1. Define the following design principles: Singleton, Factory, Builder, Facade, Prototype

Singleton

Singleton Pattern belongs to creational Type pattern. This pattern is used when we need to ensure that only one object of a particular class needs to be created. All further References to the objects are referred to the same underlying instance created, It's used for logging, drivers objects, caching and thread pool

Factory

"Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface."

What this means is that Factory it's like the bridge who allow you to create objects without exposing the creation logic to the client.

Builder

Builder pattern builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. A Builder class builds the final object step by step. This builder is independent of other objects.

Facade

Facade pattern hides the complexity of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide It's complexities. This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes.

Prototype

Prototype pattern refers to creating duplicate object while keeping performance in mind. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. This pattern involves implementing a prototype interface which tells to create a clone of the current object. This pattern is used when creation of object directly is costly. For example, an object is to be created after a costly database operation. We can cache the object, return It's clone on next request and update the database as and when needed thus reducing database calls

2. What is the difference in ART and Dalvik?

Use the Ahead-Of-Time (AOT) approach, in which the dex files were compiled before they were demanded. This itself massively improves the performance and battery life of any Android device.	Use Just-In-Time (JIT) approach in which the compilation was done on demand
Whenever you install an app, all the dex files gets converted once and for all helping to save battery life	Repeat the process of conversion in their native codes every single time you open an app wasting CPU cycles and valuable battery juice.
The installation of apps take some time and bigger space because make all the conversion	The installation of apps take just the needed time and It'll make the conversion when the user decide to open it
The cache is built during the first boot, so the boot time is considerably more in this case	The cache is built with time the device runs and apps are used as is indicated by the JIT approach. So the boot time is very fast.
The apps with ART requires more space than the apps with Dalvik	The apps with Dalvik requires less space than the apps with ART

3. What is the android manifest used for?

If an Android app have something not declare in the AndroidManifest.xml but is ask for it, the app will crash.

The AndroidManifest.xml is used for:

- Activities
- Permissions
- Receivers
- Services
- The name of your application
- Version number
- App icon
- Theme used
- Properties of the activities called intent filter
- Launcher activity

4. How does each of the following units of measure for view work: sp, dp, px, pt, in, mm

sp

A scale independent pixel, specially designated for text sizes. This is a density independent unit, however the physical size of a single "sp" is only approximately the same on every screen density. Scaling factors, depending on the density bucket of the device, as well as the user's text size preference, are applied to convert "sp" to the number of pixels at 160 dpi. The number of pixels this translates to varies depending on screen density and the density bucket the device falls into.

dp

A density independent pixel. This is a density independent unit, however the physical size of a single "dp" is only approximately the same on every screen density. There are approximately 160 "dp" in an inch. A scaling factor, depending on the density bucket of the device, is applied to convert "dp" to the number of pixels at 160 dpi. The number of

pixels a single "dp" translates to varies depending on the pixel on screen density and the density bucket the device falls into.

px

An actual pixel on the screen. This is a density dependent unit, and the physical size of a single "px" varies depending on screen density.

Pt

A point, a common font size unit, on the screen. This is a density independent unit, and the physical size of a single "pt" is the same on every screen density. There are 72 "pt" in an inch. The number of pixels a single "pt" translates to varies depending on screen density.

In

A physical inch on the screen. This is a density independent unit, and the physical size of a single "in" is the same on every screen density. The number of pixels a single "in" translates to varies depending on screen density.

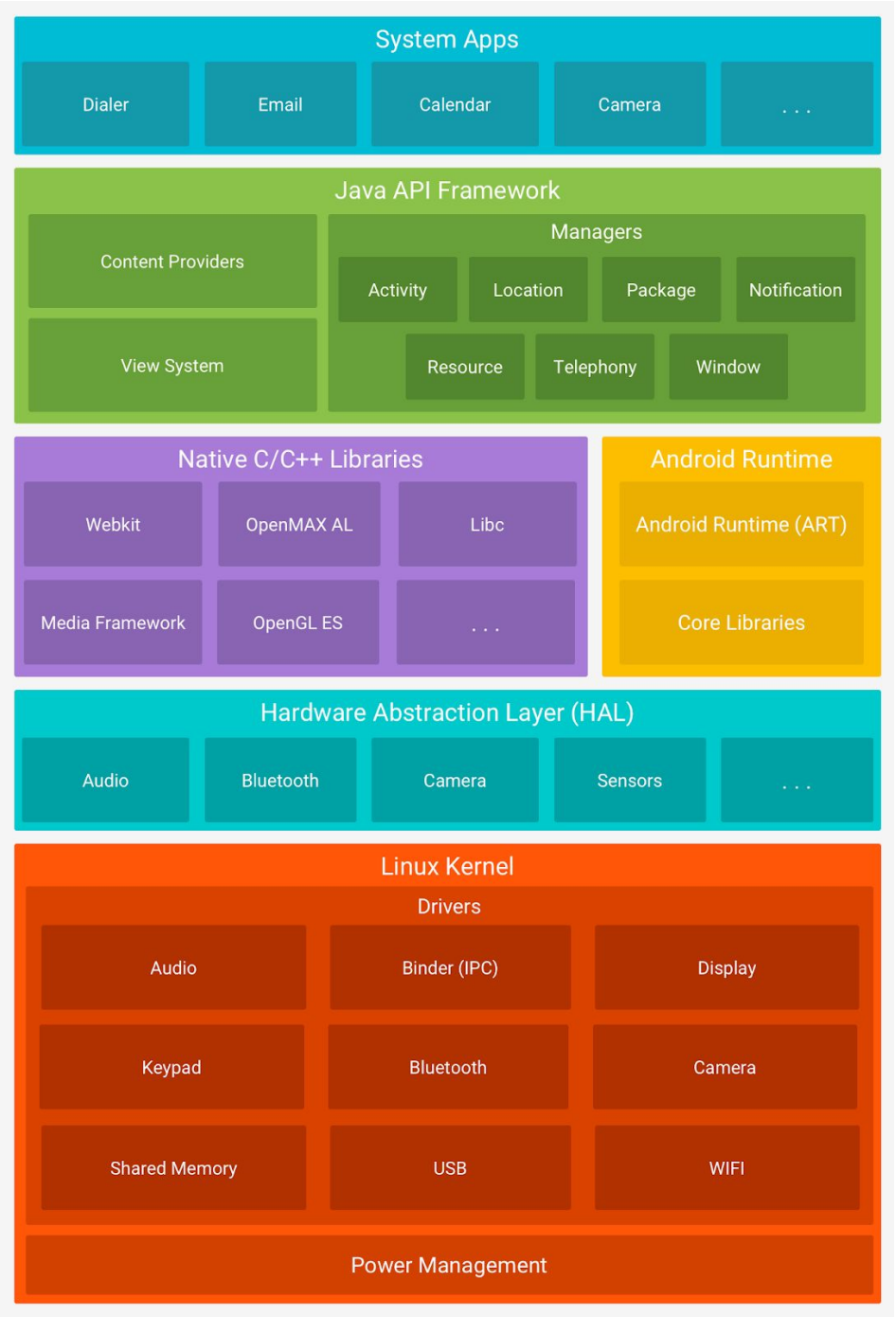
Mm

A physical millimeter on the screen. This is a density independent unit, and the physical size of a single "mm" is the same on every screen density. There are 25.4 "mm" in an inch. The number of pixels a single "mm" translates to varies depending on screen density.

Summary

Dimension	Description	Units / Physical Inch	Density Independent	Same Physical Size on Every Screen
px	Pixels	Varies	No	No
in	Inches	1	Yes	Yes
mm	Millimeters	25.4	Yes	Yes
pt	Points	72	Yes	Yes
dp	Density independent pixels	~160	Yes	No
sp	Scale independent pixels	~160	Yes	No

5. Describe what each section of the Android Platform arch. Details.



Linux Kernel

Android was created on the open source kernel of Linux. One main reason for choosing this kernel was that it provided proven core features on which to develop the Android operating system. The features of Linux kernel are:

Security

The Linux kernel handles the security between the application and the system.

Memory Management

It efficiently handles the memory management thereby providing the freedom to develop our apps.

Process Management

It manages the process well, allocates resources to processes whenever they need them.

Network Stack

It effectively handles the network communication

Driver Model

It ensures that the application works. Hardware manufacturers can build their drivers into the Linux build

Libraries:

Running on the top of the kernel, the Android framework was developed with various features. It consists of various C/C++ core libraries with numerous open source tools. Some of these are:

The Android runtime:

The Android runtime consists of core libraries of Java and ART(the Android RunTime). Older versions of Android (4.x and earlier) had Dalvik runtime.

Open GL(graphics library):

This cross-language, cross-platform application program interface (API) is used to produce 2D and 3D computer graphics.

WebKit:

This open source web browser engine provides all the functionality to display web content and to simplify page loading.

Media frameworks:

These libraries allow you to play and record audio and video.

Secure socket layer (SSL):

These libraries are there for Internet security.

Android Runtime:

It is the third section of the architecture. It provides one of the key components which is called Dalvik Virtual Machine. It acts like Java Virtual Machine which is designed specially for Android. Android uses it’s own custom VM designed to ensure that multiple instances run efficiently on a single device.The Dalvik VM uses the device’s underlying Linux kernel to handle low-level functionality,including security, threading and memory management.

Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

Activity Manager

It manages the activity lifecycle and the activity stack.

Telephony Manager

It provides access to telephony services as related subscriber information, such as phone numbers.

View System

It builds the user interface by handling the views and layouts.

Location Manager

It finds the device’s geographic location.

Applications:

Android applications can be found at the topmost layer. At application layer we write our application to be installed on this layer only. Examples of applications are Games, Messages, Contacts etc.

6. Define the following terms: View, ViewGroup, View Hierarchy.

View

A View occupies a rectangular area on the screen and is responsible for drawing and event handling.The View class is a superclass for all GUI components in Android.

Commonly used Views are :

- EditText
- ImageView
- TextView
- Button
- ImageButton
- CheckBox

ViewGroup

A ViewGroup is a special view that can contain other views (called children.) The view group is the base class for layouts and views containers.

Commonly used Views are :

- LinearLayout
- RelativeLayout
- FrameLayout
- ScrollView

View Hierarchy

It’s the component tree than we can see in the xml of the project in Android Studio

7. Explain in detail how the following layouts render and what unique items each has that must be implemented: Constraint, Linear, Coordinator, Grid and Relative?

ContraintLayout

It is a layout on Android that gives you adaptable and flexible ways to create views for your apps.

Sintax:

```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Activities.MainActivity">

</android.support.constraint.ConstraintLayout>
```

Unique:

The children of this ViewGroup need to declare

```
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"
```

LinearLayout

Android LinearLayout is a view group that aligns all children in either vertically or horizontally.

Sintax:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

</LinearLayout>
```

Unique:

This layout Need to declare the orientation, especially if this view group going to have more than one child

```
android:orientation="vertical"
```

CoordinatorLayout

The CoordinatorLayout is a new layout, introduced with the Android Design Support Library. The CoordinatorLayout is a super-powered FrameLayout (according to the official documentation).

Sintax:

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"

tools:context="com.sample.foo.usingcoordinatorlayout.Coordinator
Activity">
```

```
</android.support.design.widget.CoordinatorLayout>
```

Unique

```
app:popupTheme="@style/AppTheme.PopupOverlay"
app:layout_behavior="@string/appbar_scrolling_view_beh
avior"
app:layout_anchor="@id/image1"
app:layout_anchorGravity="bottom|right|end"
```