from one or more of the hardware-based sensors and are sometimes called virtual sensors or synthetic sensors
  - The orientation sensor type is examples of software-based sensors
- Depending on the device some sensors can be both hardware of software based, or the device can have more than one sensor of a given type
  - For example gravity sensor can be hardware or software based; also, a device can have two gravity sensors, each one having a different range

# Android – Sensors

- Android sensor framework (part of `android.hardware` package) allows access to the sensors (available on the device) and raw sensor data
- The sensor framework provides several classes and interfaces for performing a wide variety of sensor-related tasks like:
  - Determine which sensors are available on a device
  - Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution
  - Acquire raw sensor data and define the minimum rate at which you acquire sensor data
  - Register and unregister sensor event listeners that monitor sensor changes

*Sensor availability varies from device to device, but it can also vary between Android versions.*

## Android – Sensors

- The sensor framework is part of the `android.hardware` package and includes the following classes and interfaces:
  - `SensorManager` – used to create an instance of the sensor service
    - It provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, acquiring orientation information, and several sensor constants used to report sensor accuracy, set data acquisition rates, and calibrate sensors
  - `Sensor` - used to create an instance of a specific sensor
    - It provides various methods that let you determine a sensor's capabilities
  - `SensorEvent` – used to create a sensor event object
    - A sensor event object includes the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event
  - `SensorEventListener` - interface used to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes

## Android – Sensors

- Android does not specify a standard sensor configuration for devices
- Device manufacturers can incorporate any sensor configuration that they want into their Android-powered devices
  - devices can include a variety of sensors in a wide range of configurations
- There are two options for ensuring that a given sensor is present on a device:
  - Detect sensors at runtime and enable or disable application features as appropriate
  - Use Google Play filters to target devices with specific sensor configurations

# Android – Sensors

- Useful notes for creating apps that use sensors:
  - Always verify that a sensor exists on a device before you attempt to acquire data from it
  - Be sure to unregister a sensor's listener when you are done using the sensor or when the sensor activity pauses
    - If a sensor listener is registered and its activity is paused, the sensor will continue to acquire data and use battery resources unless you unregister the sensor
  - Be sure to choose a delivery rate that is suitable for the application or use-case
    - Sensors can provide data at very high rates, so sending extra data wastes system resources and uses battery power
  - Sensor data can change at a high rate, which means the system may call the `onSensorChanged(SensorEvent)` method quite often - do as little as possible **within the `onSensorChanged(SensorEvent)` method (don't block it!)**

## iOS – Location & Maps

- For a while Apple relied on a competitor (Google) for mapping data in iOS; This dependency officially ended with iOS 6 through the introduction of Apple Maps
- In iOS 8, Apple Maps officially replaced the Google-based map data with data provided primarily by a company named TomTom (but also including technology from other companies, including some acquired by Apple for this purpose)
- Apps use location data for many purposes, ranging from social networking to turn-by-turn navigation services

## iOS – Location & Maps

- Location-based information (in the app) consists of two pieces:
  - location services, and
  - maps
- Location services are provided by the `Core Location` framework, and obtain information about the user's location and heading (the direction in which a device is pointing)
- Maps are provided by the `Map Kit` framework, which supports both the display and annotation of maps

## iOS – Getting the User's Location

- Core Location framework provides several services that can be used to get and monitor the device's current location
  - uses all available onboard hardware, including Wi-Fi, GPS, Bluetooth, magnetometer, barometer, and cellular hardware to gather data
  - provides services for determining a device's geographic location, altitude, orientation, or position relative to a nearby iBeacon
  - standard location service - offers a highly configurable way to get the current location and track changes (for the specified level of accuracy)
  - significant-change location service - provides a way to get the current location and be notified when significant changes occur - delivers updates only when there has been a significant change (500 meters or more) in the device's location
  - region monitoring - lets monitoring boundary crossings for defined geographical regions and Bluetooth low-energy beacon regions

## iOS – Getting the User’s Location

- For iOS app that requires location services
  - include the `UIRequiredDeviceCapabilities` key in the app’s `Info.plist` identifying the required features with strings :
    - `location-services` – require location services in general
    - `gps` – requires the accuracy offered only by GPS hardware

    *App Store will not allow downloading the app if the device doesn’t contain the listed features*
- The app that uses `Core Location` must requests the user’s permission for location services by:
  - including the *Privacy-Location When In Use Description* (`NSLocationWhenInUseUsageDescription`) or *Privacy-Location Always and When In Use Usage Description* (`NSLocationAlwaysUsageDescription`) key in its `Info.plist` file (using a string that describes how the app intends to use location data)

# iOS – Getting the User's Location

- To use the location service, an instance of the `CLLocationManager` class should be created
- Using the location manager object:
  - set the delegate for the location manager
  - check if the location services are available (use `locationServicesEnabled()` method)
  - `requestWhenInUseAuthorization()` or `requestAlwaysAuthorization()` method should be called before the app can receive location-related information
  - can be configured the `desiredAccuracy` and `distanceFilter`
  - start receiving location notifications by calling the `startUpdatingLocation()` or `startMonitoringSignificantLocationChanges()` methods
  - stop the delivery of location updates by calling the `stopUpdatingLocation()` or `stopMonitoringSignificantLocationChanges()` methods
- When the location data become available, the location manager reports events to/call the `locationManager(_:didUpdateLocations:)` method of its delegate

## iOS – Getting the Heading and Course of a Device

- Core Location supports two different ways to get direction-related information:
  - Devices with a magnetometer can report the direction in which a device is pointing, also known as its heading
  - Devices with GPS hardware can report the direction in which a device is moving, also known as its course
- Heading and course information don’t represent the same information
  - The heading of a device reflects the actual orientation of the device relative to true north or magnetic north
  - The course of the device represents the direction of travel and doesn’t take into account the device orientation

# iOS – Geocoding Location Data

- To initiate a forward or reverse-geocoding request the `CLGeocoder` class can be used
- A *geocoder object* uses a network service to convert between *latitude* and *longitude* values and a *placemark* (collection of data such as the street, city, state, and country information)
- Reverse geocoding is supported in all versions of iOS, but forward geocoding is supported only in iOS 5.0 and later
- Geocoders rely on a network service - a live network connection must be present in order for a geocoding request to succeed
- If a device is in Airplane mode or the network is currently not configured - the geocoder can't connect to the service it needs, so it will return an appropriate error

## iOS – Geocoding Location Data

- For forward geocoding, the geocoder object parses the given information - if it finds a match returns a number of placemark objects
  - The more information provided to the forward geocoder, the better the results returned
  - The number of returned placemark objects depends greatly on the specificity of the provided information
- The geocoder object initiates the reverse geocoding request asynchronously and delivers the results to the provided (completition handler) block
  - The block is executed regardless of whether the request succeeds or fails
- The same `CLGeocoder` object can be used to initiate any number of geocoding requests but only one request at a time may be active for a given geocoder

# iOS – Displaying Maps

- The Map Kit framework allows embedding a fully functional map interface into the app
- A map can be displayed as:
  - Standard
  - Satellite
  - Hybrid

- Map Kit supports three basic coordinate systems for specifying map data points:
  - A map coordinate - a latitude and longitude on the spherical representation of the Earth (`CLLocationCoordinate2D` structure, or specifing areas using the `MKCoordinateSpan` and `MKCoordinateRegion` structures)
  - A map point - an x and y value on the Mercator map projection (`MKMapPoint` structure, or specify areas using the `MKMapSize` and `MKMapRect` structures)
  - A point - a graphical unit associated with the coordinate system of a view object (`CGPoint` structure, or specify areas using the `CGSize` and `CGRect` structures)

# iOS – Displaying Maps

- The `MKMapView` class is a self-contained interface for presenting map data in the app
  - It provides support for displaying map data, managing user interactions, and hosting custom content provided by your app
- A map view is a view - it can be manipulated the same way as other views
  - change its size and position, configure its autoresizing behaviors, add subviews
- A delegate object should be assigned to the map view
  - The map view reports all relevant interactions to its delegate, so the delegate can respond appropriately
- The `MKMapView` class has several properties that can be configured programmatically
  - which part of the map is currently visible, whether the content is displayed in 3D, what user interactions are allowed
- The *region* property of the `MKMapView` class controls the currently visible portion of the map
- Displaying the user's current location on the map can be done by setting the `showsUserLocation` property of the map view object
  - it causes the map view to use Core Location to find the user's location and add an annotation to the map

## iOS – Annotating Maps

- Annotations display content that can be defined by a single coordinate point
  - The annotations can represent information such as the user’s current location, a specific address, or a single point of interest
- Overlays display content that is defined by any number of points and may constitute one or more contiguous or noncontiguous shapes
  - The overlays can present more complex information such as routes or traffic information, or the boundaries of areas such as parks, lakes, cities, states, or countries
- Map Kit separates the data associated with an annotation or overlay from its visual presentation on the map
  - This separation allows the map to manage visible annotations and overlays much more efficiently

## iOS – Annotating Maps

- Annotation objects are typically small data objects that store the map coordinate data and any other relevant information about the annotation, such as a title string
  - The map view keeps a reference to the annotation objects you add to it and uses the data in those objects to determine when to display the corresponding view
- To display an annotation on a map, two distinct objects must be provided:
  - An annotation object, which is an object that conforms to the `MKAnnotation` protocol and manages the data for the annotation
  - An annotation view, which is a view (derived from the `MKAnnotationView` class) used to draw the visual representation of the annotation on the map surface
- Map Kit provides some standard annotation objects – `MKPointAnnotation` class can be used displaying a title string at the specified point on the map

*The annotations you create are typically anchored to a single map coordinate that doesn't change, but you can modify an annotation's coordinate programmatically if needed. In addition, you can implement support to allow users to drag annotations around the map.*

## iOS – Annotating Maps

- Overlays offer a way to layer content over an arbitrary region of the map
- Overlays are typically defined by multiple coordinates
  - You can use these coordinates to create contiguous or noncontiguous sets of lines, rectangles, circles, and other shapes, which can then be filled or stroked with colour
- To display an overlay on a map, the app must provide two distinct objects:
  - An overlay object, which is an object that conforms to the `MKOverlay` protocol and manages the data points for the overlay
  - An overlay renderer (derived from `MKOverlayRenderer`) used to draw the visual representation of the overlay on the map surface

## iOS – Annotating Maps

- Overlay objects are generally small data objects that store the points that define the overlay and any other relevant information, such as a title string
- Map Kit defines several concrete overlay objects (for specifying different types of standard shapes) and standard overlay renderers (that can draw any shapes represented by the concrete overlay objects)
- The standard overlay classes include MKCircle, MKPolygon, and MKPolyline

# iOS – Providing Directions

- Map Kit includes the `MKDirections` API, which provides route-based directions data from Apple servers
  - `MKDirections` API provide step-by-step walking or driving directions, travel time estimates, alternate routes, and other information

- Using Map Kit support for map-based directions, an app can:
  - Ask the Maps app to display directions to the user
  - Register as a routing app and provide point-to-point, turn-based directions to Maps and other apps on the user's device
  - Get general-purpose directions information

*An app that is able to display point-to-point directions can register as a routing app and make those directions available to the Maps app and to all other apps on a user's device.*

# iOS – Enabling Search

- Map Kit provides the `MKLocalSearch` API to support searches based on user-entered queries
  - Apps can use this API to perform searches for locations that users describe by name, address, or type, such as coffee or theatre
- Although local search and geocoding are similar, they support different use cases:
  - Use geocoding when you want to convert between map coordinates and a structured address, such as an Address Book address
  - Use local search when you want to find a set of locations that match the user’s input

## iOS – Sensors

- A device need to contain hardware for sensing the world around itself — where it is located, how it is oriented, how it is moving
- `Core Location` framework provides information about the device’s current location and how that location is changing overtime, along with information about the device’s orientation relative to north
- `UIEvent` class (for device shake) and the `Core Motion` framework provide information about the device’s change in speed and attitude
- The device may have an extra chip that analyzes and records the user’s activity, such as walking or running - the `Core Motion` framework provides access to this information

## iOS – Sensors

- One of the major challenges associated with applications that uses sensor data is that different devices have different hardware (type of sensors)
- The app should provide a subset of its full capabilities when it discovers that the current device lacks certain features
  - certain sensors may experience momentary inadequacy, or
  - some sensors take time to “warm up,” so that the values you’ll get from them initially will be invalid
- All sensor usage means battery usage (sometimes to a considerably greater degree)

## iOS – Sensors

- The `Core Location` framework provides facilities for the device to determine and report its location and device orientation using sensors:
  - Wi-Fi - may scan for nearby Wi-Fi devices and compare these against an online database
  - Cell - may compare nearby telephone cell towers against an online database
  - GPS - may be able to obtain a position fix (information) from GPS satellites – it is the most accurate location sensor, but it takes the longest to get a fix, and in some situations it can fail
  - Magnetometer – provide information about the device's orientation relative to north
- Core Location will automatically use whatever facilities the device has available
  - It allows to specify how accurate a position fix needs to be - more accurate fixes may require more time
- The notion of a location is encapsulated by the `CLLocation` class with properties: coordinate (latitude and longitude), altitude, speed, course, horizontalAccuracy and timestamp

# iOS – Sensors

- `Core Motion framework` reports motion and environment-related data from the onboard hardware (including accelerometers, gyroscopes, pedometer, magnetometer and barometer)
- This framework allows access to hardware-generated data that can be used in the app
  - For example, a game might use accelerometer and gyroscope data to control onscreen game behaviour
- Many services of this framework allow access to the raw values recorded by the hardware and a processed version of those values
  - Processed values do not include forms of bias that might adversely affect how you use that data
  - For example, a processed accelerometer value reflects only the acceleration caused by the user and not the acceleration caused by gravity

# iOS – Sensors

- Acceleration is detected through the device's accelerometer, supplemented by the gyroscope (if there is one)
  - The accelerometer always has something to measure (because of the gravity) - thus the device can report its attitude relative to the vertical
- Acceleration information can arrive in two ways:
  - As a prepackaged `UIEvent` – as notification of a predefined gesture performed by accelerating the device
  - using the `Core Motion` framework
    - an instance of `CMMotionManager` allows obtaining accelerometer information, gyroscope information, or device motion information (`Core Motion` can be also used to get magnetometer information).

## iOS – Sensors

- The inclusion of an electronic gyroscope in the hardware of some devices makes a huge difference in the accuracy and speed of gravity and attitude reporting
- A gyroscope has the property that its attitude in space remains constant; thus it can detect any change in the attitude of the containing device
- The advantages include:
  - The accelerometer can be supplemented by the gyroscope to detect quickly the difference between gravity and user-induced acceleration
  - The gyroscope can observe pure rotation, where little or no acceleration is involved (case where the accelerometer is helpful – cannot detect)
  - The extreme case is constant attitudinal rotation around the gravity axis, which the accelerometer alone would be completely unable to detect

# iOS – Sensors

- `Core Motion` package up the calculated combination of accelerometer and gyroscope data in a `CMDeviceMotion` instance
- Class `CMDeviceMotion` properties include: gravity, userAcceleration, rotationRate, magneticField, attitude and heading
- Some devices have a motion coprocessor chip that can detect, analyze, and keep a record of device motion (even while the device is asleep and with very little battery power)
    - It is an analysis of the device's physical motion and attitude in order to draw conclusions about what the user has been doing while carrying or wearing the device
    - This is not a form of location determination – the data recorded that the user is walking, or walked for an hour, but not where the user was walking
- Interaction with the motion coprocessor is through an instance of a `CMMotionActivityManager` class

## iOS – GameKit

- GameKit - Create experiences for players by adding features of leaderboards, achievements, matchmaking, challenges, and more
- It provides the ability to create apps that allow players to interact with each other
- Game Kit provides different functionality as:
  - a centralized game service that connects players to each other
  - leaderboards that promote competition and comparison
  - real-time and turn-based multiplayer experiences

# iOS – GameplayKit

- GameplayKit  framework provides foundational tools and technologies for building games
- GameplayKit - architect and organize game app logic
  - design modular, scalable game architecture
  - incorporate common gameplay behaviours - random number generation, artificial intelligence, pathfinding, and agent behaviour
- GameplayKit provides technologies for building and enhancing gameplay features (such as character movement, AI and opponent behaviour)

## iOS – HealthKit

- Creating a complete, personalized health and fitness experience includes:
  - Collecting and storing health and fitness data
  - Analysing and visualizing the data
  - Enabling social interactions

- HealthKit provides a central repository for health and fitness data on iPhone and Apple Watch
  - It can be used to share data between apps

- HealthKit stores variety of data (in encrypted format): characteristic data, sample data, workout data, source data
- With the user's permission, apps communicate with the HealthKit store to access and share this data

# iOS – HealthKit

- To access HealthKit store data, the `HKHealthStore` class can be used
  - Both iPhone and Apple Watch have their own HealthKit store
- Health data is automatically synced between Apple Watch and iPhone
  - old data is periodically removed from Apple Watch to save space

Source: https://developer.apple.com/documentation/healthkit

## iOS – Core Bluetooth

- The Core Bluetooth framework provides the classes needed for iOS apps to communicate with Bluetooth Low Energy (equipped) devices
  - The app can discover, explore, and interact with low energy peripheral devices, or act as Bluetooth low energy peripheral (iOS6+)
- In Bluetooth low energy communication, there are two roles:
  - the central - scans for nearby peripherals, and uses the information served up by a peripheral
  - the peripheral - has data (or services/features) that is needed by other devices and advertise it over the air
- Apple’s new iOS 13 update adds a new privacy measure that requires apps to get user’s consent in order to use the device’s Bluetooth
  - `Info.plist` needs to include usage description keys for `NSBluetoothAlwaysUsageDescription` (iOS13) or `NSBluetoothPeripheralUsageDescription` (iOS12 and previous versions)

# iOS – Core Bluetooth

- Core Bluetooth includes classes and protocols that can be used for developing apps that implement:
  - Central role - scanning, discovering and connecting with a peripheral, as well as exploring and interacting with the peripheral's data
  - Peripheral role – advertising its data or primary functionality and responding to the central in appropriate ways
- Classes: `CBCentral, CBCentralManager, CBPeripheral, CBPeripheralManager`
- Protocols: `CBCentralManagerDelegate, CBPeripheralDelegate, CBPeripheralManagerDelegate`

*In iOS 13 the Core Bluetooth framework provides supports for Basic Rate / Enhanced Data Rate (BR/EDR) wireless technology.* *https://developer.apple.com/videos/play/wwdc2019/901*

## Android – Connectivity

- Android provides rich APIs that allow applications (in addition to standard network connections) to connect and interact with other devices over
  - Bluetooth
  - NFC
  - Wi-Fi P2P
  - USB
  - SIP (Session Initiation Protocol)

# Android – Bluetooth

- The application framework provides access to the Bluetooth functionality through the Android Bluetooth APIs
  - These APIs allow wireless connection with other Bluetooth devices, enabling point-to-point and multipoint wireless features
- Using Bluetooth APIs an Android app can:
  - Scan for other Bluetooth devices
  - Query the local Bluetooth adapter for paired Bluetooth devices
  - Establish RFCOMM channels
  - Connect to other devices through service discovery
  - Transfer data to and from other devices
  - Manage multiple connections

## Android – Bluetooth

- For Android platform, there are TWO options for Bluetooth connectivity:
  - Classic Bluetooth – more battery-intensive operations
  - Bluetooth Low Energy – Android 4.3 (API Level 18)
- In order for Bluetooth-enabled devices to transmit data between each other, they must:
  - Set up Bluetooth
  - Find devices to be paired (device using service discovery process finds discoverable device)
  - Connect the devices (bonding process – exchange of security keys)
  - Transfer data

# Android – Bluetooth

- In order to use Bluetooth features in the application, several permissions are required

```
<!-- Request legacy Bluetooth permissions on older devices. -->
   <uses-permission android:name="android.permission.BLUETOOTH“  android:maxSdkVersion="30" />
   <uses-permission android:name="android.permission.BLUETOOTH_ADMIN“  android:maxSdkVersion="30" />

Android 12+
   <!-- Needed only if your app looks for Bluetooth devices. -->
   <uses-permission android:name="android.permission.BLUETOOTH_SCAN" />

   <!-- Needed only if your app makes the device discoverable to Bluetooth devices. -->
   <uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />

   <!-- Needed only if your app communicates with already-paired Bluetooth devices. -->
   <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />

   <!-- Needed only if your app uses Bluetooth scan results to derive physical location. -->
   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

https://developer.android.com/guide/topics/connectivity/bluetooth/permissions

# Android – Bluetooth

- The `android.bluetooth` package provide classes and interfaces needed for creating Bluetooth connections

- For example
  - `BluetoothAdapter` - Represents the local Bluetooth adapter (the entry-point for all Bluetooth interaction)
  - `BluetoothDevice` - Represents a remote Bluetooth device

  - Many others

## Android – Near Field Communication

- Near Field Communication (NFC) is a set of short-range wireless technologies, typically requiring a distance of 4cm or less to initiate a connection
- NFC allows sharing of small payloads data between an NFC tag and an Android-powered device, or between two Android-powered devices
- Tags can range in complexity:
  - Simple tags offer just read and write, or one-time-programmable areas to make the card read-only
  - More complex tags offer math operations and have cryptographic hardware to authenticate access to a sector
  - Most sophisticated tags contain operating environments, allowing complex interaction with code executing on the tag

# Android – Near Field Communication

- Android-powered devices with NFC simultaneously support three main modes of operation:
  - Reader/writer mode, allowing the NFC device to read and/or write passive NFC tags and stickers
  - P2P mode, allowing the NFC device to exchange data with other NFC peers (used by Android Beam)
  - Card emulation mode, allowing the NFC device itself to act as an NFC card that can be accessed by an external NFC reader

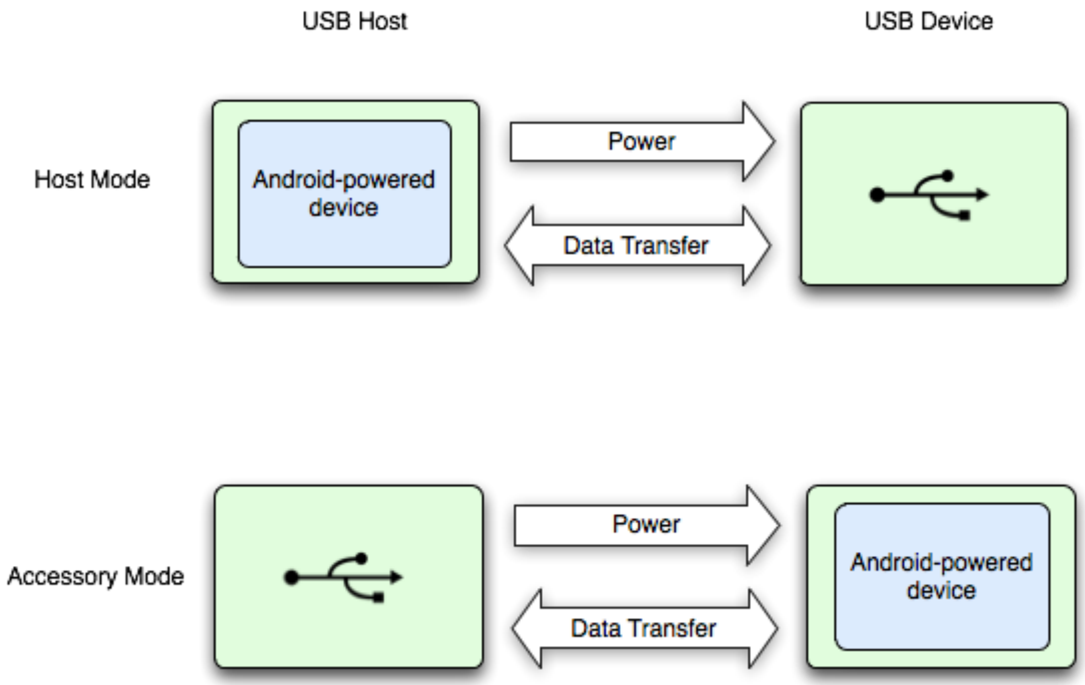
*NFC functionalities can be implemented by using the classes and interfaces of android.nfc package, or Core NFC framework for iOS applications.*

# Android – Wi-Fi peer-to-peer

- Wi-Fi peer-to-peer (P2P) allows devices to connect directly to each other via Wi-Fi (without an intermediate access point)
- Wi-Fi P2P APIs, allow apps to discover and connect to other devices and communicate over a speedy connection across distances much longer than a Bluetooth connection
  - This is useful for applications that share data among users
- The Wi-Fi P2P APIs consist of :
  - Methods (in `WifiP2pManager` class) that allow to discover, request, and connect to peers
  - Listeners that allow notification of the success or failure of `WifiP2pManager` method calls (each method can receive a specific listener passed in as a parameter)
  - Intents that notify of specific events detected by the Wi-Fi P2P framework, such as a dropped connection or a newly discovered peer

## Android – USB host and accessory

- Android supports a variety of USB peripherals and Android USB accessories (hardware that implements the Android accessory protocol)
- Two modes:
  - USB accessory
  - USB host



source: https://developer.android.com/guide/topics/connectivity/usb/

# Android – USB host and accessory

- USB accessory
  - The external USB hardware act as the USB hosts - gives Android-powered devices (that do not have host capabilities) the ability to interact with USB hardware
  - Android USB accessories must be designed to work with Android-powered devices and must adhere to the Android accessory communication protocol
- USB host
  - The Android-powered device acts as the host
  - USB devices that are designed for a wide range of applications and environments can still interact with Android applications that can correctly communicate with the device

# Networking

- The world of networking is complex - users can connect to the Internet using a wide range of technologies (Wi-Fi, cellular connections, satellite uplinks)
  - Each of these connections has distinct characteristics, including differences in bandwidth, latency, packet loss, and reliability
  - Networks are inherently unreliable - as a result, good networking code is complex
- Some note for developing reliable application:
  - Transfer only as much data as required to accomplish a task
  - Avoid timeouts whenever possible - design user interfaces that allow the user to easily cancel transactions that are taking too long to complete
  - Handle failures and slow network performance - when failures occur, the app should continue to function to the maximum degree possible in an offline state
  - Choose APIs that are appropriate for the task
  - Design the app carefully to minimize security risks

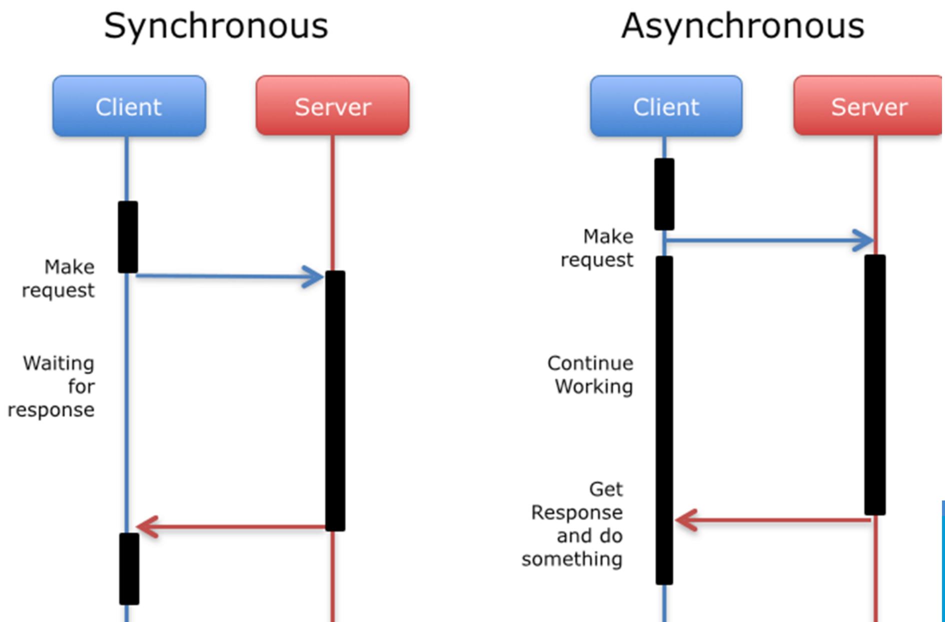
## Networking

- A device’s network environment can change at a moment’s notice
- There are a number of simple networking mistakes that can adversely affect app’s performance and usability
- The app
  - Must execute networking code asynchronously instead of  executing it in app’s main thread
  - Must handle network changes dynamically

# Networking

- In the context of Android or iOS , networking is mostly about web services
- Web services are designed to communicate with other programs (the Android or iOS app) over the HTTP ("web") protocol
- In general, there are:
  - XML/SOAP web services are more formal and thus have significantly more overhead, both at development time and at runtime, but offer more capabilities
  - RESTful services are more lighterweight,

    and are not tied to XML



## iOS – Networking

- Network Framework (introduced in iOS12)
  - gives direct access to the same high-performance user-space networking stack
  - used for create network connections to send and receive data using transport and security protocols

- supports:
  - connection establishment
  - optimized data transfer
  - built-in security
  - seamless mobility
  - native Swift support

# iOS – Networking

- The `URLSession` class and related classes provide an API for downloading data from and uploading data to endpoints indicated by URLs
  - Rich set of delegate methods for authentication and notification of events
  - Gives the app ability to perform background downloads when your app isn't running or is suspended
  - Provides status and progress properties
  - Supports cancelling, restarting, resuming and suspending

- The `URLSession` class supports
  - the data, file, ftp, http, and https URL schemes, transparent support for proxy servers and SOCKS gateways
  - the HTTP/1.1, HTTP/2, HTTP/3 protocols

# iOS – Networking

- An instance of the `URLSession` class can coordinate a group of related network data transfer tasks
  - Logic to be split into Session and Task
  - A session to have multiple tasks
  - Specific configuration on a session for all tasks
- App Transport Security (ATS) (introduced in iOS 9.0) is a security feature enabled by default for all HTTP connections made with `URLSession`
  - ATS requires that HTTP connections use HTTPS (RFC 2818)

# Android - Networking

- In order to perform network operations in the application, the manifest must include the permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

- To run a network task in an Android app can be used:
  - `ConnectivityManager` class
    - Class that answers queries about the state of network connectivity; it also notifies applications when network connectivity changes
  - `java.util.concurrent`
    - Utility classes commonly useful in concurrent programming
  - `URLConnection` or `HttpURLConnection` classes
    - Abstract classes and subclasses that represent a communications link between the application and a URL

## Android - Networking

- `ConnectivityManager` class answers queries about the state of network connectivity
- It also notifies applications when network connectivity changes
- The primary responsibilities of this class are to:
  - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.)
  - Send broadcast intents when network connectivity changes
  - Attempt to "fail over" to another network when connectivity to a network is lost
  - Provide an API that allows applications to query the coarse-grained or fine-grained state of the available networks
  - Provide an API that allows applications to request and select networks for their data traffic

# Android - Networking

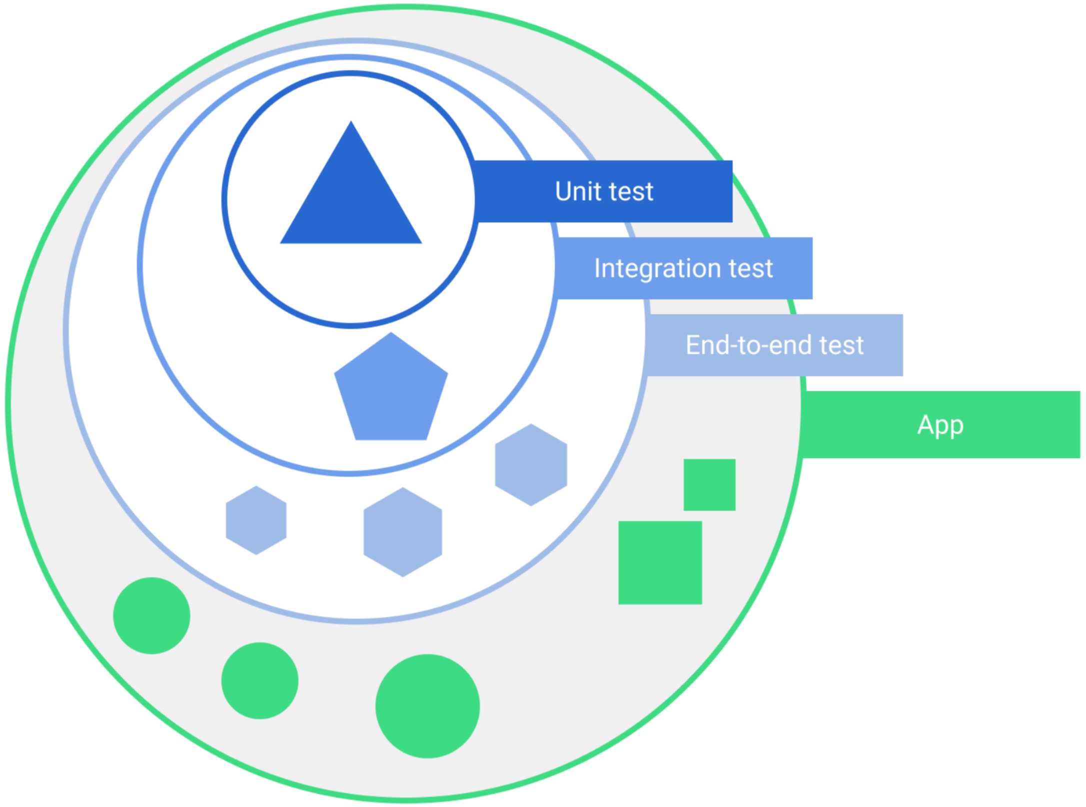
- The abstract class `URLConnection` is the superclass of all classes that represent a communications link between the application and a URL
  - Instances of this class can be used both to read from and to write to the resource referenced by the URL
- A `HttpURLConnection` class extends `URLConnection` with support for HTTP-specific features

# App Testing

- Testing your app is an integral part of the app development process
- By running tests against your app consistently, you can verify your app's correctness, functional behavior, and usability before you release it publicly
- Advantages of testing:
  - Rapid feedback on failures
  - Early failure detection
  - Safer code refactoring – code optimization
  - Stable development velocity

# App Testing

- Different types of tests
- based on the subject: Functional, Performance, Accessibility, Compatibility testing
- based on the size, or degree of isolation: Unit (small tests), End-to-end (big tests), Medium tests



https://developer.android.com/training/testing/fundamentals

## App Testing

- Testing frameworks help to optimize the entire testing process and ensure quality standards are met.
  - Appium (Android/iOS)
  - Detox (Android/iOS)
  - Calabash (Android/iOS)
  - UI Automator (part of Android SDK)
  - Espresso (Android)
  - Robotium (Android)
  - EarlGray (iOS)
  - XCTest/XCUITest (iOS)
  - OCMock (iOS)

# Android – Publishing the app

- Publishing is the general process that makes your Android applications available to users
- Publishing an Android application include several tasks
- Preparing the application for release
  - Configure the application for release (check the manifest file, configure settings to meet Google Play requirements)
  - Build and sign (with encrypted signing certificate) a release version of your application
  - Test the release version of your application
  - Update application resources for release
  - Prepare remote servers and services that your application depends on
- Releasing the application to users
  - releasing through an app marketplace such as Google Play includes
    - Preparing promotional materials
    - Configuring options and uploading assets
    - Publishing the release version of your application

# iOS – Publishing the app

- The final step for your iOS app is to submit it to Apple’s App Store
- Preparing and submitting an application is a multistep process:
  - Create a developer account with Apple and sign up for the Developer Program
    - in Xcode set up the Signing (with the provisional account) and Team information
  - Create App Store listing (via App Store Connect)
  - Make screenshots of the app and upload it to the App Store listing
  - Include all required app’s information in the App Store listing
  - Upload the app to App Store connect using Xcode
    - select the Generic iOS Device from the list of simulators
    - go to Product >> Archive and let it bundle the app
    - select the archive and choose Distribute App (select the method of distribution - iOS App Store)
  - Submit the app for review
    - In App Store Connect click “Submit for Review”

*If your app is rejected you will be provided with the reasons for rejections and what to fix.*

## Resources

- Android Studio Dolphin Essentials - Java Edition, Neil Smyth, Payload Media Inc., 2022.
- Programming iOS 14 - Dive Deep into Views, View Controllers, and Frameworks, Matt Neuburg, O'Reilly Media Inc., 2021
- iOS 12 App Development Essentials, Neil Smyth, Payload Media, Inc., 2018.
- https://developer.android.com/guide/topics/permissions/overview#normal
- https://developer.android.com/training/location/index.html
- https://developer.android.com/guide/topics/sensors/sensors_overview
- https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/LocationAwarenessPG/Introduction/Introduction.html

- If you need to setup Google Play services and add the library to your project
  - https://developers.google.com/android/guides/setup
  - https://developers.google.com/maps/documentation/android-api/start

# Resources

- Android Cookbook, Ian Darwin, O'Reilly Press, 2017.
- iOS 12 App Development Essentials, Neil Smyth, Payload Media, Inc., 2018.
- Programming iOS 14 - Dive Deep into Views, View Controllers, and Frameworks, Matt Neuburg, O'Reilly Media Inc., 2021.
- https://developer.apple.com/documentation/gameplaykit
- https://developer.apple.com/documentation/gamekit
- https://developer.apple.com/documentation/healthkit
- https://developer.apple.com/library/archive/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/AboutCoreBluetooth/Introduction.html#//apple_ref/doc/uid/TP40013257-CH1-SW1
- https://developer.android.com/guide/topics/connectivity/
- https://developer.apple.com/documentation/network
- https://developer.apple.com/documentation/foundation/urlsession
- https://developer.android.com/training/testing/fundamentals
- https://help.apple.com/xcode/mac/current/#/dev067853c94