**What is cold start in android?**

A cold start refers to an app’s starting from scratch, the system’s process has not, until this start, created the app’s process. Cold starts happen in cases such as your app’s being launched for the first time since the device booted, or since the system killed the app.

**What are 9-Patch images? List examples of them.**

A 9-Patch image is a stretchable bitmap image that you can use as the background of a view. Android automatically resizes the graphic to accommodate the contents of the view. It is a standard PNG image that includes an extra 1-pixel border.

**What are SVG (Vector Images)? How do we support them in Android?**

Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. The SVG specification is an open standard developed and recommended by the World Wide Web Consortium (W3C) and it integrates with other W3C standards such as the DOM and XSL.

While Android does not support SVGs directly, with the launch of Lollipop a new class was introduced called VectorDrawable. This allows developers to draw assets in a similar fashion using only code.

A VectorDrawable is a vector graphic defined in an XML file as a set of points, lines, and curves along with its associated color information. Vector drawable enables images to be scaled without loss of display quality, which means the same file is resized for different screen densities without loss of image quality. Vector images can also be used for animation by using multiple XML files instead of multiple images for each display resolution.

**What are the process priorities in Android?**

Android places each process into an "importance hierarchy" based on the components running in them and the state of those components to determine which processes should be killed when low on memory. These process types are in order of importance;

* Foreground process: This is the process that is required for what the user is currently doing.
* Visible process: This is doing the work that the user is currently aware of, so killing it would have a noticeable negative impact on the user experience.
* Service process: This process holds a Service that has been started with the startService() method. Though these processes are not directly visible to the user, they are generally doing things that the user cares about (such as background network data upload or download), so the system will always keep such processes running unless there is not enough memory to retain all foreground and visible processes.
* Background process: Processes hosting Activities that aren’t visible and that don’t have any running Services. There will generally be many background processes that Android will kill using a last-seen-first-killed pattern to obtain resources for foreground processes.
* Empty process: To improve overall system performance, Android will often retain an application in memory after it has reached the end of its lifetime. Android maintains this cache to improve the start-up time of applications when they’re relaunched. These processes are routinely killed, as required.

**What are the different types of services in Android?**

* Started services
* Bound services
* Foreground services
* Scheduled services
* Background services

**What are the main differences between Intent and Services?**

Service is a base class of service implementation. It can be extended for creating any service. Since the service run inside the application's main thread, it may reduce the application performance. while Intent is a base class for Services that handle asynchronous requests expressed as Intents on demand using a worker thread.

Service class uses the application’s main thread, while Intent creates a worker thread and uses that thread to run the service.

**What is the job of Intent service? Why do we need it?**

The Intent Service is used to perform a certain task in the background. Once done, the instance of Intent Service terminate itself automatically. Examples for its usage would be to download a certain resource from the Internet.

Intent service is needed for;

Thread management: It automatically processes all incoming requests in a separate thread taking the burden of thread management away from the developer.

Request Queue: It queues up all the incoming requests and they are processed one by one

Stop point: Once the request Queue is empty it automatically stops itself, so the developer need not worry about handling the service lifecycle.

**What is Alarm manager? What is Job scheduler?**

* Alarm Manager provides access to the system alarm services. These allow you to schedule your application to be run at some point in the future. When an alarm goes off, the Intent that had been registered for it is broadcast by the system, automatically starting the target application if it is not already running.
* Job Scheduler is an API for scheduling various types of jobs against the framework that will be executed in the application's own process.

**Explain what is Doze mode.**

Doze mode is a feature in Marshmallow, which prevents certain tasks from running if your device is in idle state. Doze in devices reduces power consumption by deferring background CPU and Network activity for applications.

**What happens to the lifecyle of Activity A and Activity B**

…Not clear…

**How to persist data between configuration changes?**

To properly handle configuration changes, it is important that the activity restores its previous state. A combination of onSaveInstanceState(), ViewModel objects, and persistent storage can be used to save and restore the UI state of your activity across configuration changes.

**Kinds of Launch Modes?**

Standard, SingleTop, SingleTask, SingleInstance

**Kinds of intents:**

Explicit, Implicit

**When to use a Service vs a Thread?**

A Service is meant to run your task independently of the Activity, it allows you to run any task in background. This run on the main UI thread so when you want to perform any network or heavy load operation then you have to use the Thread there.

Threads run a task in its own thread instead of main UI thread. You would use a thread when performing some heavy network operation like sending bytes to the server continuously.

**How to ensure thread safety?**

Thread safety can be achieved by avoiding shared state. For the state, you need to share you can either use message parsing together with immutable classes or the concurrent data structures together with synchronized blocks and volatile fields.

**How to resume a stopped Service?**

Service can be started and stopped from Activity using startService(intent) and stopService(intent) respectively by sending Intent messages. Service starts a new thread to run in background. Android provides HandlerThread class to start a thread with Looper.

**How will you implement a client-server connection to a service?**

Bound Services can be used to implement a client-server connection to a service. A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

**Criteria that the android system uses to determine which processes to kill in case it needs to?**

Android places each process into an "importance hierarchy" based on the components running in them and the state of those components to determine which processes should be killed when low on memory. These process types are in order of importance;

* Foreground process: This is the process that is required for what the user is currently doing.
* Visible process: This is doing the work that the user is currently aware of, so killing it would have a noticeable negative impact on the user experience.
* Service process: This process holds a Service that has been started with the startService() method. Though these processes are not directly visible to the user, they are generally doing things that the user cares about (such as background network data upload or download), so the system will always keep such processes running unless there is not enough memory to retain all foreground and visible processes.
* Background process: Processes hosting Activities that aren’t visible and that don’t have any running Services. There will generally be many background processes that Android will kill using a last-seen-first-killed pattern to obtain resources for foreground processes.
* Empty process: To improve overall system performance, Android will often retain an application in memory after it has reached the end of its lifetime. Android maintains this cache to improve the start-up time of applications when they’re relaunched. These processes are routinely killed, as required.

**How to pass objects to activities?**

Objects can be passed to activities either by using Serializeable or Parceleable.

**Differences between Serializeable and Parceleable?**

Serialization is a mechanism of converting the state of an object into byte streams that includes the object data as well as information about the object type and the type of data stored in it. It has an automatic serialization process thereby creates lots temporary objects that garbage collection while Parcelable is an Android specific interface which allows users to serialize objects that can be transferred between two activities. It has custom implementation and more efficient than serializable.

**When to use an asynctask?**

AsyncTask class allows to run instructions in the background and to synchronize again with the main thread. It also reports progress of the running tasks. AsyncTasks should be used for short background operations which need to update the user interface.

**ListView vs RecyclerView**

RecyclerView integrated animations for adding, updating and removing items, enforces the recycling of views by using the ViewHolder pattern, supports both vertical and horizontal scrolling and can be used together with DiffUtil while ListView doesn’t embrace the usage of the ViewHolder pattern, supports only vertical scrolling, simple to use compared to RecyclerView.

**How to implement custom content provider?**

Implementing a custom content provider involves the following steps:

* Create a class that extends ContentProvider
* Create a contract class
* Create the UriMatcher definition
* Implement the onCreate() method
* Implement the getType() method
* Implement the CRUD methods
* Add the content provider to your AndroidManifest.xml

**Different kinds of permissions?**

* Normal permissions
* Signature permissions
* Dangerous permissions
* Special permissions

**How to have a consistent layout between devices?**

By having a consistent layout between devices, the app can be made available to the greatest number of users with different devices, using a single APK. This can be achieved with the following techniques:

* Use view dimensions that allow the layout to resize.
* Create alternative UI layouts according to the screen configuration.
* Provide bitmaps that can stretch with the views.

**Advantages of using RxJava?**

There are many benefits of using Rx Java. Some of which are;

* Callback: RxJava renders the way to terminate the urgency to build layers of callbacks.
* Multithreading: It helps to perform several complex operations in any thread, maintaining proper synchronization and switching threads effortlessly.
* Error Handling: RxJava helps to identify errors on the surfaced.
* Code Production: RxJava helps to simplify the code that is required to handle data and events by allowing it to describe the desired goal, rather than writing a list of instructions for the app.
* Flexibility: Once Observables are set, RxJava serves a large range of operators that can be used to filter, merge and transform the data that are omitted by the Observables.

**Cold vs Hot observables?**

A Hot observable emits items all the time, even when no one is listening while a Cold observable only emits items when it has a subscriber.

**Error handling operators in RxJava?**

* onErrorResumeNext: Instructs an Observable to emit a sequence of items if it encounters an error
* onErrorReturn: Instructs an Observable to emit a particular item when it encounters an error
* onExceptionResumeNext: Instructs an Observable to continue emitting items after it encounters an exception (but not another variety of throwable)
* retry: If a source Observable emits an error, resubscribe to it in the hopes that it will complete without error
* retryWhen: If a source Observable emits an error, pass that error to another Observable to determine whether to resubscribe to the source
* doOnError: It can be used to intercept an error before the error makes its way to the consumer.
* onErrorReturnItem: It just returns a value if an error is encountered

**What is Dagger2?**

Dagger is a fully static, compile-time dependency injection framework for both Java and Android. It is an adaptation of an earlier version created by Square and now maintained by Google.

Dagger 2 is the first to implement the full stack with generated code. The guiding principle is to generate code that mimics the code that a user might have hand-written to ensure that dependency injection is as simple, traceable and performant as it can be.

**What is Live Data and ViewModels?**

LiveData is an observable data holder class. Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services. This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.

The ViewModel class is designed to store and manage UI-related data in a lifecycle conscious way. The ViewModel class allows data to survive configuration changes such as screen rotations.

**MVP vs MVVM?**

The responsibility of the Model View Controller is to process incoming requests. It gets the input from users via the View, then processes the user’s data through the Model, passing back the results to View. It normally acts as a mediator between the View and the Model while The Model View View Model is responsible for displaying methods, commands, and other functions that assist in maintaining the state of the View, manipulating the Model as the result of actions on the View, and triggering the events in the View itself.