* **JAVA CODE**
* **AutoPlay**
* **VideoAutoPlayHelper**
* package com.nehak.instagramfeed.autoPlay  
    
  import android.graphics.Rect  
  import androidx.recyclerview.widget.LinearLayoutManager  
  import androidx.recyclerview.widget.RecyclerView  
  import com.nehak.instagramfeed.feedUI.adapters.FeedAdapter  
  import com.nehak.instagramfeed.feedUI.holders.FeedViewHolder  
  import com.nehak.instagramfeed.feedUI.holders.VideoFeedViewHolder  
  import com.nehak.instagramfeed.player.InstaLikePlayerView  
    
  */\*\*  
   \* Create By SUBHANKAR, RAKESH, RAVI  
   \*/*class VideoAutoPlayHelper(var recyclerView: RecyclerView) {  
    
   private var lastPlayerView: InstaLikePlayerView? = null  
   val MIN\_LIMIT\_VISIBILITY =  
   20; // When playerView will be less than 20% visible than it will stop the player  
    
   var currentPlayingVideoItemPos = -1; // -1 indicates nothing playing  
    
   fun onScrolled(recyclerView: RecyclerView, feedAdapter: FeedAdapter) {  
    
   val firstVisiblePosition: Int = findFirstVisibleItemPosition()  
   val lastVisiblePosition: Int = findLastVisibleItemPosition()  
    
   val pos = getMostVisibleItem(firstVisiblePosition, lastVisiblePosition)  
    
   if (pos == -1) {  
   /\*check if current view is more than MIN\_LIMIT\_VISIBILITY\*/  
   if (currentPlayingVideoItemPos != -1) {  
   val viewHolder: RecyclerView.ViewHolder =  
   recyclerView?.findViewHolderForAdapterPosition(currentPlayingVideoItemPos)!!  
    
   val currentVisibility = getVisiblePercentage(viewHolder);  
   if (currentVisibility < MIN\_LIMIT\_VISIBILITY) {  
   lastPlayerView?.removePlayer()  
   }  
   currentPlayingVideoItemPos = -1;  
   }  
    
    
   } else {  
    
   if (currentPlayingVideoItemPos != pos) {  
   currentPlayingVideoItemPos = pos;  
   attachVideoPlayerAt(pos);  
   }  
    
   }  
    
   }  
    
   private fun attachVideoPlayerAt(pos: Int) {  
   val feedViewHolder: FeedViewHolder =  
   (recyclerView.findViewHolderForAdapterPosition(pos) as FeedViewHolder?)!!  
    
   if(feedViewHolder is VideoFeedViewHolder) {  
   */\*\* in case its a video\*\*/* if (lastPlayerView==null || lastPlayerView != feedViewHolder.customPlayerView) {  
   feedViewHolder.customPlayerView.startPlaying()  
   // stop last player  
   lastPlayerView?.removePlayer();  
   }  
   lastPlayerView = feedViewHolder.customPlayerView;  
    
   } else {  
   */\*\* in case its a image\*\*/* if (lastPlayerView != null) {  
   // stop last player  
   lastPlayerView?.removePlayer();  
   lastPlayerView = null  
   }  
    
   }  
   }  
    
   private fun getMostVisibleItem(firstVisiblePosition: Int, lastVisiblePosition: Int): Int {  
    
   var maxPercentage = -1;  
   var pos = 0;  
   for (i in firstVisiblePosition..lastVisiblePosition) {  
   val viewHolder: RecyclerView.ViewHolder =  
   recyclerView.findViewHolderForAdapterPosition(i)!!  
    
   var currentPercentage = getVisiblePercentage(viewHolder);  
   if (currentPercentage > maxPercentage) {  
   maxPercentage = currentPercentage.toInt();  
   pos = i;  
   }  
    
   }  
    
   if (maxPercentage == -1 || maxPercentage < MIN\_LIMIT\_VISIBILITY) {  
   return -1;  
   }  
    
   return pos;  
   }  
    
   private fun getVisiblePercentage(  
   holder: RecyclerView.ViewHolder  
   ): Float {  
   val rect\_parent = Rect()  
   recyclerView.getGlobalVisibleRect(rect\_parent)  
   val location = IntArray(2)  
   holder.itemView.getLocationOnScreen(location)  
    
   val rect\_child = Rect(  
   location[0],  
   location[1],  
   location[0] + holder.itemView.getWidth(),  
   location[1] + holder.itemView.getHeight()  
   )  
    
   val rect\_parent\_area =  
   ((rect\_child.right - rect\_child.left) \* (rect\_child.bottom - rect\_child.top)).toFloat()  
   val x\_overlap = Math.max(  
   0,  
   Math.min(rect\_child.right, rect\_parent.right) - Math.max(  
   rect\_child.left,  
   rect\_parent.left  
   )  
   ).toFloat()  
   val y\_overlap = Math.max(  
   0,  
   Math.min(rect\_child.bottom, rect\_parent.bottom) - Math.max(  
   rect\_child.top,  
   rect\_parent.top  
   )  
   ).toFloat()  
   val overlapArea = x\_overlap \* y\_overlap  
   val percent = overlapArea / rect\_parent\_area \* 100.0f  
    
   return percent  
   }  
    
    
   private fun findFirstVisibleItemPosition(): Int {  
   if (recyclerView.*layoutManager* is LinearLayoutManager) {  
   return (recyclerView.*layoutManager* as LinearLayoutManager?)!!.findFirstVisibleItemPosition()  
   }  
    
   return -1  
   }  
    
   private fun findLastVisibleItemPosition(): Int {  
   if (recyclerView.getLayoutManager() is LinearLayoutManager) {  
   return (recyclerView.getLayoutManager() as LinearLayoutManager).findLastVisibleItemPosition()  
   }  
   return -1  
   }  
    
   fun startObserving() {  
    
   recyclerView.addOnScrollListener(object : RecyclerView.OnScrollListener() {  
   override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {  
   super.onScrolled(recyclerView, dx, dy)  
   if (recyclerView.*adapter* is FeedAdapter) {  
   onScrolled(recyclerView, recyclerView.*adapter* as FeedAdapter)  
   } else {  
   throw IllegalStateException("Adapter should be FeedAdapter or extend FeedAdapter")  
   }  
   }  
   })  
   }  
    
  }
* **VideoPreLoadingService**

package com.nehak.instagramfeed.autoPlay;  
  
import android.app.IntentService  
import android.content.Context  
import android.content.Intent  
import android.net.Uri  
import com.google.android.exoplayer2.upstream.DataSource  
import com.google.android.exoplayer2.upstream.DataSpec  
import com.google.android.exoplayer2.upstream.DefaultDataSourceFactory  
import com.google.android.exoplayer2.upstream.cache.CacheKeyFactory  
import com.google.android.exoplayer2.upstream.cache.CacheUtil  
import com.google.android.exoplayer2.upstream.cache.SimpleCache  
import com.google.android.exoplayer2.util.Util  
import com.nehak.instagramfeed.R  
import com.nehak.instagramfeed.other.Lg  
import com.nehak.instagramfeed.other.Constants  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.GlobalScope  
import kotlinx.coroutines.Job  
import kotlinx.coroutines.async  
  
*/\*\*  
 \*  
 \* This class provides way to precache the video from remoteUrl in to Cache Directory  
 \* So Player will not reload videos from server if they are already loaded in cache  
 \*/*class VideoPreLoadingService : IntentService(VideoPreLoadingService::class.*java*.*simpleName*) {  
 private lateinit var mContext: Context  
 private var simpleCache: SimpleCache? = null  
 private var cachingJob: Job? = null  
 private var videosList: ArrayList<String>? = null  
  
 override fun onHandleIntent(intent: Intent?) {  
 mContext = *applicationContext* if (intent != null) {  
 val extras = intent.*extras* videosList = extras?.getStringArrayList(Constants.VIDEO\_LIST)  
  
 if (!videosList.*isNullOrEmpty*()) {  
 preCacheVideo(videosList)  
 }  
 }  
 }  
  
 private fun preCacheVideo(videosList: ArrayList<String>?) {  
 var videoUrl: String? = null  
 if (!videosList.*isNullOrEmpty*()) {  
 videoUrl = videosList[0]  
 videosList.removeAt(0)  
 } else {  
 stopSelf()  
 }  
 if (!videoUrl.*isNullOrBlank*()) {  
 val videoUri = Uri.parse(videoUrl)  
 val dataSpec = DataSpec(videoUri)  
  
 val defaultCacheKeyFactory = CacheUtil.*DEFAULT\_CACHE\_KEY\_FACTORY* val progressListener =  
 CacheUtil.ProgressListener **{** requestLength, bytesCached, newBytesCached **->** val downloadPercentage: Double = (bytesCached \* 100.0  
 / requestLength)  
  
 Lg.v("VideoPreLoadingService", "downloadPercentage url" + videoUrl)  
 Lg.v("VideoPreLoadingService", "downloadPercentage % = " + downloadPercentage)  
 **}** val dataSource: DataSource =  
 DefaultDataSourceFactory(  
 mContext,  
 Util.getUserAgent(this, getString(R.string.*app\_name*))  
 ).createDataSource()  
  
 cachingJob = GlobalScope.*async*(Dispatchers.IO) **{** cacheVideo(dataSpec, defaultCacheKeyFactory, dataSource, progressListener)  
 preCacheVideo(videosList)  
 **}** cachingJob?.start();  
 }  
 }  
  
 private fun cacheVideo(  
 dataSpec: DataSpec,  
 defaultCacheKeyFactory: CacheKeyFactory?,  
 dataSource: DataSource,  
 progressListener: CacheUtil.ProgressListener  
 ) {  
 try {  
 CacheUtil.cache(  
 dataSpec,  
 simpleCache,  
 defaultCacheKeyFactory,  
 dataSource,  
 progressListener,  
 null  
 )  
 } catch (e: java.io.EOFException) {  
 Lg.printStackTrace(e)  
 } catch (e: Exception) {  
 Lg.printStackTrace(e)  
 }  
 }  
  
 override fun onDestroy() {  
 super.onDestroy()  
 cachingJob?.cancel()  
  
 }  
}

* **customViewBinding**
* package com.nehak.instagramfeed.customViewBinding  
    
  import androidx.databinding.BindingAdapter  
  import androidx.recyclerview.widget.RecyclerView  
    
    
  */\*\*  
   \* Create By SUBHANKAR, RAKESH, RAVI  
   \*/*@BindingAdapter(value = ["setAdapter"])  
  fun RecyclerView.bindRecyclerViewAdapter(adapter: RecyclerView.Adapter<\*>?) {  
   this?.*run* **{** this.setHasFixedSize(true)  
   this.*adapter* = adapter  
   **}**}
* **dataModels**
* package com.nehak.instagramfeed.dataModels  
    
  */\*\*  
   \*/  
  \* Create By SUBHANKAR, RAKESH, RAVI*data class FeedItem(val thumbnailIndex: Int?) {  
   var dimension: String? = null;  
   var downloadUrl: String? = null;  
  }
* **feedUI**
* **adapters**
* **FeedAdapter**
* package com.nehak.instagramfeed.feedUI.adapters  
    
  import android.content.Context  
  import android.graphics.drawable.Drawable  
  import android.net.Uri  
  import android.view.LayoutInflater  
  import android.view.View  
  import android.view.ViewGroup  
  import android.widget.ImageView  
  import androidx.constraintlayout.widget.ConstraintLayout  
  import androidx.recyclerview.widget.DiffUtil  
  import androidx.recyclerview.widget.ListAdapter  
  import com.nehak.instagramfeed.dataModels.FeedItem  
  import com.nehak.instagramfeed.databinding.HorizontalItemFeedBinding  
  import com.nehak.instagramfeed.databinding.ItemFeedBinding  
  import com.nehak.instagramfeed.feedUI.holders.FeedViewHolder  
  import com.nehak.instagramfeed.feedUI.holders.ImageFeedViewHolder  
  import com.nehak.instagramfeed.feedUI.holders.VideoFeedViewHolder  
  import com.nehak.instagramfeed.other.Constants  
    
    
  */\*\*  
   \* Create By SUBHANKAR, RAKESH, RAVI  
   \*/*class FeedAdapter(val context: Context) :  
   ListAdapter<FeedItem, FeedViewHolder>(DIFF\_CALLBACK) {  
    
   companion object {  
   */\*\* Mandatory implementation inorder to use "ListAdapter" - new JetPack component" \*\*/* val DIFF\_CALLBACK = object : DiffUtil.ItemCallback<FeedItem>() {  
   override fun areItemsTheSame(oldItem: FeedItem, newItem: FeedItem): Boolean {  
   return false;// oldItem == newItem  
   }  
    
   override fun areContentsTheSame(oldItem: FeedItem, newItem: FeedItem): Boolean {  
   return false;//oldItem == newItem  
   }  
    
   }  
   val TOTAL\_ITEMS = 200;  
    
   const val FEED\_TYPE\_VIDEO = 1;  
   const val FEED\_TYPE\_IMAGES\_MULTIPLE = 2;  
   }  
    
   override fun getItemViewType(position: Int): Int {  
   if (position % 2 == 0) {  
   return FEED\_TYPE\_VIDEO;  
   } else {  
   return FEED\_TYPE\_IMAGES\_MULTIPLE  
   }  
   }  
    
    
   override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): FeedViewHolder {  
    
   if (viewType == FEED\_TYPE\_IMAGES\_MULTIPLE) {  
   return ImageFeedViewHolder(  
   HorizontalItemFeedBinding.inflate(  
   LayoutInflater.from(  
   parent.*context* ), parent, false  
   )  
   )  
   }  
   return VideoFeedViewHolder(  
   ItemFeedBinding.inflate(LayoutInflater.from(parent.*context*), parent, false)  
   )  
   }  
    
   override fun onBindViewHolder(holder: FeedViewHolder, position: Int) {  
    
   if (holder is VideoFeedViewHolder) {  
   handleViewHolder(holder, position)  
   } else if (holder is ImageFeedViewHolder) {  
   handleViewHolder(holder, position)  
   }  
    
   }  
    
   private fun handleViewHolder(holder: VideoFeedViewHolder, position: Int) {  
   /\*Reset ViewHolder \*/  
  // removeImageFromImageView(holder.videoThumbnail)  
   holder.customPlayerView.reset()  
    
   /\*Set seperate ID for each player view, to prevent it being overlapped by other player's changes\*/  
   holder.customPlayerView.*id* = View.generateViewId()  
    
   /\*circlular repeatation of items\*/  
   val videoPos = (position % Constants.videoList.size);  
    
   /\*Set ratio according to video\*/  
   (holder.videoThumbnail.*layoutParams* as ConstraintLayout.LayoutParams).dimensionRatio =  
   Constants.videoList.get(videoPos).dimension  
    
   /\*Set video's direct url\*/  
   holder.customPlayerView.setVideoUri(Uri.parse(Constants.videoList.get(videoPos).downloadUrl))  
    
   /\*Set video's thumbnail locally (by drawable), you can set it by remoteUrl too\*/  
   val resID: Int = context.getResources().getIdentifier(  
   "thumbnail\_" + videoPos,  
   "drawable",  
   context.getPackageName()  
   )  
    
   val res: Drawable = context.getResources().getDrawable(resID, null)  
   holder.videoThumbnail.setImageDrawable(res);  
   }  
    
   private fun handleViewHolder(holder: ImageFeedViewHolder, position: Int) {  
    
   /\* Set adapter (items are being used inside adapter, you can setup in your own way\*/  
   val feedAdapter = ImageAdapter(holder.itemView.*context*, position)  
   holder.recyclerViewImages.*adapter* = feedAdapter  
    
    
   }  
    
   override fun getItemCount(): Int {  
   return TOTAL\_ITEMS;  
   }  
    
   fun removeImageFromImageView(imageView: ImageView) {  
   try {  
   imageView.*background* = null  
   imageView.setImageDrawable(null)  
   } catch (e: Exception) {  
   e.printStackTrace()  
   }  
   }  
    
  }
* **ImageAdapter**
* package com.nehak.instagramfeed.feedUI.adapters  
    
  import android.content.Context  
  import android.view.LayoutInflater  
  import android.view.ViewGroup  
  import android.widget.ImageView  
  import androidx.recyclerview.widget.DiffUtil  
  import androidx.recyclerview.widget.ListAdapter  
  import com.nehak.instagramfeed.other.Extensions.Companion.getResource  
  import com.nehak.instagramfeed.dataModels.FeedItem  
  import com.nehak.instagramfeed.databinding.ImageItemSingleBinding  
  import com.nehak.instagramfeed.feedUI.holders.ImageViewHolder  
  import kotlin.random.Random  
    
    
  */\*\*  
   \* Create By SUBHANKAR, RAKESH, RAVI  
   \*/*class ImageAdapter(val context: Context, val parentPosition: Int) :  
   ListAdapter<FeedItem, ImageViewHolder>(DIFF\_CALLBACK) {  
    
   companion object {  
   */\*\* Mandatory implementation inorder to use "ListAdapter" - new JetPack component" \*\*/* val DIFF\_CALLBACK = object : DiffUtil.ItemCallback<FeedItem>() {  
   override fun areItemsTheSame(oldItem: FeedItem, newItem: FeedItem): Boolean {  
   return false;// oldItem == newItem  
   }  
    
   override fun areContentsTheSame(oldItem: FeedItem, newItem: FeedItem): Boolean {  
   return false;//oldItem == newItem  
   }  
    
   }  
    
   public val TOTAL\_IMAGES = 4;  
   }  
    
    
   override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ImageViewHolder {  
    
   return ImageViewHolder(  
   ImageItemSingleBinding.inflate(LayoutInflater.from(parent.*context*), parent, false)  
   )  
   }  
    
   override fun onBindViewHolder(holder: ImageViewHolder, position: Int) {  
    
   /\* Show any random image from drawable\*/  
   val posToPick = (parentPosition+position)%7;  
   holder.imageView.setImageDrawable(holder.itemView.*context*.*getResource*("image\_"+posToPick));  
    
   }  
    
   override fun getItemCount(): Int {  
   return TOTAL\_IMAGES;  
   }  
    
   fun removeImageFromImageView(imageView: ImageView) {  
   try {  
   imageView.*background* = null  
   imageView.setImageDrawable(null)  
   } catch (e: Exception) {  
   e.printStackTrace()  
   }  
   }  
    
   fun rand(start: Int, end: Int): Int {  
   *require*(start <= end) **{** "Illegal Argument" **}** val rand = *Random*(System.nanoTime())  
   return (start..end).*random*(rand)  
   }  
  }
* **holders**
* **FeedViewHolder**
* package com.nehak.instagramfeed.feedUI.holders  
    
  import android.view.View  
  import androidx.recyclerview.widget.RecyclerView  
    
  */\*\*  
   \* Create By SUBHANKAR, RAKESH, RAVI  
   \*/*abstract class FeedViewHolder(root: View) : RecyclerView.ViewHolder(root) {  
    
  }
* **ImageFeedViewHolder**
* package com.nehak.instagramfeed.feedUI.holders  
    
  import android.view.View  
  import androidx.recyclerview.widget.LinearLayoutManager  
  import androidx.recyclerview.widget.PagerSnapHelper  
  import androidx.recyclerview.widget.RecyclerView  
  import androidx.recyclerview.widget.SnapHelper  
  import com.google.android.material.tabs.TabLayout  
  import com.nehak.instagramfeed.databinding.HorizontalItemFeedBinding  
  import com.nehak.instagramfeed.feedUI.adapters.ImageAdapter.Companion.*TOTAL\_IMAGES  
    
  /\*\*  
   \* Create By SUBHANKAR, RAKESH, RAVI  
   \*/*class ImageFeedViewHolder(root: View) : FeedViewHolder(root) {  
   lateinit var recyclerViewImages: RecyclerView;  
    
   constructor(binding: HorizontalItemFeedBinding) : this(binding.*root*) {  
   recyclerViewImages =  
   binding.recyclerViewImages;  
    
   */\*\* Keep the item center aligned\*\*/* val snapHelper: SnapHelper = PagerSnapHelper();  
   snapHelper.attachToRecyclerView(recyclerViewImages);  
    
    
   */\*\*  
   \* Add dots (fixed size for now)  
   \*/* for (i in 0 *until TOTAL\_IMAGES*) {  
   binding.dots.addTab(binding.dots.newTab())  
   }  
    
   recyclerViewImages.addOnScrollListener(object : RecyclerView.OnScrollListener() {  
   override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {  
   super.onScrolled(recyclerView, dx, dy)  
   val itemPosition: Int =  
   (recyclerViewImages.*layoutManager* as LinearLayoutManager).findFirstCompletelyVisibleItemPosition()  
   try {  
   val tab: TabLayout.Tab? = binding.dots.getTabAt(itemPosition)  
   if (tab != null) tab.select()  
   } catch (e: Exception) {  
   e.printStackTrace()  
   }  
   }  
   })  
    
   }  
    
  }
* **ImageViewHolder**
* package com.nehak.instagramfeed.feedUI.holders  
    
  import androidx.annotation.NonNull  
  import androidx.recyclerview.widget.RecyclerView  
  import com.nehak.instagramfeed.databinding.ImageItemSingleBinding  
    
  */\*\*  
   \* Create By SUBHANKAR, RAKESH, RAVI  
   \*/*class ImageViewHolder(binding: ImageItemSingleBinding) : RecyclerView.ViewHolder(binding.*root*) {  
   val imageView =  
   binding.imageView;  
  }
* **VideoFeedViewHolder**
* package com.nehak.instagramfeed.feedUI.holders  
    
  import com.nehak.instagramfeed.databinding.ItemFeedBinding  
  import com.nehak.instagramfeed.feedUI.holders.FeedViewHolder  
    
  */\*\*  
   \* Create By SUBHANKAR, RAKESH, RAVI  
   \*/*class VideoFeedViewHolder(binding: ItemFeedBinding) : FeedViewHolder(binding.*root*) {  
   val videoThumbnail =  
   binding.feedThumbnailView;  
   val customPlayerView =  
   binding.feedPlayerView;  
    
  }
* **FeedListFragment**
* package com.nehak.instagramfeed.feedUI  
    
  import android.animation.Animator  
  import android.os.Bundle  
  import android.view.LayoutInflater  
  import android.view.View  
  import android.view.ViewGroup  
  import android.view.animation.AccelerateInterpolator  
  import android.view.animation.DecelerateInterpolator  
  import androidx.fragment.app.Fragment  
  import androidx.recyclerview.widget.RecyclerView  
  import com.nehak.instagramfeed.other.Extensions.Companion.findFirstVisibleItemPosition  
  import com.nehak.instagramfeed.other.Extensions.Companion.isAtTop  
  import com.nehak.instagramfeed.autoPlay.VideoAutoPlayHelper  
  import com.nehak.instagramfeed.databinding.FragmentFeedListBinding  
  import com.nehak.instagramfeed.feedUI.adapters.FeedAdapter  
  import com.nehak.instagramfeed.other.Lg  
    
  */\*\*  
   \* Create By SUBHANKAR, RAKESH, RAVI  
   \*/*class FeedListFragment : Fragment() {  
    
   private var controlsVisibleShowHide: Boolean = false;  
   private val HIDE\_THRESHOLD = 100;  
   private var isHeaderAlreadyHidden = false;  
   lateinit var binding: FragmentFeedListBinding;  
   private var scrolledDistance: Int = 0;  
    
   override fun onCreateView(  
   inflater: LayoutInflater,  
   container: ViewGroup?,  
   savedInstanceState: Bundle?  
   ): View {  
   binding = FragmentFeedListBinding.inflate(inflater, container, false)  
   return binding.*root*;  
   }  
    
   override fun onActivityCreated(savedInstanceState: Bundle?) {  
   super.onActivityCreated(savedInstanceState)  
    
   /\* Set adapter (items are being used inside adapter, you can setup in your own way\*/  
   val feedAdapter = FeedAdapter(requireContext())  
   binding.*adapter* = feedAdapter  
    
   /\*Helper class to provide AutoPlay feature inside cell\*/  
   val videoAutoPlayHelper =  
   VideoAutoPlayHelper(recyclerView = binding.recyclerView)  
   videoAutoPlayHelper.startObserving();  
    
   /\*Helper class to provide show/hide toolBar\*/  
   attachScrollControlListener(binding.customToolBar, binding.recyclerView)  
   }  
    
    
   */\*\*  
   \* This method will show hide view passed as @param -toolBar  
   \*/* fun attachScrollControlListener(toolBar: View?, recyclerView: RecyclerView) {  
    
   recyclerView.addOnScrollListener(object : RecyclerView.OnScrollListener() {  
   override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {  
   super.onScrolled(recyclerView, dx, dy)  
    
    
   var firstVisibleItem = -1  
   try {  
   firstVisibleItem = recyclerView.*findFirstVisibleItemPosition*()  
   } catch (e: Exception) {  
   Lg.printStackTrace(e)  
   }  
    
   if (firstVisibleItem == -1) {  
   return  
   }  
    
   //show views if first item is first visible position and views are hidden  
    
   if (firstVisibleItem == 0 && recyclerView.computeVerticalScrollOffset() < HIDE\_THRESHOLD) {  
   if (!controlsVisibleShowHide) {  
   controlsVisibleShowHide = true  
   showTopBarWithAnim(toolBar, recyclerView, true, null)  
   scrolledDistance = 0  
   }  
   } else {  
   if (scrolledDistance < -HIDE\_THRESHOLD && !controlsVisibleShowHide) {  
   controlsVisibleShowHide = true  
   showTopBarWithAnim(toolBar, recyclerView, true, null)  
   scrolledDistance = 0  
   } else if (dy > 0/\* && hideForcefully()\*/ || scrolledDistance > HIDE\_THRESHOLD && controlsVisibleShowHide) {  
   controlsVisibleShowHide = false  
   showTopBarWithAnim(toolBar, recyclerView, false, null)  
   scrolledDistance = 0  
   }  
   }  
    
   if (controlsVisibleShowHide && dy > 0 || !controlsVisibleShowHide && dy < 0) {  
   scrolledDistance += dy  
   }  
    
   }  
   })  
    
   }  
    
    
   */\*\*\*  
   \* Animation to show/hide  
   \*/* fun showTopBarWithAnim(  
   toolBar: View?,  
   recyclerView: RecyclerView,  
   show: Boolean,  
   animationListener: Animator.AnimatorListener?  
   ) {  
   if (show) {  
   if (!isHeaderAlreadyHidden) {  
   return  
   }  
   isHeaderAlreadyHidden = false  
   toolBar?.animate()?.translationY(0f)  
   ?.setInterpolator(DecelerateInterpolator(2f))  
   } else {  
   // To check if at the top of recycler view  
   if (recyclerView.*isAtTop*()  
   ) {  
   // Its at top  
   return  
   }  
   if (isHeaderAlreadyHidden) {  
   return  
   }  
   isHeaderAlreadyHidden = true  
   toolBar?.animate()  
   ?.translationY((-toolBar?.getHeight()!!).toFloat())  
   ?.setInterpolator(AccelerateInterpolator(2F))  
    
   }  
   }  
  }
* **player**
* **InstaLikePlayerView**
* /\*  
   \* Copyright (C) 2016 The Android Open Source Project  
   \*  
   \* Licensed under the Apache License, Version 2.0 (the "License");  
   \* you may not use this file except in compliance with the License.  
   \* You may obtain a copy of the License at  
   \*  
   \* http://www.apache.org/licenses/LICENSE-2.0  
   \*  
   \* Unless required by applicable law or agreed to in writing, software  
   \* distributed under the License is distributed on an "AS IS" BASIS,  
   \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
   \* See the License for the specific language governing permissions and  
   \* limitations under the License.  
   \*/  
  package com.nehak.instagramfeed.player  
    
  import android.annotation.SuppressLint  
  import android.content.Context  
  import android.graphics.Bitmap  
  import android.graphics.Matrix  
  import android.graphics.RectF  
  import android.net.Uri  
  import android.os.Handler  
  import android.os.HandlerThread  
  import android.os.Looper  
  import android.util.AttributeSet  
  import android.view.\*  
  import android.widget.FrameLayout  
  import android.widget.ImageView  
  import com.google.android.exoplayer2.C  
  import com.google.android.exoplayer2.Player  
  import com.google.android.exoplayer2.SimpleExoPlayer  
  import com.google.android.exoplayer2.Timeline  
  import com.google.android.exoplayer2.source.ProgressiveMediaSource  
  import com.google.android.exoplayer2.source.TrackGroupArray  
  import com.google.android.exoplayer2.source.ads.AdsLoader.AdViewProvider  
  import com.google.android.exoplayer2.text.Cue  
  import com.google.android.exoplayer2.text.TextOutput  
  import com.google.android.exoplayer2.trackselection.TrackSelectionArray  
  import com.google.android.exoplayer2.ui.PlayerControlView  
  import com.google.android.exoplayer2.ui.spherical.SingleTapListener  
  import com.google.android.exoplayer2.upstream.DefaultHttpDataSourceFactory  
  import com.google.android.exoplayer2.upstream.cache.CacheDataSource  
  import com.google.android.exoplayer2.upstream.cache.CacheDataSourceFactory  
  import com.google.android.exoplayer2.util.Assertions  
  import com.google.android.exoplayer2.util.Util  
  import com.google.android.exoplayer2.video.VideoListener  
  import com.nehak.instagramfeed.R  
  import com.nehak.instagramfeed.other.Constants  
  import kotlinx.coroutines.Dispatchers  
  import kotlinx.coroutines.GlobalScope  
  import kotlinx.coroutines.launch  
    
  class InstaLikePlayerView @JvmOverloads constructor(  
   context: Context?,  
   attrs: AttributeSet? = /\* attrs= \*/null,  
   defStyleAttr: Int = /\* defStyleAttr= \*/0  
  ) : FrameLayout(  
   context!!, attrs, defStyleAttr  
  ), AdViewProvider {  
   var videoSurfaceView: View?  
   private var player: Player? = null  
   private var textureViewRotation = 0  
   private var isTouching = false  
    
   */\*\*  
   \* Returns the player currently set on this view, or null if no player is set.  
   \*/* fun getPlayer(): Player? {  
   return player  
   }  
    
   */\*\*  
   \* Set the [Player] to use.  
   \*  
   \*  
   \* To transition a [Player] from targeting one view to another, it's recommended to use  
   \* [.switchTargetView] rather than this method. If you do  
   \* wish to use this method directly, be sure to attach the player to the new view \*before\*  
   \* calling `setPlayer(null)` to detach it from the old one. This ordering is significantly  
   \* more efficient and may allow for more seamless transitions.  
   \*  
   \** ***@param*** *player The [Player] to use, or `null` to detach the current player. Only  
   \* players which are accessed on the main thread are supported (`player.getApplicationLooper() == Looper.getMainLooper()`).  
   \*/* fun setPlayer(player: Player?) {  
   Assertions.checkState(Looper.myLooper() == Looper.getMainLooper())  
   Assertions.checkArgument(  
   player == null || player.*applicationLooper* == Looper.getMainLooper()  
   )  
   if (this.player === player) {  
   return  
   }  
   val oldPlayer = this.player  
   if (oldPlayer != null) {  
   val oldVideoComponent = oldPlayer.*videoComponent* if (oldVideoComponent != null) {  
   oldVideoComponent.clearVideoSurfaceView(videoSurfaceView as SurfaceView?)  
   }  
   }  
   this.player = player  
   if (player != null) {  
   val newVideoComponent = player.*videoComponent* if (newVideoComponent != null) {  
   newVideoComponent.setVideoSurfaceView(videoSurfaceView as SurfaceView?)  
   }  
   } else {  
   }  
   }  
    
   override fun setVisibility(visibility: Int) {  
   super.setVisibility(visibility)  
    
   // Work around https://github.com/google/ExoPlayer/issues/3160.  
   videoSurfaceView?.setVisibility(visibility)  
    
   }  
    
   override fun dispatchKeyEvent(event: KeyEvent): Boolean {  
   if (player != null && player!!.*isPlayingAd*) {  
   return super.dispatchKeyEvent(event)  
   }  
   val isDpadKey = isDpadKey(event.*keyCode*)  
   return false  
   }  
    
   override fun onTouchEvent(event: MotionEvent): Boolean {  
   return if (player == null) {  
   false  
   } else when (event.*action*) {  
   MotionEvent.*ACTION\_DOWN* -> {  
   isTouching = true  
   true  
   }  
   MotionEvent.*ACTION\_UP* -> {  
   if (isTouching) {  
   isTouching = false  
   performClick()  
   return true  
   }  
   false  
   }  
   else -> false  
   }  
   }  
    
   override fun performClick(): Boolean {  
   super.performClick()  
   return false  
   }  
    
   override fun onTrackballEvent(ev: MotionEvent): Boolean {  
   return false  
   }  
    
   override fun getAdViewGroup(): ViewGroup? {  
   return null  
   }  
    
   override fun getAdOverlayViews(): Array<View?> {  
   return *arrayOfNulls*(0)  
   }  
    
   @SuppressLint("InlinedApi")  
   private fun isDpadKey(keyCode: Int): Boolean {  
   return keyCode == KeyEvent.*KEYCODE\_DPAD\_UP* || keyCode == KeyEvent.*KEYCODE\_DPAD\_UP\_RIGHT* || keyCode == KeyEvent.*KEYCODE\_DPAD\_RIGHT* || keyCode == KeyEvent.*KEYCODE\_DPAD\_DOWN\_RIGHT* || keyCode == KeyEvent.*KEYCODE\_DPAD\_DOWN* || keyCode == KeyEvent.*KEYCODE\_DPAD\_DOWN\_LEFT* || keyCode == KeyEvent.*KEYCODE\_DPAD\_LEFT* || keyCode == KeyEvent.*KEYCODE\_DPAD\_UP\_LEFT* || keyCode == KeyEvent.*KEYCODE\_DPAD\_CENTER* }  
    
   init {  
   if (*isInEditMode*) {  
   videoSurfaceView = null  
    
   } else {  
   val playerLayoutId = R.layout.*exo\_simple\_player\_view* LayoutInflater.from(context).inflate(playerLayoutId, this)  
   *descendantFocusability* = *FOCUS\_AFTER\_DESCENDANTS* // Content frame.  
   videoSurfaceView = findViewById(R.id.*surface\_view*)  
   init()  
   }  
   }  
    
   private var lastPos: Long? = 0  
   private var videoUri: Uri? = null;  
    
   var cacheDataSourceFactory = CacheDataSourceFactory(  
   Constants.simpleCache,  
   DefaultHttpDataSourceFactory(  
   Util.getUserAgent(  
   context!!, context.getString(  
   R.string.*app\_name* )  
   )  
   ),  
   CacheDataSource.*FLAG\_IGNORE\_CACHE\_ON\_ERROR* )  
    
   fun init() {  
   reset()  
    
   /\*Setup player + Adding Cache Directory\*/  
   val simpleExoPlayer = SimpleExoPlayer.Builder(*context*).build()  
   simpleExoPlayer.*repeatMode* = Player.*REPEAT\_MODE\_ONE*;  
   simpleExoPlayer.addListener(object : Player.EventListener {  
   override fun onPlayerStateChanged(playWhenReady: Boolean, playbackState: Int) {  
   super.onPlayerStateChanged(playWhenReady, playbackState)  
   if (playbackState == Player.*STATE\_READY*) {  
   simpleExoPlayer.seekTo(lastPos!!)  
   *alpha* = 1f  
    
   }  
   }  
    
   })  
    
   simpleExoPlayer.*playWhenReady* = false  
   setPlayer(simpleExoPlayer);  
    
   }  
    
   */\*\*  
   \* This will resuse the player and will play new URI we have provided  
   \*/* fun startPlaying() {  
    
   val mediaSource =  
   ProgressiveMediaSource.Factory(cacheDataSourceFactory)  
   .createMediaSource(videoUri)  
   (player as SimpleExoPlayer).prepare(mediaSource)  
    
   player?.seekTo(lastPos!!)  
   player?.*playWhenReady* = true  
    
    
   }  
    
   */\*\*  
   \* This will stop the player, but stopping the player shows blackscreen  
   \* so to cover that we set alpha to 0 of player  
   \* and lastFrame of player using imageView over player to make it look like paused player  
   \*  
   \* (If we will not stop the player, only pause , then it can cause memory issue due to overload of player  
   \* and paused player can not be payed with new URL, after stopping the player we can reuse that with new URL  
   \*  
   \*/* fun removePlayer() {  
   getPlayer()?.setPlayWhenReady(false)  
   lastPos = getPlayer()?.*currentPosition* reset()  
   getPlayer()?.stop(true)  
    
   }  
    
   fun reset() {  
   // This will prevent surface view to show black screen,  
   // and we will make it visible when it will be loaded  
   *alpha* = 0f  
   }  
    
   fun setVideoUri(uri: Uri?) {  
   this.videoUri = uri;  
   }  
  }
* **MainActivity.kt**
* package com.nehak.instagramfeed  
    
  import android.content.Intent  
  import android.os.Bundle  
  import androidx.databinding.DataBindingUtil  
  import androidx.fragment.app.FragmentActivity  
  import com.nehak.instagramfeed.autoPlay.VideoPreLoadingService  
  import com.nehak.instagramfeed.dataModels.FeedItem  
  import com.nehak.instagramfeed.databinding.ActivityMainBinding  
  import com.nehak.instagramfeed.other.Constants  
  import java.util.\*  
    
  class MainActivity : FragmentActivity() {  
    
   override fun onCreate(savedInstanceState: Bundle?) {  
   super.onCreate(savedInstanceState)  
   DataBindingUtil.setContentView<ActivityMainBinding>(this, R.layout.*activity\_main*)  
    
   startPreCachingOfVideo()  
    
   }  
    
   */\*\*  
   \* This method will start service to preCache videos from remoteUrl in to Cache Directory  
   \* So the Player will not reload videos from server if they are already loaded in cache  
   \*/* fun startPreCachingOfVideo() {  
    
   val preloadingServiceIntent = Intent(this, VideoPreLoadingService::class.*java*)  
   preloadingServiceIntent.putStringArrayListExtra(  
   Constants.VIDEO\_LIST,  
   Constants.videoList.*toStringArray*()  
   )  
   startService(preloadingServiceIntent)  
    
   }  
  }  
    
  private fun <E> ArrayList<E>.toStringArray(): ArrayList<String> {  
    
   val arr = ArrayList<String>();  
   *forEach* **{** arr.add((**it** as FeedItem).downloadUrl!!);  
   **}** return arr;  
    
  }
* **MainApplication**
* package com.nehak.instagramfeed  
    
  import android.app.Application  
  import com.google.android.exoplayer2.database.ExoDatabaseProvider  
  import com.google.android.exoplayer2.upstream.cache.LeastRecentlyUsedCacheEvictor  
  import com.google.android.exoplayer2.upstream.cache.SimpleCache  
  import com.nehak.instagramfeed.other.Constants.Companion.*simpleCache*import java.io.File  
    
  class MainApplication : Application() {  
    
   override fun onCreate() {  
   super.onCreate()  
   setUpForPreCaching()  
   }  
    
    
   fun setUpForPreCaching() {  
    
   val exoPlayerCacheSize = 50 \* 1024 \* 1024.toLong()// Set the size of cache for video  
   var leastRecentlyUsedCacheEvictor: LeastRecentlyUsedCacheEvictor? = null  
   var exoDatabaseProvider: ExoDatabaseProvider? = null  
    
   if (leastRecentlyUsedCacheEvictor == null) {  
   leastRecentlyUsedCacheEvictor = LeastRecentlyUsedCacheEvictor(exoPlayerCacheSize)  
   }  
    
   if (exoDatabaseProvider != null) {  
   exoDatabaseProvider = ExoDatabaseProvider(this)  
   }  
    
   if (*simpleCache* == null) {  
   val cache: File = File(getCacheDir(), "Video\_Cache")  
   if (!cache.exists()) {  
   cache.mkdirs()  
   }  
   *simpleCache* =  
   SimpleCache(cache, leastRecentlyUsedCacheEvictor, exoDatabaseProvider)  
   }  
    
   }  
    
  }