[EXTERNAL EXPORT] AdViewManager

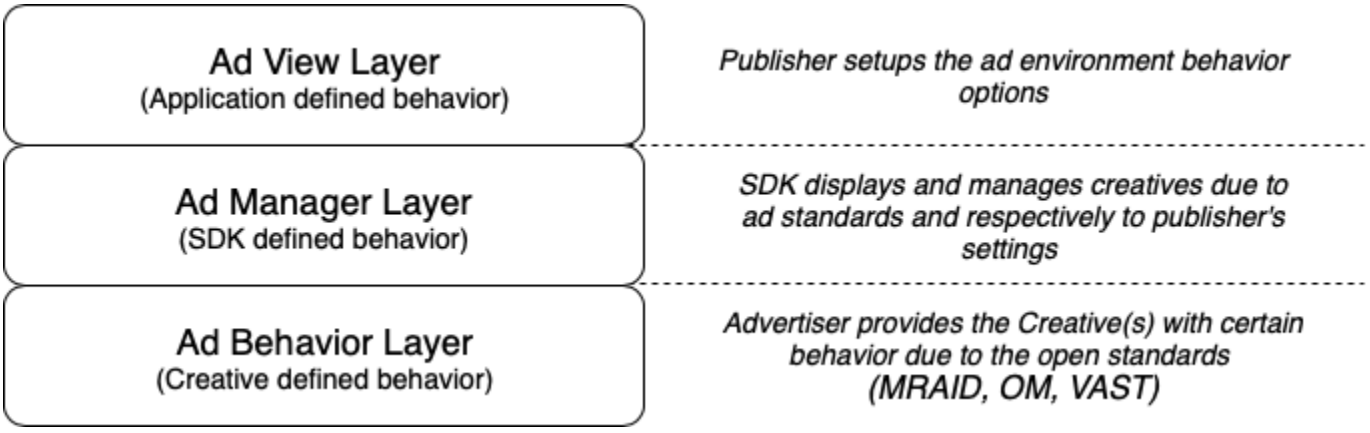
Motivation

The current flow of managing ads contains one crucial lack - a lot of objects without strictly defined responsibilities and as a result a high coupling between these objects.
In result, the code does not reflect the Domain properly. To avoid this we have to develop a transparent architecture corresponded to SOLID principles. In general, we already have all need objects, but we need to rearrange their responsibilities and decouple dependencies.

All text below describes the purpose of the ad manager and its usage that should be implemented in the same way on both platforms iOS and Android.

Vertical Architecture

To achieve robust architecture we have to apply the responsibilities of each entity to one of three domain layers. No one entity should act on several layers directly. it could delegate some responsibilities.



Ad View Layer contains objects for displaying ads f.e BannerView, VideoAdView, InterstitialController (ModalController as a display part). The Ad View layer is **responsible** for supporting Application flow.

Ad Manager Layer contains objects that responsible for the transition of the ads from one state to another or from one view to another. The responsibility of these actions lies on AdViewManager. Also, ModalManager and TransactionsManager are a part of this layer. The Manager layer is **responsible** for supporting SDK features and flows (display layout, AdChoices button, Close Button, Learn More button, Clickthrough browser).

Ad Behaviour Layer contains objects responsible for ad behavior despite the presentation style. In fact, we are talking about Creatives here: VideoCreative HTMLCreative. The Ad Behavior layer is **responsible** for supporting features and flows defined by creatives content provided by publishers.

The AdViewManager takes the central part in this process. Only it knows how to show, load, and transfer particular Creative in particular View. However, the AdViewManager is rather about the concept, not about implementation details because there are a lot of crucial differences between Creatives and its Views. And in a result, we have critical differences in the kind of how AdViewManager should work in a particular situation.

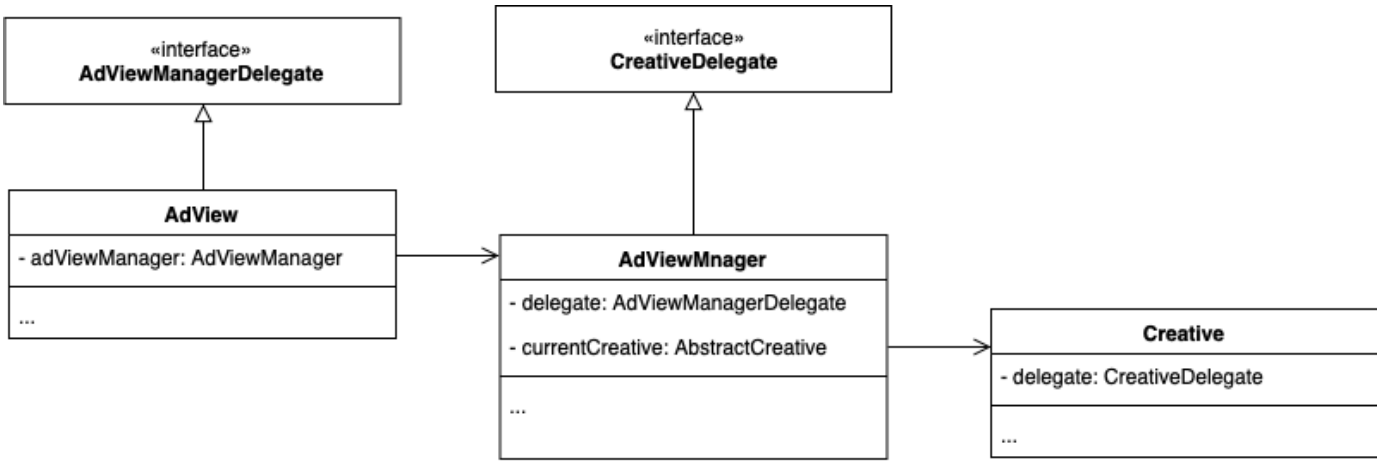
To understand what kind of control we need from AdViewManager for supported kinds of ads let's take a look at the chart with the relations between Views and Creative. In other words, let's take a look at how exactly certain creatives could be displayed.

	Banner	Interstitial	Banner and Interstitial	Non Modal Interstitial
Display Ad *HTMLCreative*	+ *Regular Banner*	+ *Regular Interstitial*	+ *MRAID Expand, MRAID Video Interstitial*	+ *MRAID Resize*
Video Ad *VideoCreative* *+ HTMLCreative*	-	+ *Regular Video*	+ *Video 300x250*	-

Now we can see that there are a lot of behavior kinds but not all of them are needed for a certain kind of ad. On the other hand, there is a pitfall thing that is not obvious at the first glance. In the case of HTML ad, we do not actually know whether it is an MRAID ad. So that we should be prepared for the fact that Banner ad could be displayed in the Interstitial or Non Modal mode on demand. However, it is not a reason to mix all logic together "just in case". Since we have on-demand behavior the implementation and architecture should be built with on-demand principles as well. Each specific behavior should be implemented by a specific class.

Classes and responsibilities

The important point is that **AdView** and **Creative** does not have references to each other. Only AdView Manager knows how to display Creative in the AdView.



AdView uses **AdViewManager** via public methods and properties. **AdViewManager** updates the **AdView** through **AdViewManagerDelegate** (**AdViewManagerListener**).

AdViewManager uses **Creative** via public methods and properties. **Creative** triggers **AdViewManager's** actions through **CreativeDelegate** (**CreativeListener**).

AdViewManagerDelegate

This protocol serves as a bridge between changes in the AdViewManger and AdView. Need to keep in mind that AdView should not know anything about the Creative or Creatives kind, so methods of this protocol should just inform about the kind of change without any specifics.

Method or Property	Description
viewControllerForModalPresentation: UIViewController	[iOS] The View controller that should be used for show modal content of the add
displayView: UIView	[iOS] The actual top layer view that displays the ad
adLoaded(adDetails: AdDetails) void	Is called when AdViewManager prepared the ad for displaying
failedToLoad(error: Error) void	Is called when AdViewManager is failed to load the ad
adDidComplete() void	Is called when ad is finished displaying
adDidDisplay() void	Is called when ad appeared in the display view
adDidClose() void	Is called when an ad was closed
adWasClicked() void	Is called when user tapped on the creative to see open the content
adViewWasClicked() void	Is called when user tapped on the creative's view
adDidLeaveApp() void	Is called when click on the ad leads to opening content in the external browser
adClickthroughDidClose() void	Is called when a user closes the ad's clickthrough.
adDidExpand() void	Is called when an MRAID ad is expanded
adDidCollapse() void	Is called when an MRAID ad is collapsed

All these methods and properties are common for all kinds of ads. If you need to add some specific behavior for a certain ad kind need to create a new protocol inherit from this one and use it respectively.

CreativeDelegate

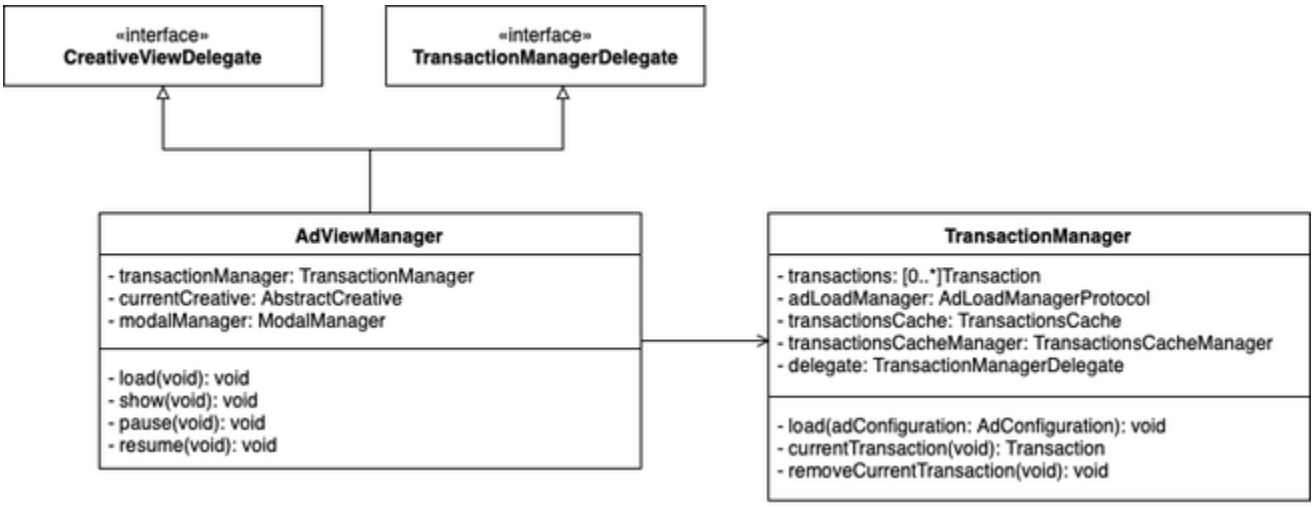
CreativeDelegate is a protocol responsible for informing **AdViewManager** about events in the **Creative** that require some actions.

Method	Description
creativeDidComplete(creative: OXMAbstractCreative) void	Is called when a creative finishes its lifecycle.
creativeDidDisplay(creative: OXMAbstractCreative *) void	Is called when a creative finishes its lifecycle.
creativeWasClicked(creative: OXMAbstractCreative *) void	Is called when ad's content is rendered in the displayed view.
creativeViewWasClicked(creative: AbstractCreative *) void	Is called when the user triggers the showing the ad content (clickthrough or Learn More)
creativeClickthroughDidClose(creative: AbstractCreative *) void	Is called when clickthrough is closed
creativeInterstitialDidClose(creative: AbstractCreative *) void	Is called when the user closes the creatives clickthrough.
creativeInterstitialDidLeaveApp(creative: AbstractCreative *) void	Is called when click on the ad leads to opening content in the external browser
creativeFullScreenDidFinish(creative: AbstractCreative) void	Is called when creatives back to the normal state from the fullscreen mode (VideoAdView)
creativeShouldDisplayModalState(state: ModalState, completion: ((void) void))	Is called when Creative due to its internal implementation should show content in the new Modal View

AdViewManager

Since each ad Is unique, it is hard to split the responsibilities of AdViewManger into certain parts depending on the ad kind. Hence AdViewManager will manage all of them at once but still as a high-level defined algorithm but not a special defined approach. The main criteria is

that AdViewManager should not be aware of the ad type (Video, Display) that it manages, only about ad kind (banner, interstitial). In other words, AdViewManager should rely only on Ad Configuration properties but not Creative types.



The AdViewManager responsibilities:

- Managing transactions by delegating this process to the Transaction Manager.
- Showing creatives into their respective views while serving as a middle-man between their own calls (is a **CreativeViewListener**)

Method	Description
load() void	creates the AdLoader and performs the loading

show() void	• displays the Ad (first creative)
	  • banner - in the AdView
	  • interstitial - In the modal view

pause() void	stops the creative execution (video ad)
resume() void	resumes the creative execution (video ad)

TransactionManager

The aims of TransactionManager are:

- Take transaction from the cache (video)
- Load the transaction
- Manage the transactions cache
- Inform the AdViewManager about transaction readiness
- Manage and provide current transaction

Note: the transaction cache was removed.

Method	Description
fetchTransaction(adConfiguration: AdConfiguration) void	Initiates the process of receiving the transaction. May include the stage of retrieving transaction from the transactions cache or direct loading from the web.
getCurrentTransaction() Transaction	Returns the transaction that should be displayed right now.
dismissCurrentTransaction() Transaction	Removes the current transaction from internal cache. Returns the next transactions in the list.

TransactionManagerDelegate

This interface is intended to inform the **AdViewManager** about the result of the transaction fetching.

Method	Description
onFetchingCompleted(transaction: Transaction. error: Error) void	Is called when **TransactionManager** has finished the fetching process. In case of failure, the **error** should be not null and contains the description of the issue. In case of success, the error should be null and the **transaction** represents the loaded transaction.

ModalManager

Modal Manger is a class dedicated to displaying the modal views with:

- **Interstitial Ads** (HTML Interstitial, Video Interstitial, Opt-InVideo, End Card, FullScreen Mode for 300x250 Video)
- **MRAID Ads** (Resize, Expand, Video);
- **Browsers** (Clickthrough, Ad Choices Legal page)

It might look like a god-class that should manage a huge amount of ad kinds, but it does not.

The purposes of the Modal Manager is pretty straightforward:

- to display the content view (ad or browser) in the interstitial view
- to manage content views related to the single source (f.e. MRAID Ad, interstitial ad with opened clickthrough)
- to dismiss the interstitial view
- to notify listeners about changing the state of interstitial presentation

So, the Modal Manager has nothing in common with the displayed content. It is just dedicated to managing the ad views in a particular manner. Modal Manager has a deal with States that should be represented in a particular order the FILO order.

The Modal State describes what needs to display and how exactly. Basically Modal state it is a structure that contains:

- view - that should be displayed
- InterstitialDisplayProperties - the additional properties for a particular ad view

Modal Manager puts each new state into the internal stack and shows the topmost. On dismissing the topmost state the next one should be displayed.

Methods for managing states:

Method	Description
push(modalState: ModalState) void	Adds the state to the stack and displays it as topmost one.
reset(modalState: ModalState) void	Replaces the topmost state with a provided one and displays it.
popState() void	Removes the topmost state and shows the new one if any. If the stack is empty closes interstitial.
clearStates() void	Removes all states closes interstitial.

ModalState

ModalState is a structure that contained all assets needed for displaying interstitial.

Property	Type	Description
displayView	View / UIView	The view that should be displayed in the interstitial.
viewControllerForModalPresentation	UIViewController	The view controller which should be used for presenting the interstitial
interstitialDisplayProperties	InterstitialDisplayProperties	Properties that applied exclusively to displaying the interstitial views.

modalManagerDelegate	ModalManagerDelegate	Creative represented via a protocol which should process the events from modal manager

InterstitialDisplayProperties

Property	Type	Description
contentFrame	CGFrame	Custom frame size for the interstitial ad. Used for MRAID Resize
interstitialLayout	InterstitialLayout	The orientation that should be applied to the particular ad
closeDelayLeft	Double	The amount of time left before the close button should become visible.