Research

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

**Singleton**: is a pattern that involves a single class responsible for the creation of an object and to ensure only a single object is created. This class allows its only object to be accessed directly without the need to instantiate the object itself. The class provides a static method so it can be accessed by other classes.

Example:

public class DogObject {

private static DogObject instance = new DogObject();

private DogObject(){}

public static DogObject getInstance(){

return instance;

}

public showName(){

System.out.println(“Alexander”);

}

}

In the main method, call:

DogObject dogObj = DogObject.getInstance();

dogObj.showName();

**Factory**: is one of the most common design used in Java. In Factory, object is created without having to specify the exact class of the object to be created. An example of this would be a Shape interface and Square and Triangle classes implementing the Shape interface. The class with the main method would pass the information “TRIANGLE” OR “SQUARE” to the ShapeFactory to get the actual object it needs.

**Builder**: is a pattern that uses simple objects to build a complex object using a step by step approach. This step by step approach helps the Builder class build the final object. This builder class is independent of other objects. An example of this would be the Meal options at Chick-fil-A restaurant. The items you build to form a Meal are built by the MealBuilder. If one wants a spicy chicken sandwich, large fries, and a lemonade for beverage, the MealBuilder will create a Meal class with an ArrayList of Items (food items and beverage) and the MealBuilder will build different type of Meal objects by combining them.

**Façade**: this pattern hides the implementation/complexities of the system, but provides the client with the interface. This pattern has a single class with its main goal to delegate calls to the methods as required. An example of this would be a Shape interface and classes such as Triangle, Circle, Square implement the Shape interface. Another class called the ShapeCaller uses the concrete classes to delegate calls to these classes.

**Prototype**: is a pattern referring to the creation of a duplicate object while keeping performance high. This pattern implements a prototype interface in order to request the creation of a clone of the current object. It is used when creating new objects directly is too costly. One can reduce calls to the database by caching the object, returning its clone and update the database as needed. For example, a Shape abstract class implementing the Clonable interface with other concrete classes such as Triangle, Circle and Square extending the Shape abstract class. The pattern uses the ShapeCache class to store the Shape objects and return its clone to get a Shape object.

**2. What are the differences in ART and Dalvik?**

Dalvik is the virtual machine process in Android OS. Its job is to execute applications written in Android. Applications are commonly written in Java, then compiled to bytecode by the Java Virtual Machine (JVM), which is then translated to Dalvik bytecode. ART replaces Dalvik as Android’s process virtual machine to execute applications written in Android. However, unlike Dalvik, the transformations of the application’s bytecode into native instructions are later executed by the device’s runtime environment.

There are major improvements ART brings to the table that were lacking in Dalvik. Dalvik uses Just-In-Time (JIT) compilation to compile bytecode, whereas ART uses Ahead-of-Time (AoT)compilation by performing the transformation at/upon the installation of an application. By reducing the amount of compilation done across the application operation, a device’s processor usage is reduced and battery duration is improved tremendously. The advantages of ART are its improved performance, garbage collection, and applications’ debugging and profiling. Moreover, the differences are the compilation approach (JIT vs AOT), Boot Time (Dalvik’s boot time is faster than ART’s), space usage (ART uses more space to run an application), speed (ART is much faster and smoother than Dalvik).

**3. What is the android manifest used for?**

Android manifest is used to describes your app’s essential information to the build tools of Android, OS, and Google Play. This essential information about the application are required by the system before it can run any of the application’s code. It is also used for declaring the app’s package name, components of the app, permissions that the app needs to access protected parts of the system or apps and to declare the hardware and software features the app needs. The package name serves as a unique identifier for the application. The application components declared in the android manifest usually include the activities, services, broadcast receivers and content providers.

The android manifest is also used to determine the processes that host the application components. Moreover, it declares the minimum level of the Android API that the application requires and it lists the libraries that the application must be linked against.

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

Screen size is grouped into categories of small, medium, large, extra large, double extra, and triple extra. Screen density is the amount of pixels within the screen area. It is usually measured in dots per inch (DPI). Screen density is grouped into categories of low, medium, high and extra high. Resolution is the total number of pixels in the screen.

* sp: Scale Independent Pixel. It is scaled based on the user’s font size preference. Fonts should use sp.
* dp: Density Independent Pixel. dp varies based on screen density. In 160 dpi screen, 1 dp equals to 1 pixel. It is recommended to use dp if it’s not for font size.
* px: The usual standard pixel, which maps to the screen pixel.
* pt: is 1/72 of an inch, with respect to the physical screen size.
* in: is inches, with respect to the physical screen size.
* mm: is millimeters, with respect to the physical screen size.

It is recommended to use sp for font sizes and dp for everything else because it will make the UI compatible for android devices with different densities.

**5. Describe what each section of the Android Platform arch. details.**

Android Platform Architecture

* Linux Kernel:
  + is the foundation of the Android platform. It contains low level drivers for hardware components support, such as camera, keypad, display, etc. Android Runtime relies on Linux Kernel for system services, such as:
    - Memory, process management and threading
    - Network stack
    - Driver model
    - Security
* Hardware Abstraction Layer (HAL):
  + provides abstraction between hardware and the rest of software stack. Audio, Bluetooth, Camera and Sensors are some of the examples.
* Native C/C++ Libraries:
  + These are exposed to developers through Android Application Framework.
  + Contains C/C++ libraries used by the components of Android devices. Some of the features include here are the OpenGL (Open Graphics Library) and SGL (Scalable Graphics Library), and WebKit library to provide Web Browsing engine and other features.
* Android Runtime (ART):
  + Is designed to run apps in an environment where battery, processing and memory are limited.
  + ART allows each app to run in its own process within its own instance of ART VM, which makes process management more efficient.
  + ART also has a very efficient garbage collection and is capable to do both Ahead-of-Time (AOT) and Just-In-Time (JIT) compilation.
* Application Framework (Java API Framework):
  + is a collection of APIs written in Java, which gives developers access to the Android OS complete features.
  + Enables code reuse for core components and services, such as:
    - Activity Manager
    - Window Manager
    - Content Providers
    - View System
    - Package Manager
    - Telephony Manager
    - Resource Manager
    - Location Manager
    - Notification Manager
* System apps:
  + Are downloadable from the Android/Google Play Store along with other applications
  + This architecture layer is at the top and it uses all the layers below in order for these apps to function properly.
  + Applications written by developers are to be installed in this layer only.
  + Some examples of applications are:
    - Browser
    - Calendar
    - Maps
    - Contacts

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

* View is the basic building blocks or class for all Android UI components. It is just a simple rectangular box which responds to the user’s actions. Some of the most common View subclasses are as follow:
  + TextView
  + EditText
  + Button
  + WebView
  + ImageView
  + CheckBox
  + RadioButton
  + DatePicker
  + TimePicker
  + ListView
  + RecyclerView
* ViewGroup is a subclass of View and it is a base class for layouts. ViewGroup is just like an invisible placeholder or container to hold other Views or ViewGroups. ViewGroup is used to define the layout properties. For example, LinearLayout is the ViewGroup that contains Views, such as Button, TextView, EditText, etc. Some of the commonly used ViewGroups are as follow:
  + LinearLayout
  + RelativeLayout
  + ConstraintLayout
  + FrameLayout
  + GridView
* View Hierarchy is a View inside another View. The outer View becomes the inner View’s parent and the inner View is its child. It is basically just nested views. These nested views can be done in complicated ways. This whole structure is referred to as View Hierarchy.

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

All of these ViewGroups must have layout\_width and layout\_height implementation.

* ConstraintLayout: is a large and complex layout with a flat view hierarchy. This layout is not a nested ViewGroup. Each constraint represents a connection to another view. It defines the positioning of the view in respect to the vertical or horizontal axis. Each View must have a minimum of one constraint for each axis. This layout must have its layout\_constraint attributes be implemented in the activity xml file.
* LinearLayout: is a ViewGroup that aligns all the children Views in a single direction, either vertically or horizontally. This layout has its unique items implemented, such as its orientation attribute to set the direction of the layout, its layout weight and layout gravity attributes.
* CoordinatorLayout: is a super-powered FrameLayout. This layout is intended as a top level application décor or layout and as a container for an interaction with child views. CoordinatorLayout has an ability to coordinate the animations and transitions of the views within the layout itself. Its unique items that must be implemented are its behaviors, anchor and insetEdge.
* GridLayout: is a layout that places its children views in a rectangular grid. The rectangular grid is a set of thin lines that divides the viewing area into cells. Its unique items that must be implemented are its LayoutParams must be defined to either row spec or column spec, setOrientation, setRowCount, and setColumnCount. Each of these specs defines the set of rows or columns and how the children views should be aligned.
* RelativeLayout: is a ViewGroup that displays child views in a relative position. Each view’s position can be specified as relative to sibling views or it could also be in positions relative to the parent’s relativelayout area. This layout has advantage in designing a user interface due to the fact that it can eliminate nested ViewGroups and keep your view hierarchy flat and to improve performance. The properties available uniquely to Relative Layout are the alignParentTop, centerVertical, below, and toRightOf.