1: Sensor Framework

(A-a)

```
SensorManager sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

(A-b)

```
Sensor sensor = ...;
float maxRange = sensor.getMaximumRange();
```

(A-c)

```
Sensor accSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
registerListener (listener, accSensor, SENSOR_DELAY_FASTEST, handler)
```

Where sensorManager, accSensor and handler have been defined appropriately.

(B)

(Note: I am still unsure about this. The reusing is only a problem if the MainActivity also doesn't copy the accelerometer data. What do you think?)

The SensorEvent objects passed to the SensorEventListener can be resused by the system. This means that the values array may be overwritten by the system to hold data for some other event. It is thus necessary to copy the values in the onSensorChanged method, and pass that copy to the listenerActivity.

As a sidenote, it is unadvisable to block in the onSensorChanged method. The main activity should thus not directly update its display, but rather store the new values invalidate the views.

2: Activity Lifecycle

a) An activity A is in the foreground, then the user stars another activity B.

- Activity B will be created and the method OnCreate() will be called.
- Activity B will get started and the method OnStart() gets called.
- Activity A will be paused and the OnPause() method of the class Activity will be called.
- Activity A is only partially visible.
- Activity B will be resumed and the method OnResume() will be called.

-Activity B is created with onCreate()

-Activity B is started with onStart()

-Activity A is stopped with onPause()

-Activity B is resumed with onResume()

If A is not partially visible :

-Activity A is stopped with onStop()
b) Activity A is no longer visible.

- Now Activity A is no longer visible.

- Activity A will get stopped and OnStop() will be called.

c) User navigates back to Activity A.

- Since Activity A got stopped before, it first has to be restarted and the method OnRestart() will be called.
- Now the method OnStart() will get called.
- Since A is about to get resumed, Activity B will get paused and OnPause() will get called.
- Activity B is only partially visible now.
- After that Activity A gets resumed and OnResume() will get called.
- Since A got resumed and is covering B, Activity B gets stopped and OnStop() will be called.

Note: It can happen that the System needs the memory used by a paused or stopped Activity elsewhere.
The System can then temporarily destroy/kill the Activity and OnDestroy() will be called.
To display the Activity again it has to completely recreated, started and restored to its previous state.
In that case : OnCreate(), OnStart() and OnResume() will be called.

Three cases:

The app process has been previously killed. So:

-Activity A is created with onCreate()

-Activity A is started with onStart()

-Activity A is resumed with onResume()

The activity A was stopped. (was not longer visible at some point)

-Activity A is restarted with onRestart()

-Activity A is started with onStart()

-Activity A is resumed with onResume()

The activity A was only paused. (still partially visible)

-Activity A is resumed with onResume()

3: Resources

The Android XML Layouts can be defined so that interface elements are layed out relative to each other and the screen border. This is good for small variations in screen size. For vastly different screen sizes and densities different layouts can be defined. The corresponding activity can then instanciate the most appropriate one in its onCreate.

Another way to do it is to instaciate all Views programmatically and make this layout and drawing code work in consideration of the available screen realestate. But often doing things all

programmatically is more tedious.

Android XML elements can have an id. An element with id x can be referenced in code by $R.id.x$.

4: Intents

What are Intents?

An Intent is a messaging object and an abstract description of an operation to be performed.

What are they used for?

You can use an Intent to request an action from another app component and carrying any necessary data to it. There are basic cases:
1. To start an activity

It will start the window which is an instance of the Activity. The Activity can then use the data that got carried over and perform its own methods. One can use startActivityForResult() to receive a result when the Activity finishes.

2. To start a service

This will start a service which will perform its operations with help of the data carried over. Compared to the activity it will perform in the background without an user interface.

3. To deliver a broadcast

This will basically send a message which any app can receive.

What is the difference between Explicit Intents and Implicit Intents?

Explicit Intents:
- For an Explicit Intent you have to specify the component to start by a fully-qualified class name.

- Typically you use an Explicit Intent to start a component in your own app.

- For 1. and 2. from the question before the system will immediately start the app component communicated in the Intent.

Implicit Intents:
- Compared to the Explicit Intents - An Implicit Intent will will not name a specific app component like an action or service.

- Instead it will declare a general action which is allowed to be handled by an component of a completely different app.

- To find out which components are the appropriate ones the system will check the Intent filters which should be declared in the manifest file. If it has a match then the system starts the match. If there are more than one matches the user can pick from the list. And if none are found the call will fail and the app will crash if one did not verify before that there is at least one match.
Note: The intent-Filter does not influence any explicit Intents

5: Service Lifecycle

(a)

wrong, a service can be stopped with the method stopService().

(b)

wrong, we call "bound service" a service that have at least one activity is bonded to it. Thus, an unbound service doesn't interact with client process.

From android.developper.com :

"A service is "bound" when an application component binds to it by calling bindService()."

(c)

wrong, if someone has called startService(), service is only destroy after a call of stopService() or stopSelf() and all clients have called onUnbind().

(d)

true, services are always called in a separate thread.

6: AndroidManifest file

(1)

<uses-permission android:name= "android.permission.WRITE_SMS"/>

(2)

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

(3)

<uses-permission android:name="android.permission.READ_PHONE_STATE" />