Pseudo code

Make new variables with given initial values:

flag = false

fastforward_increment = 5000

rewind_increment = 5000

startTime = 0

finalTime = 0

Set activity force to full screen state

Set video streaming-path within MediaSource

Start the video player

finalTime = total duration time of the video

startTime = current position time of the video

When player state changed:
If playback state in buffering mode:
Set progress bar view to Visible
Else if playback state in Ready mode:
Set progress bar view to Invisible

When the play button is clicked:
Start the video player
Set pause button is visible and play button is invisible

When the pause button is clicked:
Pause the video player
Set play button is visible and pause button is invisible

When the forward button is clicked:
If fastforward_increment time is less than or equal to final time:
startTime = startTime + fastforward_increment

When the rewind button is clicked:
If rewind_increment time is greater than zero:
startTime = startTime - rewind_increment

When the full screen button is clicked:
    If flag value is true:
        Request for change screen to portrait mode and set flag value is ‘false’
    Else:
        Request for change screen to Landscape mode and set flag value is ‘true’

Application Overview

To develop this application I have used the Library: exoplayer: 2.7.3 for the video player. Whereby all the functionalities of the application are integrated within the XML file. Such as fast forward, rewind, buffering, and player UI. ExoPlayer is a library, we can easily take advantage of new features as they become available by updating your app. ExoPlayer supports features like Dynamic adaptive streaming over HTTP (DASH), SmoothStreaming and Common Encryption, which are not supported by MediaPlayer.

The application also might be developed within built-in MediaPlayer control within VideView component. But using the library it’s designed to be easy to customize and extend.

Design pattern that I have considered for this application: (Proxy Design Pattern)

ExoPlayer library skeleton can be summed up in three significant classes that stick together the greater part of the playback segments: SimpleExoPlayer, ExoPlayerImpl and ExoPlayerImplInternal. The connection between those three classes is the great example of the pattern.

The SimpleExoPlayer class implements the ExoPlayer interface and instantiates the ExoPlayerImpl class in its constructor. The ExoPlayerImpl class also implements the ExoPlayer interface and in turn instantiates the ExoPlayerImplInternal class, which in essence implements the core playback logic.

SimpleExoPlayer: The most high level class, implements ExoPlayer interface. Within its constructor, it has initialized an ExoPlayerImpl instance. Overall, SimpleExoPlayer is a wrapper class that hides the implementation details and exposes a simple API for developers to use directly.

ExoPlayerImpl: This class has a dual purpose. First it acts as an intermediate wrapper that hides the real playback logic of different ExoPlayer's components and their interactions. In addition, it behaves like a message receiver, receiving ExoPlayer's playback state changes and immediately triggering different event listener's callbacks that developers can choose to react to.

Some key-points that are very important to keep in mind for when we look into the next implementation layer in ExoPlayerImplInternal:

- The eventHandler member variable in the constructor is a handler attached to UIThread
- The listener object keeps track of all registered ExoPlayer Listeners. Such as Play/Pause, forward, rewind button, etc integrating process.
- The handleEvent() method is only invoked by the eventHandler. This is a fundamental concurrency pattern in Android, the handler binds to a thread, UIThread in this case, so others can later send code blocks to execute on that thread.

ExoPlayerImplInternal

The bulk of the actual logic lives in this class. It acts as a glue to all the different components needed for media playback. All of the code above has so far been executed on MainThread, and it makes senses to do that because user behaviors are mostly triggered on MainThread for reasons that it wants users to know immediately what is happening when ExoPlayer changes states. However, the when dealing with operations across a network boundary such as downloading manifest playlists, video segments files, rendering byte array on screens, etc, we do not want to be blocking the main thread. Instead, blocking or CPU intensive code shoud be executed on a worker thread. To facilitate that, ExoPlayerImplInternal:

1. Creates a HandlerThread instance in its constructor. This is the worker thread that all internal playback code logic runs on.
2. It implements Handler.Callback, needed to enable code to be executed on the worker thread, such as video progress, timing state and play, forward, and rewin functionalities control over user through.
3. It receives the previously mentioned eventHandler in its constructor. The eventHandler acts as a communication channel between the UIThread and the worker thread.

Summery

The ExoPlayer library has separated ExoPlayer interface implementations into three different layers:

SimpleExoPlayer: api level, easy to use getters and setters.

ExoPlayerImpl: informational and transitional level, monitor playback state changes

ExoPlayerImplInternal: playback level, implementing core playback logic.

By doing so, a lot of low level implementation details are hidden in lower layers, reducing the overall project complexity.