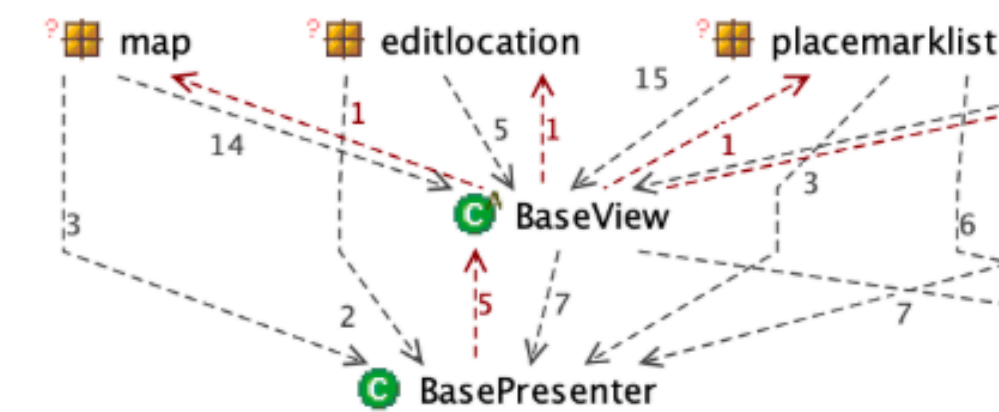


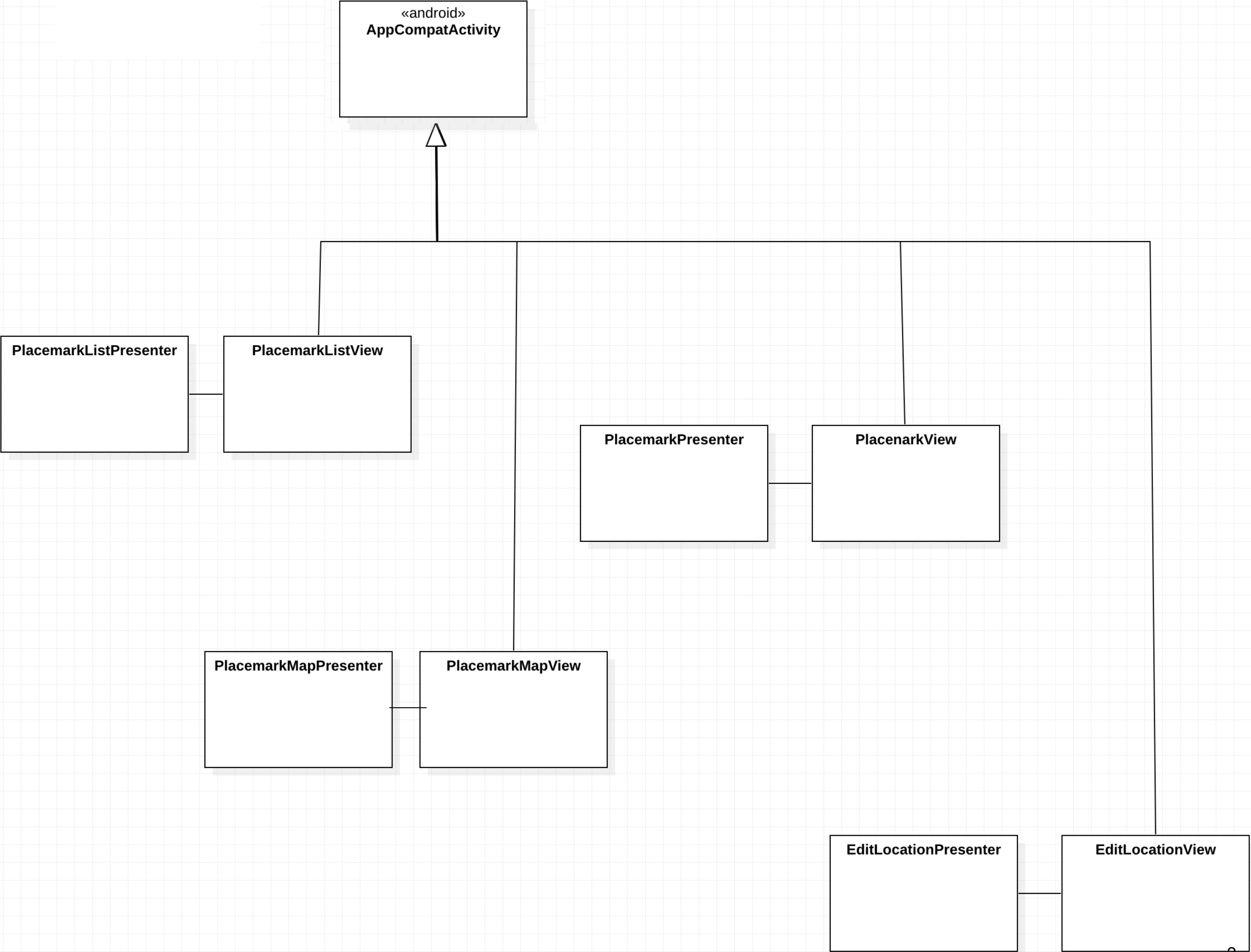
# BaseView BasePresenter

## BaseView BasePresenter

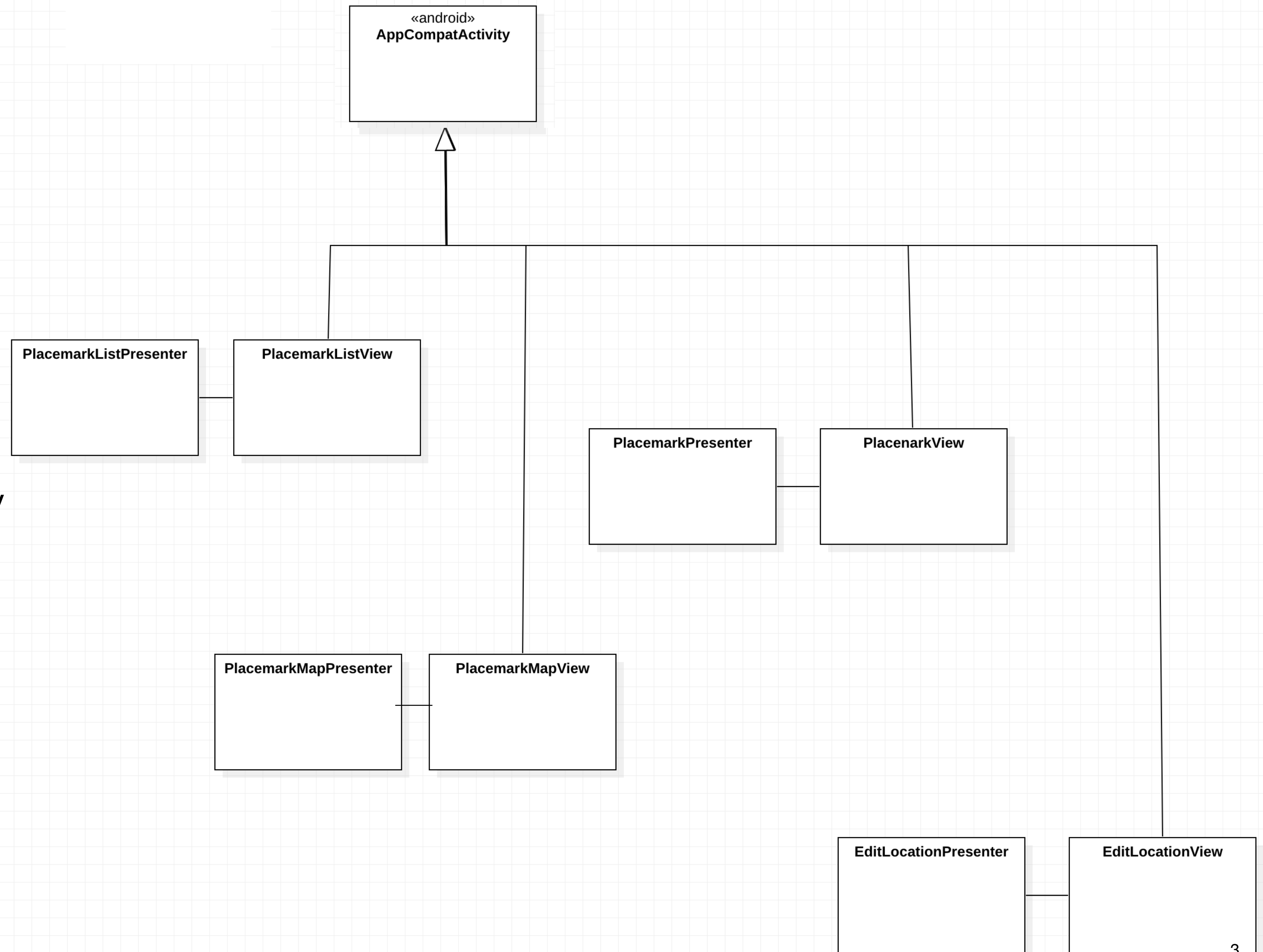


Factor out common MVP  
features into  
BaseView/Presenter classes

Current View/  
Presenter classes



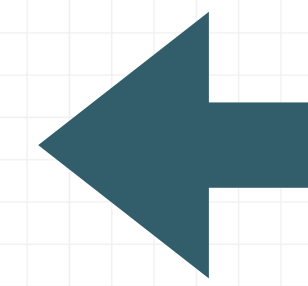
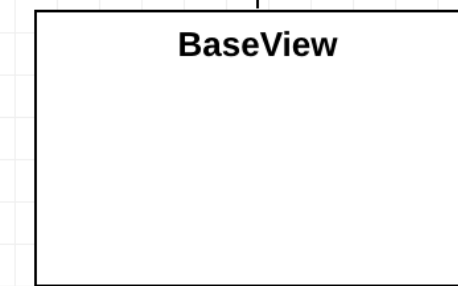
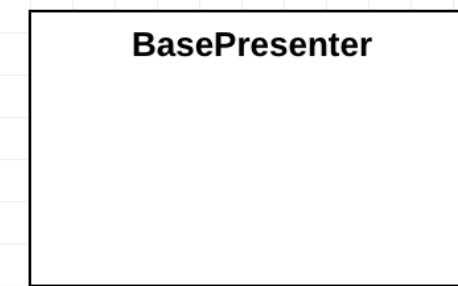
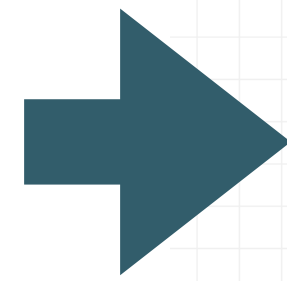
Current View/  
Presenter classes



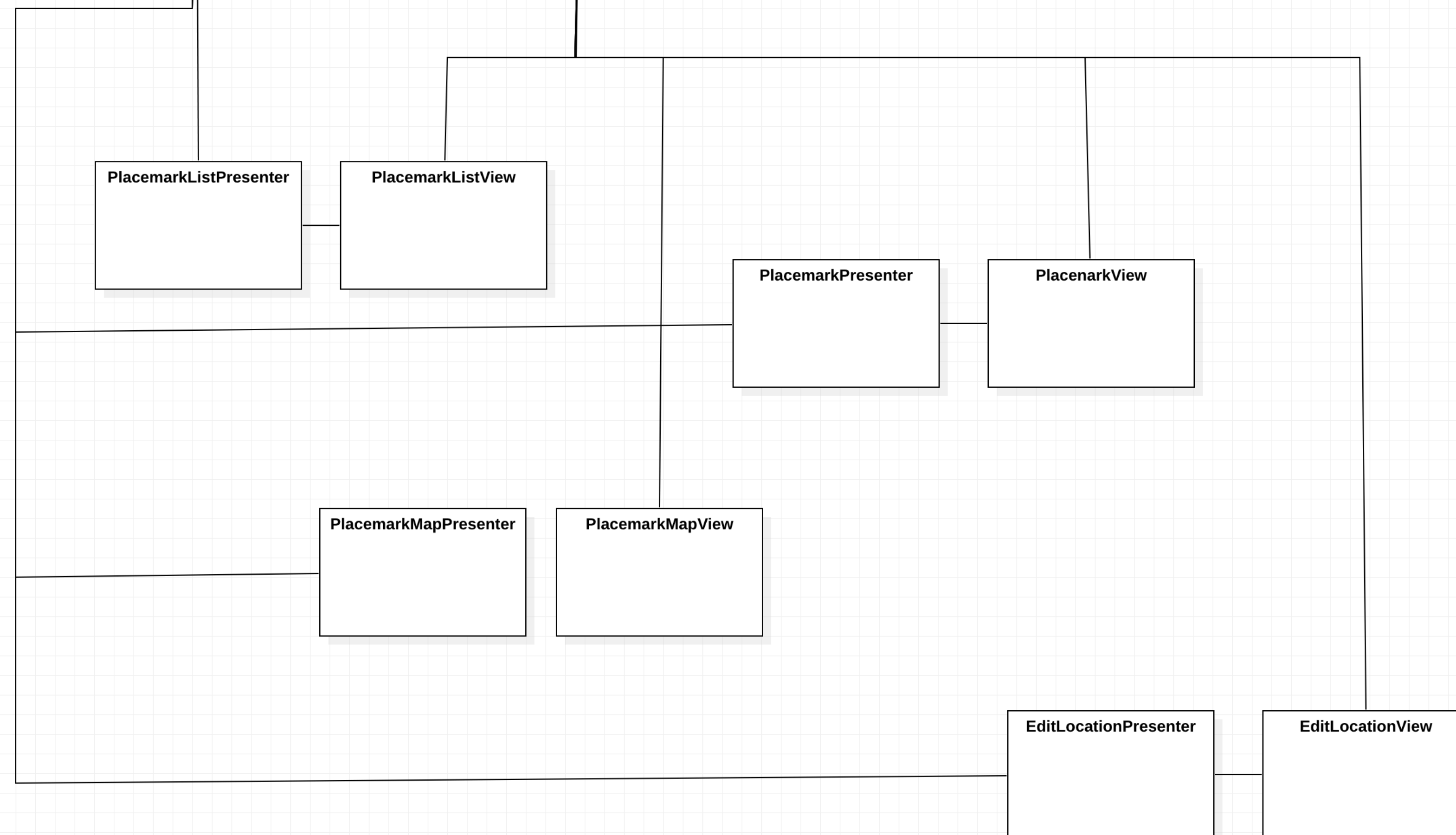
Potential commonality  
can be identified, and  
factored out into:

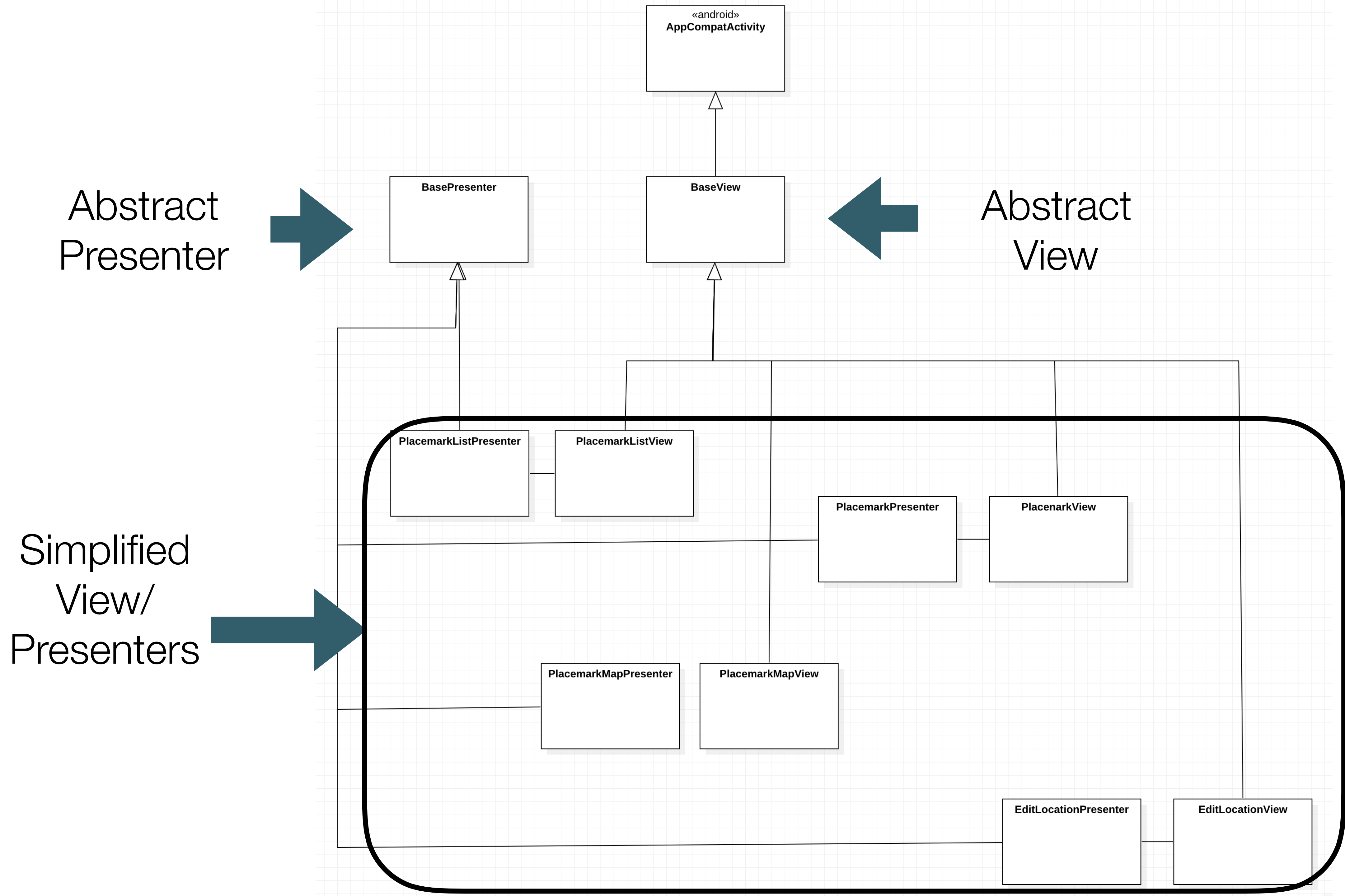
- BasePresenter
- BaseView

Abstract  
Presenter

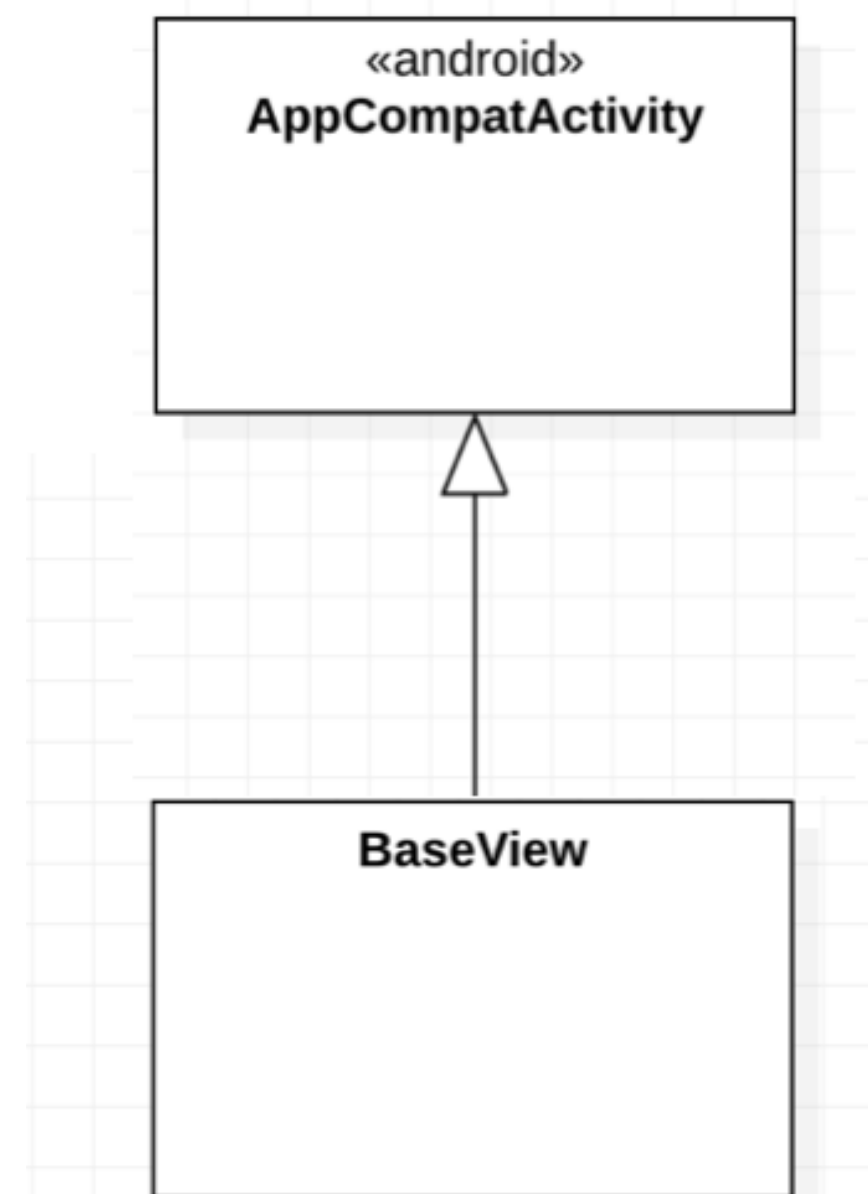


Abstract  
View





# BaseView



```
open abstract class BaseView() : AppCompatActivity(), AnkoLogger {

    var basePresenter: BasePresenter? = null

    fun navigateTo(view: VIEW, code: Int = 0, key: String = "", value: Parcelable? = null) {
        ...
    }

    fun initPresenter(presenter: BasePresenter): BasePresenter {
        ...
    }

    fun init(toolbar: Toolbar) {
        ...
    }

    override fun onDestroy() {
        ...
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        ...
    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>,
                                            grantResults: IntArray) {
        ...
    }

    ...
}

open fun showPlacemark(placemark: PlacemarkModel) {}
open fun showPlacemarks(placemarks: List<PlacemarkModel>) {}
open fun showProgress() {}
open fun hideProgress() {}
}
```

```
open abstract class BaseView() : AppCompatActivity(), AnkoLogger {

    var basePresenter: BasePresenter? = null

    fun navigateTo(view: VIEW, code: Int = 0, key: String = "", value: Parcelable? = null) {
        ...
    }

    fun initPresenter(presenter: BasePresenter): BasePresenter {
        ...
    }

    fun init(toolbar: Toolbar) {
        ...
    }

    override fun onDestroy() {
        ...
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        ...
    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>,
                                            grantResults: IntArray) {
        ...
    }

    open fun showPlacemark(placemark: PlacemarkModel) {}
    open fun showPlacemarks(placemarks: List<PlacemarkModel>) {}
    open fun showProgress() {}
    open fun hideProgress() {}
}
```

Centralise Intent/  
StartActivity for all views

```
open abstract class BaseView() : AppCompatActivity(), AnkoLogger {

    var basePresenter: BasePresenter? = null

    fun navigateTo(view: VIEW, code: Int = 0, key: String = "", value: Parcelable? = null) {
        ...
    }

    fun initPresenter(presenter: BasePresenter): BasePresenter {
        ...
    }

    fun init(toolbar: Toolbar) {
        ...
    }

    override fun onDestroy() {
        ...
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        ...
    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>,
                                            grantResults: IntArray) {
        ...
    }

    open fun showPlacemark(placemark: PlacemarkModel) {}
    open fun showPlacemarks(placemarks: List<PlacemarkModel>) {}
    open fun showProgress() {}
    open fun hideProgress() {}
}
```

Common initialisation



```
open abstract class BaseView() : AppCompatActivity(), AnkoLogger {

    var basePresenter: BasePresenter? = null

    fun navigateTo(view: VIEW, code: Int = 0, key: String = "", value: Parcelable? = null) {
        ...
    }

    fun initPresenter(presenter: BasePresenter): BasePresenter {
        ...
    }

    fun init(toolbar: Toolbar) {
        ...
    }

    override fun onDestroy() {
        ...
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        ...
    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>,
                                            grantResults: IntArray) {
        ...
    }

    open fun showPlacemark(placemark: PlacemarkModel) {}
    open fun showPlacemarks(placemarks: List<PlacemarkModel>) {}
    open fun showProgress() {}
    open fun hideProgress() {}
}
```

Common Lifecycle  
handling

```
open abstract class BaseView() : AppCompatActivity(), AnkoLogger {

    var basePresenter: BasePresenter? = null

    fun navigateTo(view: VIEW, code: Int = 0, key: String = "", value: Parcelable? = null) {
        ...
    }

    fun initPresenter(presenter: BasePresenter): BasePresenter {
        ...
    }

    fun init(toolbar: Toolbar) {
        ...
    }

    override fun onDestroy() {
        ...
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        ...
    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>,
                                           grantResults: IntArray) {
        ...
    }

    open fun showPlacemark(placemark: PlacemarkModel) {}
    open fun showPlacemarks(placemarks: List<PlacemarkModel>) {}
    open fun showProgress() {}
    open fun hideProgress() {}
}
```

Placeholders for individual  
view events of interest

<pre> open abstract class BaseView() : AppCompatActivity(), AnkoLogger {      var basePresenter: BasePresenter? = null      fun navigateTo(view: VIEW, code: Int = 0, key: String = "", value: Parcelable? = null) {         ...     }      fun initPresenter(presenter: BasePresenter): BasePresenter {         ...     }      fun init(toolbar: Toolbar) {         ...     }      override fun onDestroy() {         ...     }      override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {         ...     }      override fun onRequestPermissionsResult(requestCode: Int, permissions: Array&lt;String&gt;,  grantResults: IntArray) {         ...     }      open fun showPlacemark(placemark: PlacemarkModel) {}     open fun showPlacemarks(placemarks: List&lt;PlacemarkModel&gt;) {}     open fun showProgress() {}     open fun hideProgress() {} } </pre>	
<pre> fun navigateTo(view: VIEW, code: Int = 0, key: String = "", value: Parcelable? = null) {     ... } </pre>	Centralise Intent/ StartActivity for all views
<pre> fun initPresenter(presenter: BasePresenter): BasePresenter {     ... }  fun init(toolbar: Toolbar) {     ... } </pre>	Common initialisation
<pre> override fun onDestroy() {     ... }  override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {     ... }  override fun onRequestPermissionsResult(requestCode: Int, permissions: Array&lt;String&gt;,  grantResults: IntArray) {     ... } </pre>	Common Lifecycle handling
<pre> open fun showPlacemark(placemark: PlacemarkModel) {} open fun showPlacemarks(placemarks: List&lt;PlacemarkModel&gt;) {} open fun showProgress() {} open fun hideProgress() {} </pre>	Placeholders for individual view events of interest

# BasePresenter



**BasePresenter**

```
open class BasePresenter(var view: BaseView?) {  
  
    var app: MainApp = view?.application as MainApp  
  
    open fun doActivityResult(requestCode: Int, resultCode: Int, data: Intent) {  
    }  
  
    open fun doRequestPermissionsResult(requestCode: Int, permissions: Array<String>,  
                                         grantResults: IntArray) {  
    }  
  
    open fun onDestroy() {  
        view = null  
    }  
}
```

# BasePresenter

```
open class BasePresenter(var view: BaseView?) {  
    var app: MainApp = view?.application as MainApp  
  
    open fun doActivityResult(requestCode: Int, resultCode: Int, data: Intent) {  
    }  
  
    open fun doRequestPermissionsResult(requestCode: Int, permissions: Array<String>,  
                                        grantResults: IntArray) {  
    }  
  
    open fun onDestroy() {  
        view = null  
    }  
}
```

Common Lifecycle  
handling

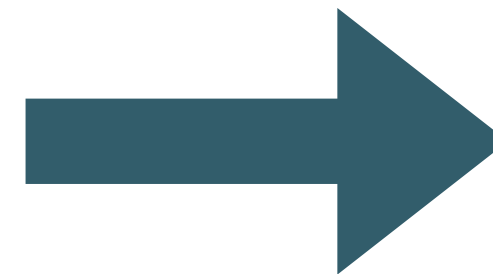
## onDestroy()

```
open abstract class BaseView() : AppCompatActivity(), AnkoLogger {  
    ...  
  
    override fun onDestroy() {  
        basePresenter?.onDestroy()  
        super.onDestroy()  
    }  
}
```

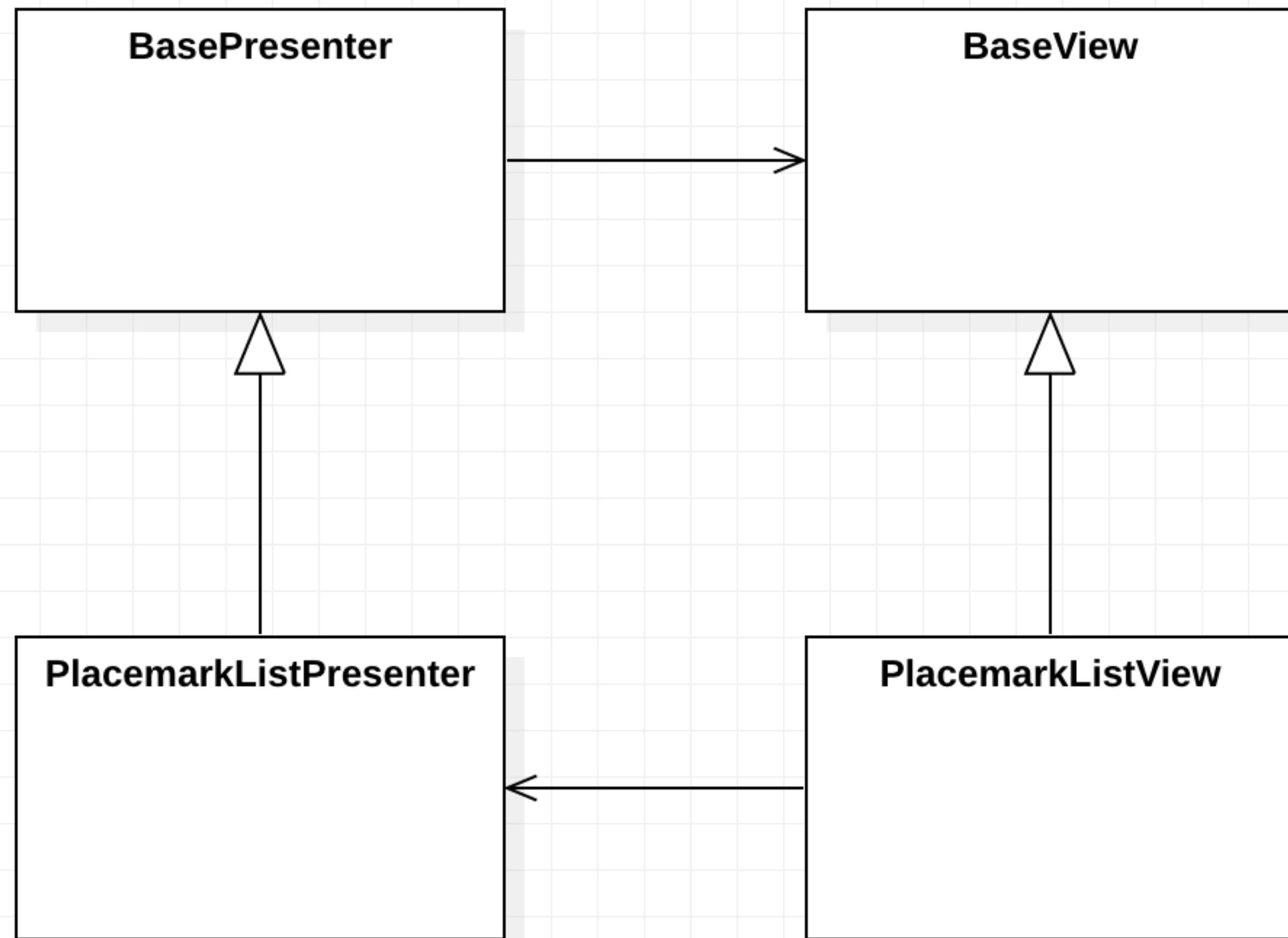
```
open class BasePresenter(var view: BaseView?) {  
    ...  
  
    open fun onDestroy() {  
        view = null  
    }  
}
```

Will be called when  
the Activity is being  
removed by Android

invoke destroy  
in presenter



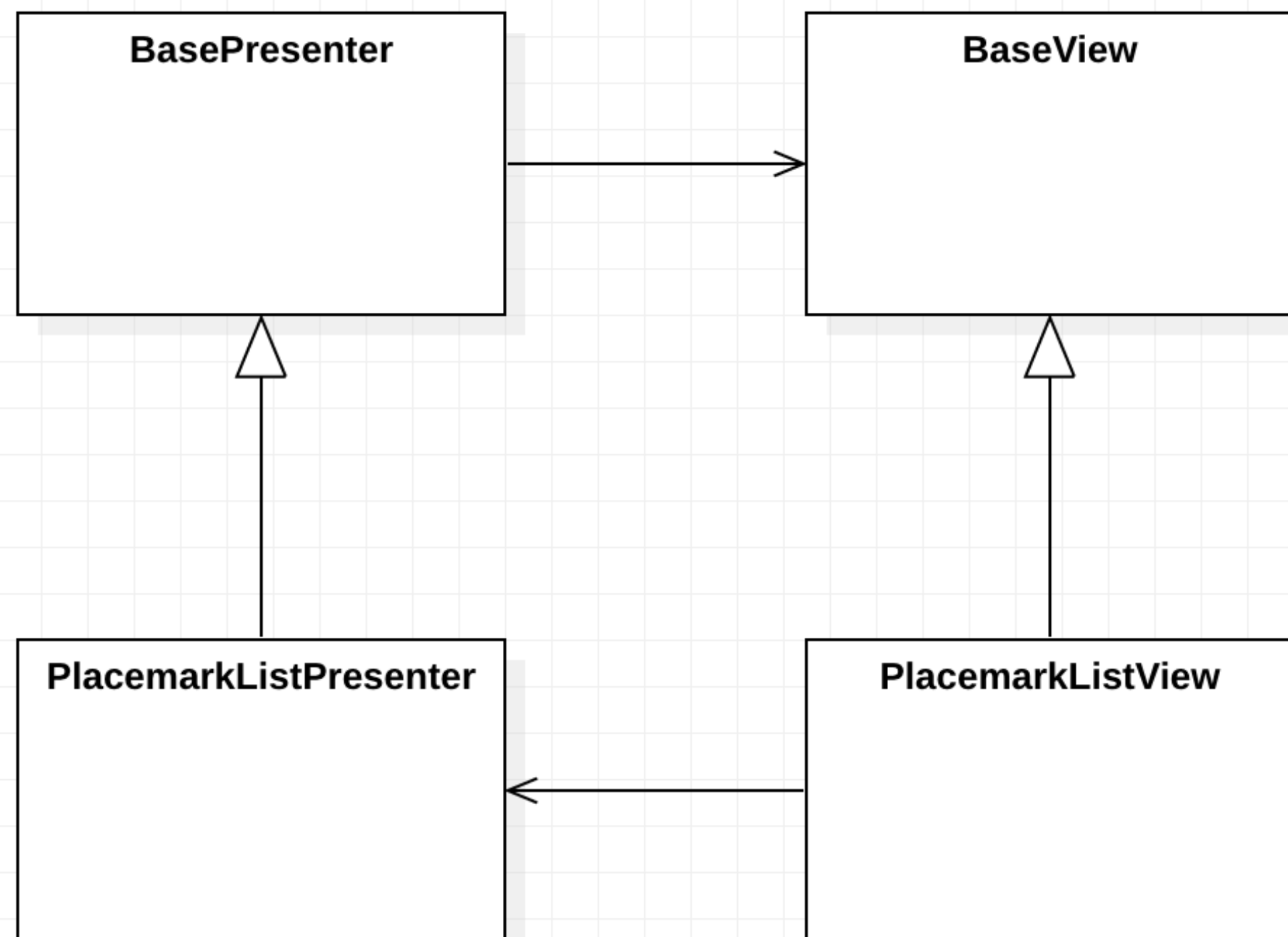
Set the view to null - to  
avoid potential null  
pointer violations



BasePresenter has  
reference to BaseView

PlacemarkListView has  
reference to  
PlacemarListPresenter





```

class PlacemarkListPresenter(view: BaseView) : BasePresenter(view) {

    fun doAddPlacemark() {
        view?.navigateTo(VIEW.PLACEMARK)
    }

    fun doEditPlacemark(placemark: PlacemarkModel) {
        view?.navigateTo(VIEW.PLACEMARK, 0, "placemark_edit", placemark)
    }

    fun doShowPlacemarksMap() {
        view?.navigateTo(VIEW.MAPS)
    }

    fun loadPlacemarks() {
        view?.showPlacemarks(app.placemarks.findAll())
    }
}

```

```

class PlacemarkListView : BaseView(), PlacemarkListener {

    lateinit var presenter: PlacemarkListPresenter

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_placemark_list)
        init(toolbarMain, false)

        presenter = initPresenter(PlacemarkListPresenter(this)) as PlacemarkListPresenter

        val layoutManager = LinearLayoutManager(this)
        recyclerView.layoutManager = layoutManager
        presenter.loadPlacemarks()
    }

    override fun showPlacemarks(placemarks: List<PlacemarkModel>) {
        recyclerView.adapter = PlacemarkAdapter(placemarks, this)
        recyclerView.adapter?.notifyDataSetChanged()
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.menu_main, menu)
        return super.onCreateOptionsMenu(menu)
    }

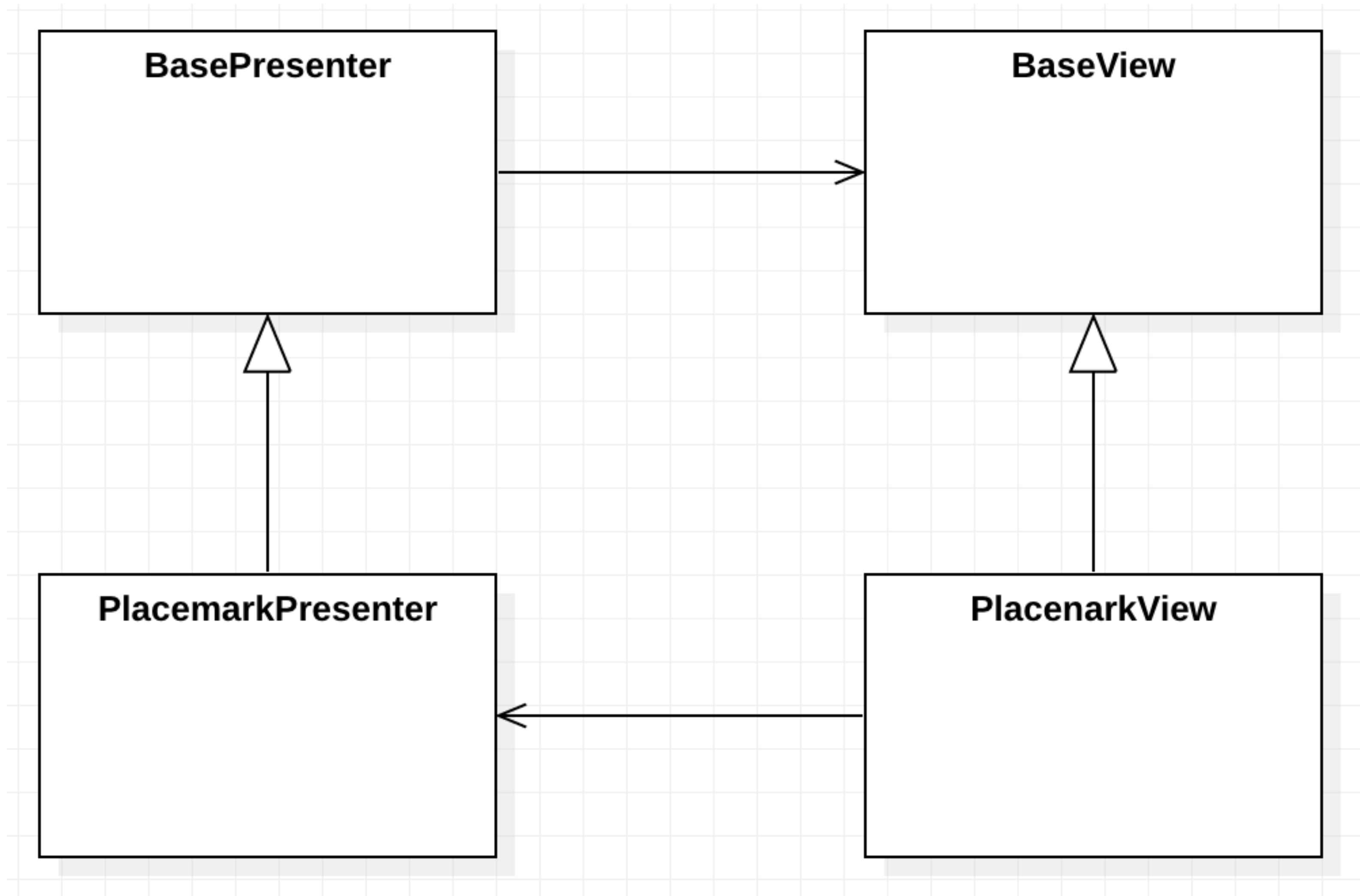
    override fun onOptionsItemSelected(item: MenuItem?): Boolean {
        when (item?.itemId) {
            R.id.item_add -> presenter.doAddPlacemark()
            R.id.item_map -> presenter.doShowPlacemarksMap()
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onPlacemarkClick(placemark: PlacemarkModel) {
        presenter.doEditPlacemark(placemark)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        presenter.loadPlacemarks()
        super.onActivityResult(requestCode, resultCode, data)
    }
}

```





```

class PlacemarkView : BaseView(), AnkoLogger {

    lateinit var presenter: PlacemarkPresenter
    var placemark = PlacemarkModel()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_placemark)

        init(toolbarAdd)

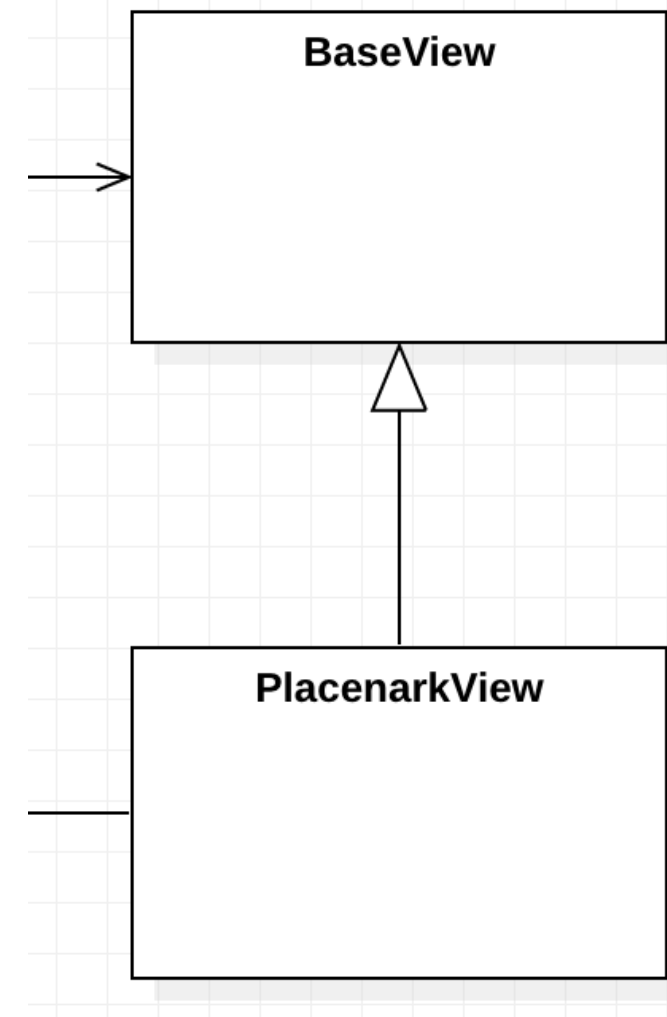
        presenter = initPresenter (PlacemarkPresenter(this)) as PlacemarkPresenter

        chooseImage.setOnClickListener { presenter.doSelectImage() }
        placemarkLocation.setOnClickListener { presenter.doSetLocation() }
    }

    override fun showPlacemark(placemark: PlacemarkModel) {
        placemarkTitle.setText(placemark.title)
        description.setText(placemark.description)
        placemarkImage.setImageBitmap(readImageFromPath(this, placemark.image))
        if (placemark.image != null) {
            chooseImage.setText(R.string.change_placemark_image)
        }
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.menu_placemark, menu)
        return super.onCreateOptionsMenu(menu)
    }
}

```



```

    override fun onOptionsItemSelected(item: MenuItem?): Boolean {
        when (item?.itemId) {
            R.id.item_delete -> {
                presenter.doDelete()
            }
            R.id.item_save -> {
                if (placemarkTitle.text.toString().isEmpty()) {
                    toast(R.string.enter_placemark_title)
                } else {
                    presenter.doAddOrSave(placemarkTitle.text.toString(), description.text.toString())
                }
            }
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (data != null) {
            presenter.doActivityResult(requestCode, resultCode, data)
        }
    }

    override fun onBackPressed() {
        presenter.doCancel()
    }
}

```

```
class PlacemarkPresenter(view: BaseView) : BasePresenter(view) {

    var placemark = PlacemarkModel()
    var defaultLocation = Location(52.245696, -7.139102, 15f)
    var edit = false;

    init {
        if (view.intent.hasExtra("placemark_edit")) {
            edit = true
            placemark = view.intent.extras?.getParcelable<PlacemarkModel>("placemark_edit")!!
            view.showPlacemark(placemark)
        }
    }

    fun doAddOrSave(title: String, description: String) {
        placemark.title = title
        placemark.description = description
        if (edit) {
            app.placemarks.update(placemark)
        } else {
            app.placemarks.create(placemark)
        }
        view?.finish()
    }

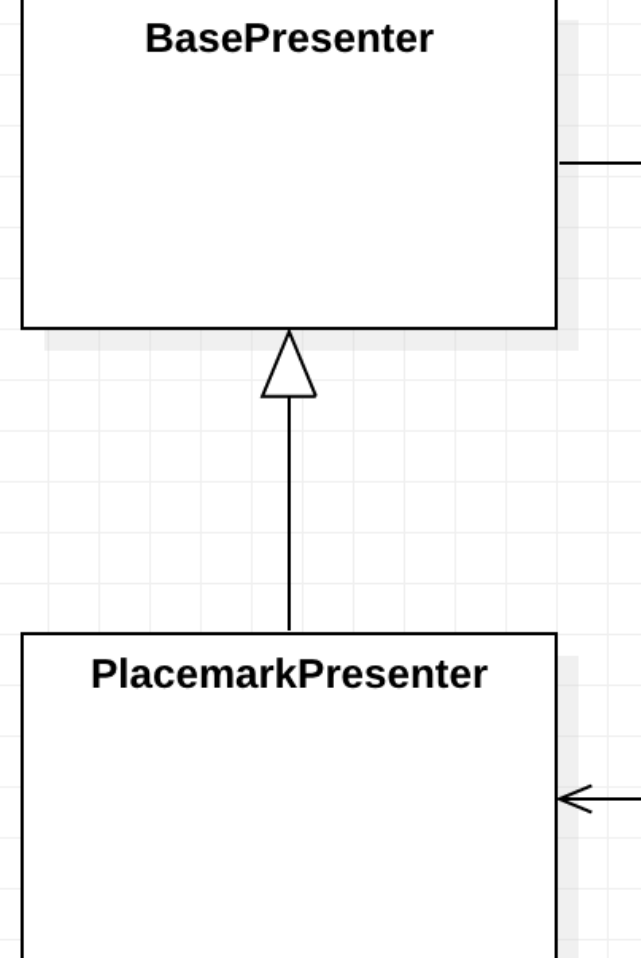
    fun doCancel() {
        view?.finish()
    }

    fun doDelete() {
        app.placemarks.delete(placemark)
        view?.finish()
    }

    fun doSelectImage() {
        view?.let {
            showImagePicker(view!!, IMAGE_REQUEST)
        }
    }
}
```

```
fun doSetLocation() {
    if (edit == false) {
        view?.navigateTo(VIEW.LOCATION, LOCATION_REQUEST, "location", defaultLocation)
    } else {
        view?.navigateTo(
            VIEW.LOCATION,
            LOCATION_REQUEST,
            "location",
            Location(placemark.lat, placemark.lng, placemark.zoom)
        )
    }
}

override fun doActivityResult(requestCode: Int, resultCode: Int, data: Intent) {
    when (requestCode) {
        IMAGE_REQUEST -> {
            placemark.image = data.data.toString()
            view?.showPlacemark(placemark)
        }
        LOCATION_REQUEST -> {
            val location = data.extras?.getParcelable<Location>("location")!!
            placemark.lat = location.lat
            placemark.lng = location.lng
            placemark.zoom = location.zoom
        }
    }
}
```





<https://structure101.com/>



ProductsResourcesDownloadStore



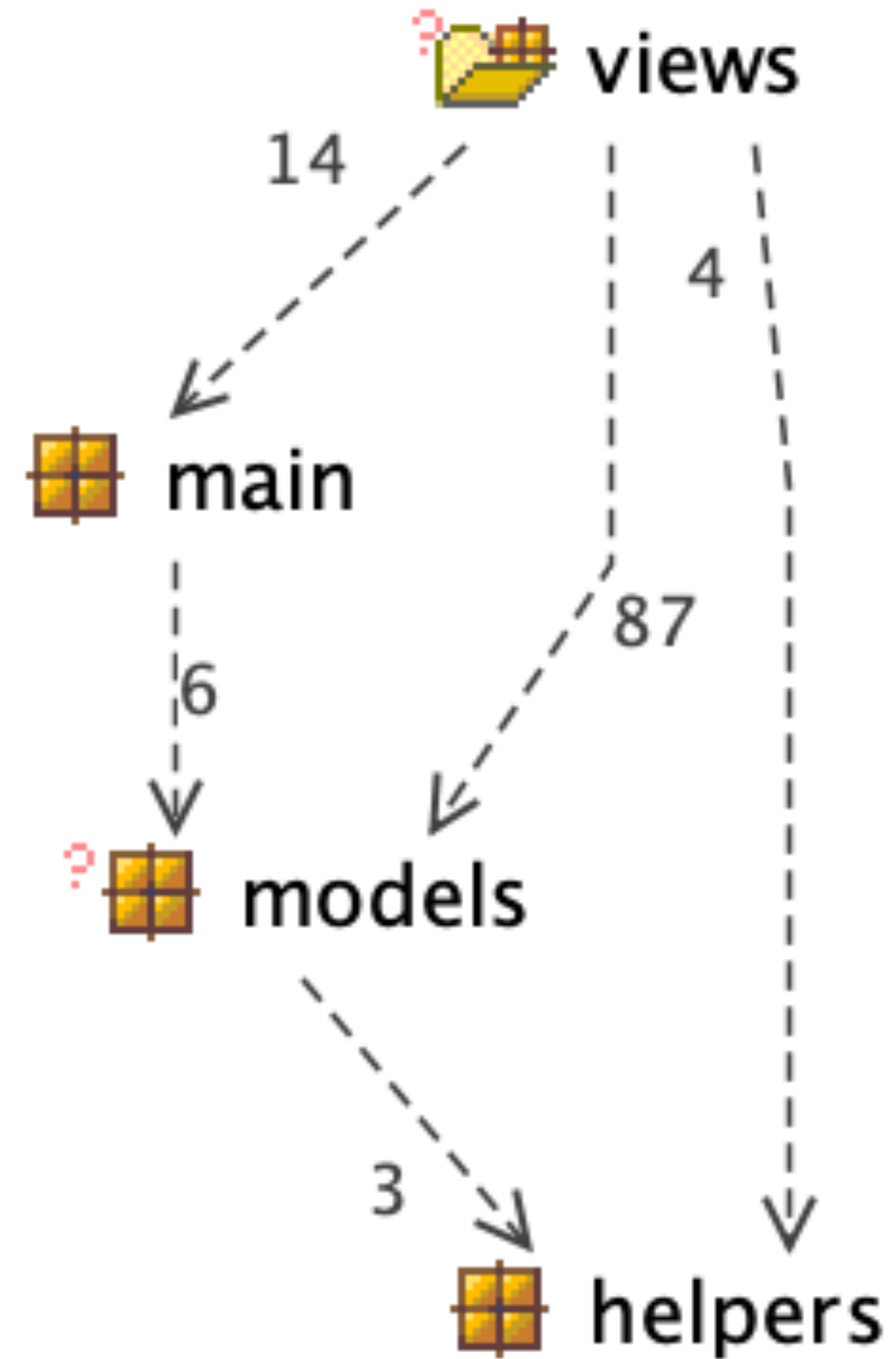
# structure101

PUT YOUR  
SOFTWARE STRUCTURE  
TO WORK

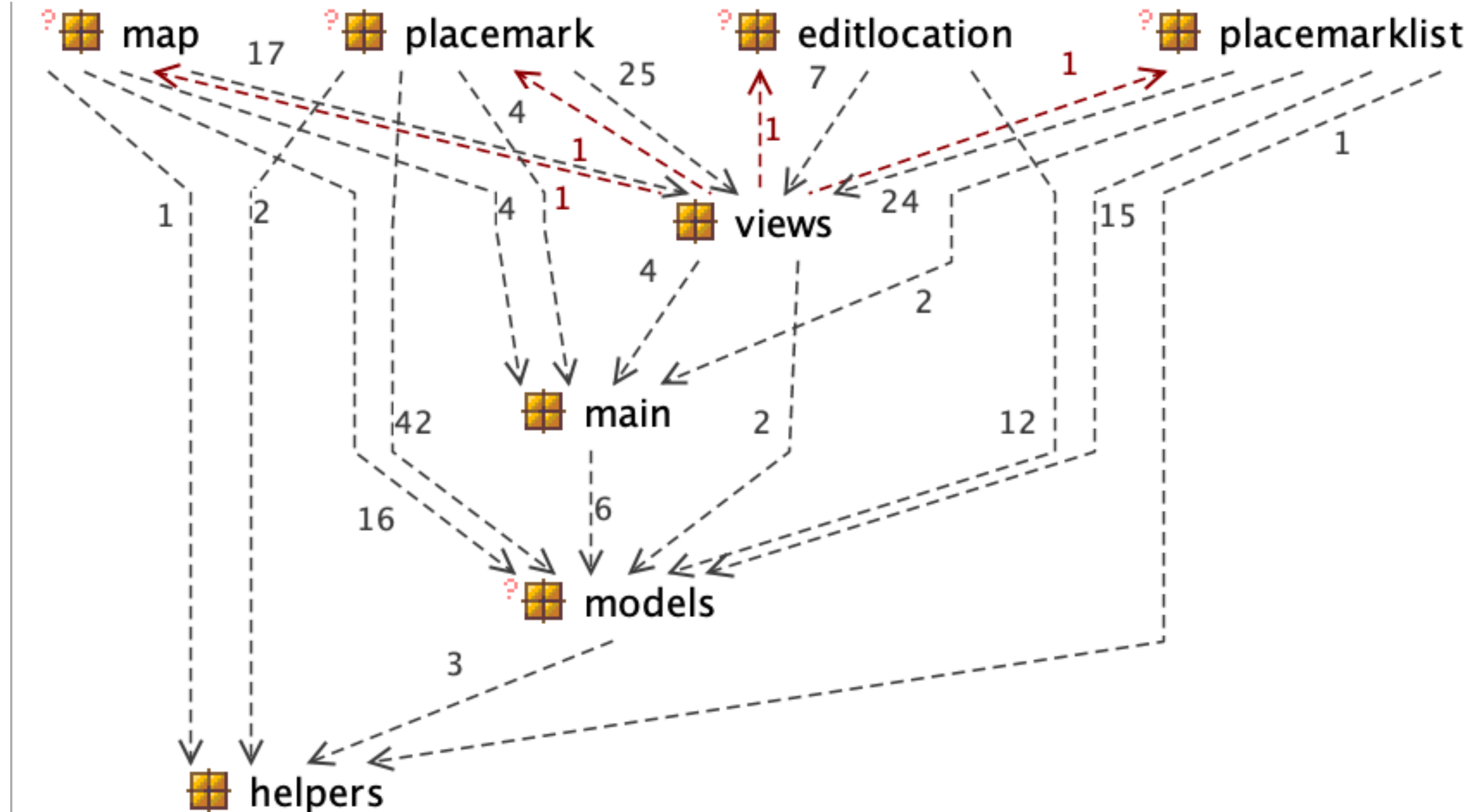
DOWNLOAD NOW

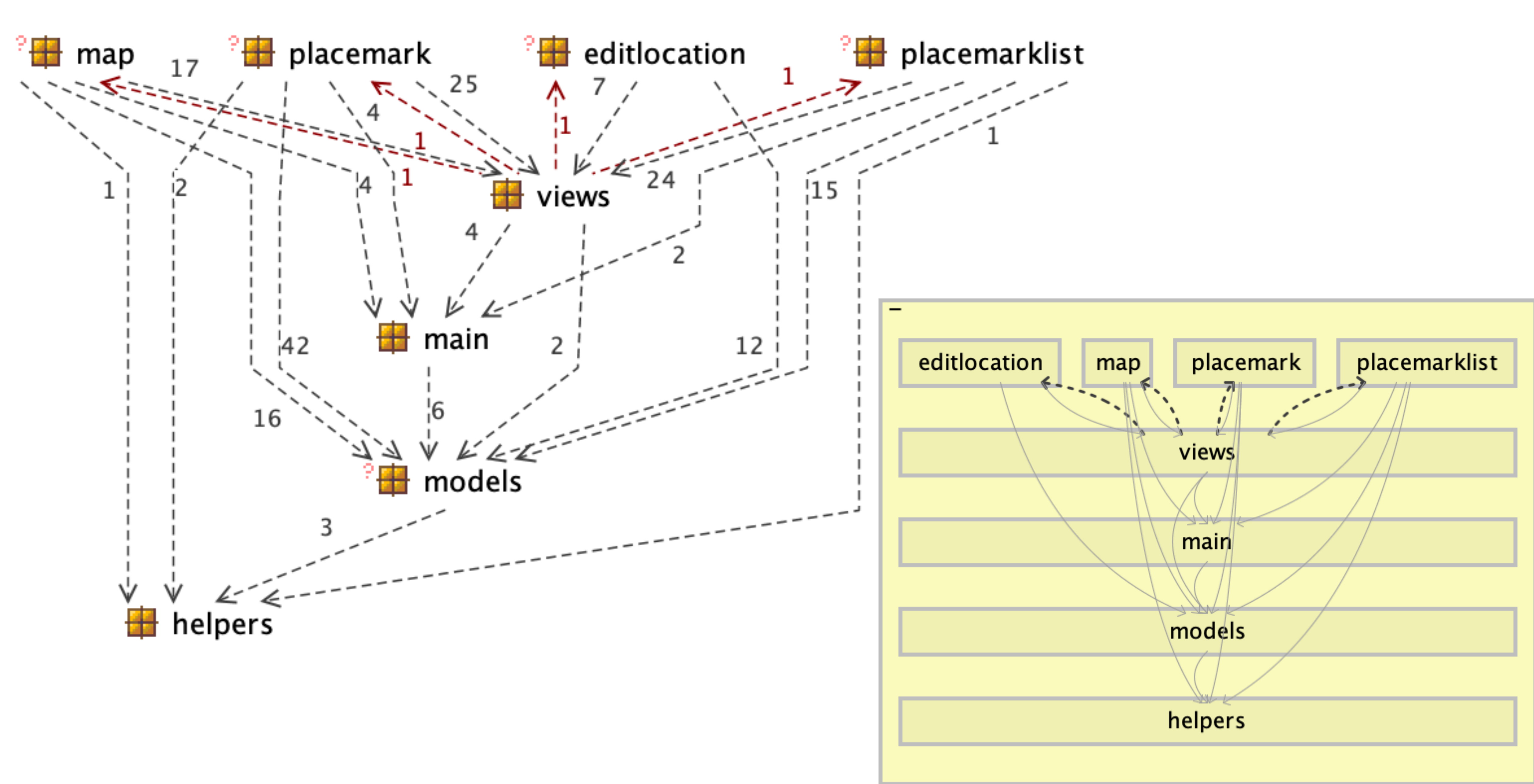


- ▼ org.wit.placemark
  - ▼ helpers
    - FileHelpers.kt
    - ImageHelpers.kt
  - ▼ main
    - MainApp
  - ▼ models
    - PlacemarkJSONStore.kt
    - PlacemarkMemStore.kt
    - PlacemarkModel.kt
    - PlacemarkStore
  - ▼ views
    - ▶ editlocation
    - ▶ map
    - ▶ placemark
    - ▶ placemarklist
    - BasePresenter
    - BaseView.kt

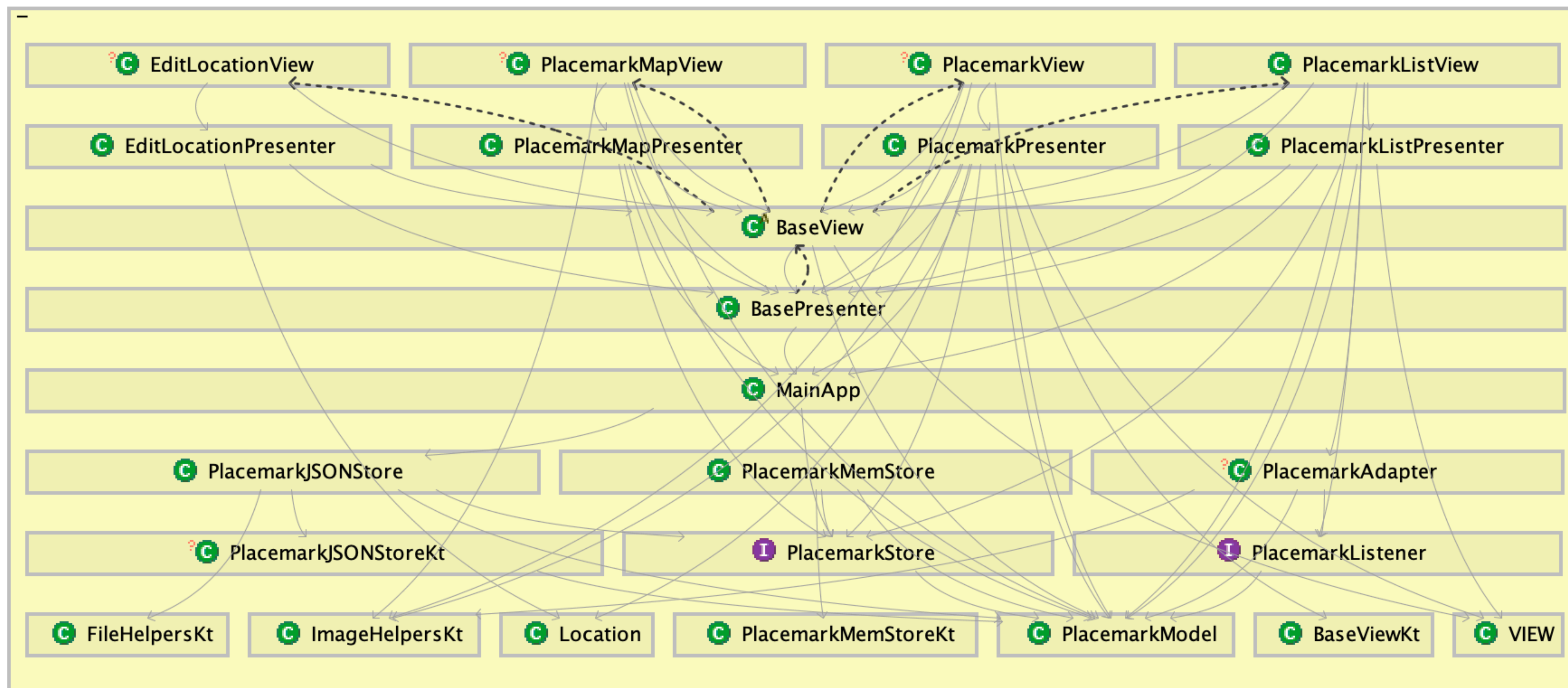


- ▼ org.wit.placemark
  - ▼ helpers
    - FileHelpers.kt
    - ImageHelpers.kt
  - ▼ main
    - MainApp
  - ▼ models
    - PlacemarkJSONStore.kt
    - PlacemarkMemStore.kt
    - PlacemarkModel.kt
    - PlacemarkStore
  - ▼ views
    - ▼ editlocation
      - EditLocationPresenter
      - EditLocationView
    - ▼ map
      - PlacemarkMapPresenter
      - PlacemarkMapView
    - ▼ placemark
      - PlacemarkPresenter
      - PlacemarkView
    - ▼ placemarklist
      - PlacemarkAdapter.kt
      - PlacemarkListPresenter
      - PlacemarkListView
  - BasePresenter
  - BaseView.kt

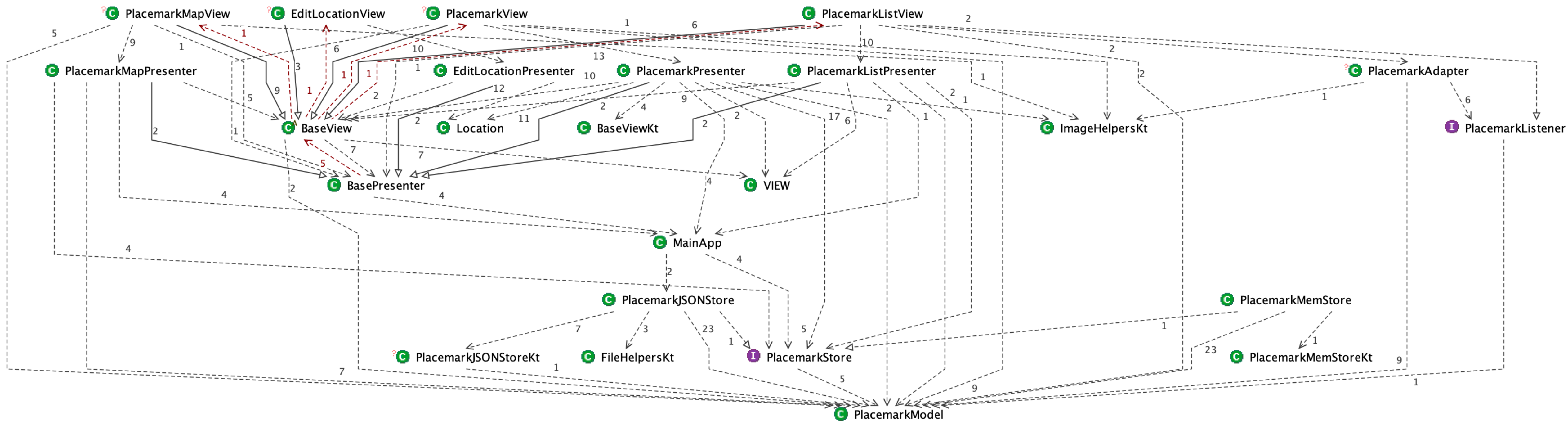












## MVWTF: Demystifying Architecture Patterns

As an Android developer, one of the questions I constantly see asked within the community is “what architecture pattern should I use?”

This discussion usually leads to a handful of buzzwordy acronyms:

- MVC
- MVP
- MVVM
- MVI
- MVU?? (We don’t talk about this but apparently it’s the new kid on the block)

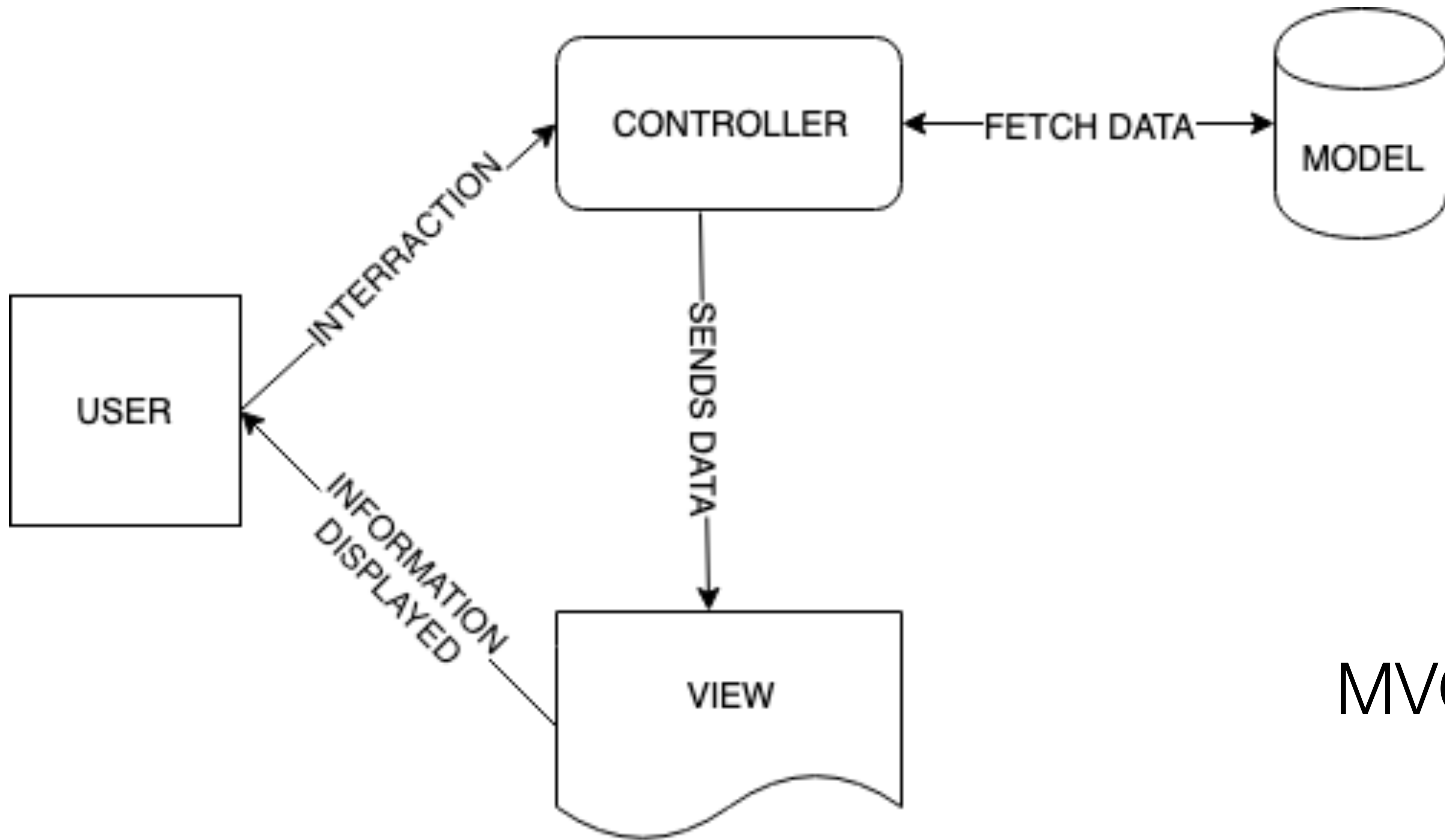
This can be really intimidating to new Android devs, as well as seasoned veterans who are constantly questioning if they’re using the right one. Whether you’re trying to decide which one to learn, or wondering if the one you already use is best for you, this post will help lead you to the right decision.

We should first start off by understanding why we even need architecture patterns. When that question is asked, we get even more buzzwords, saying we want code that is:

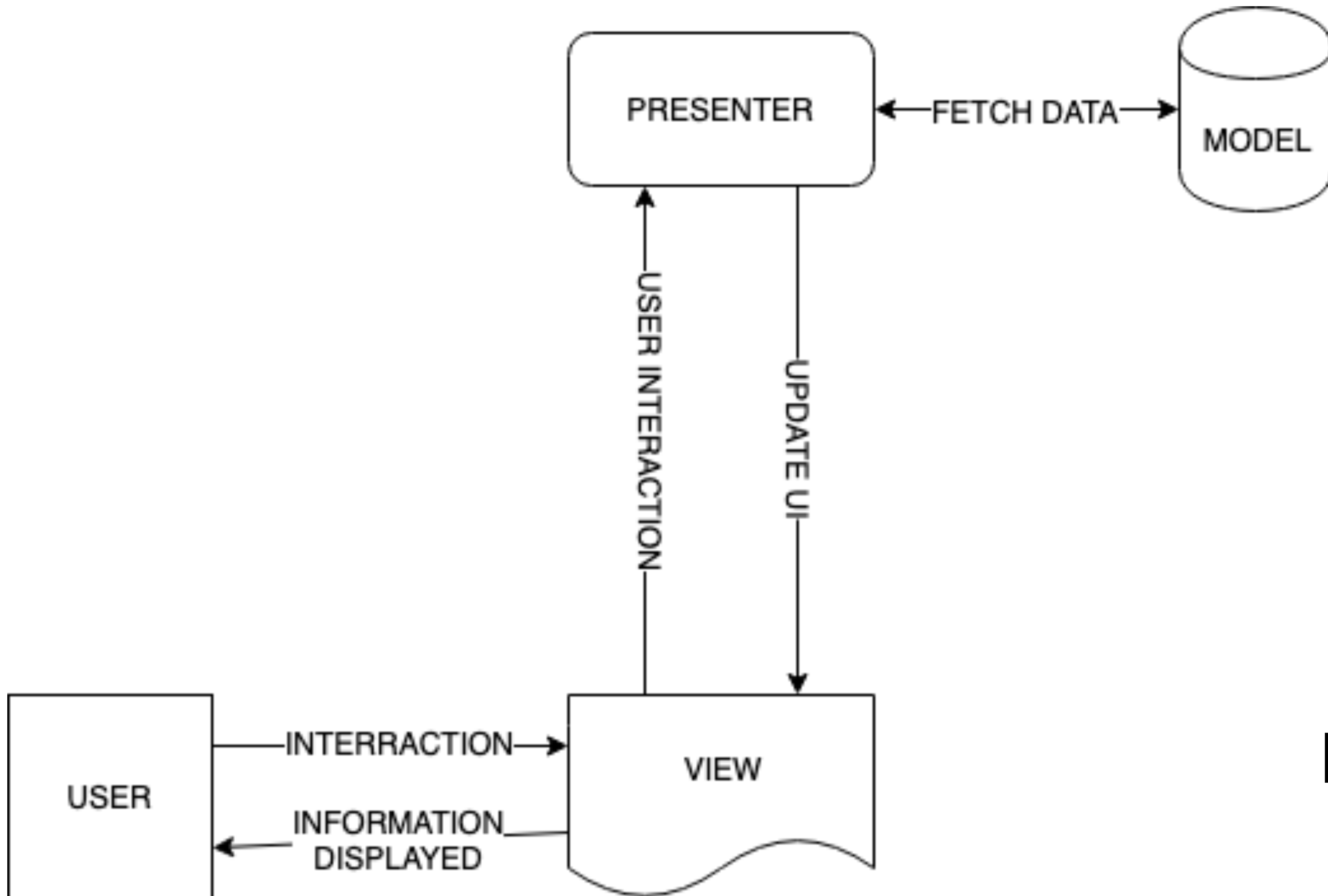
- Maintainable
- Extensible
- Robust
- Testable

<https://androidessence.com/mvwtf>

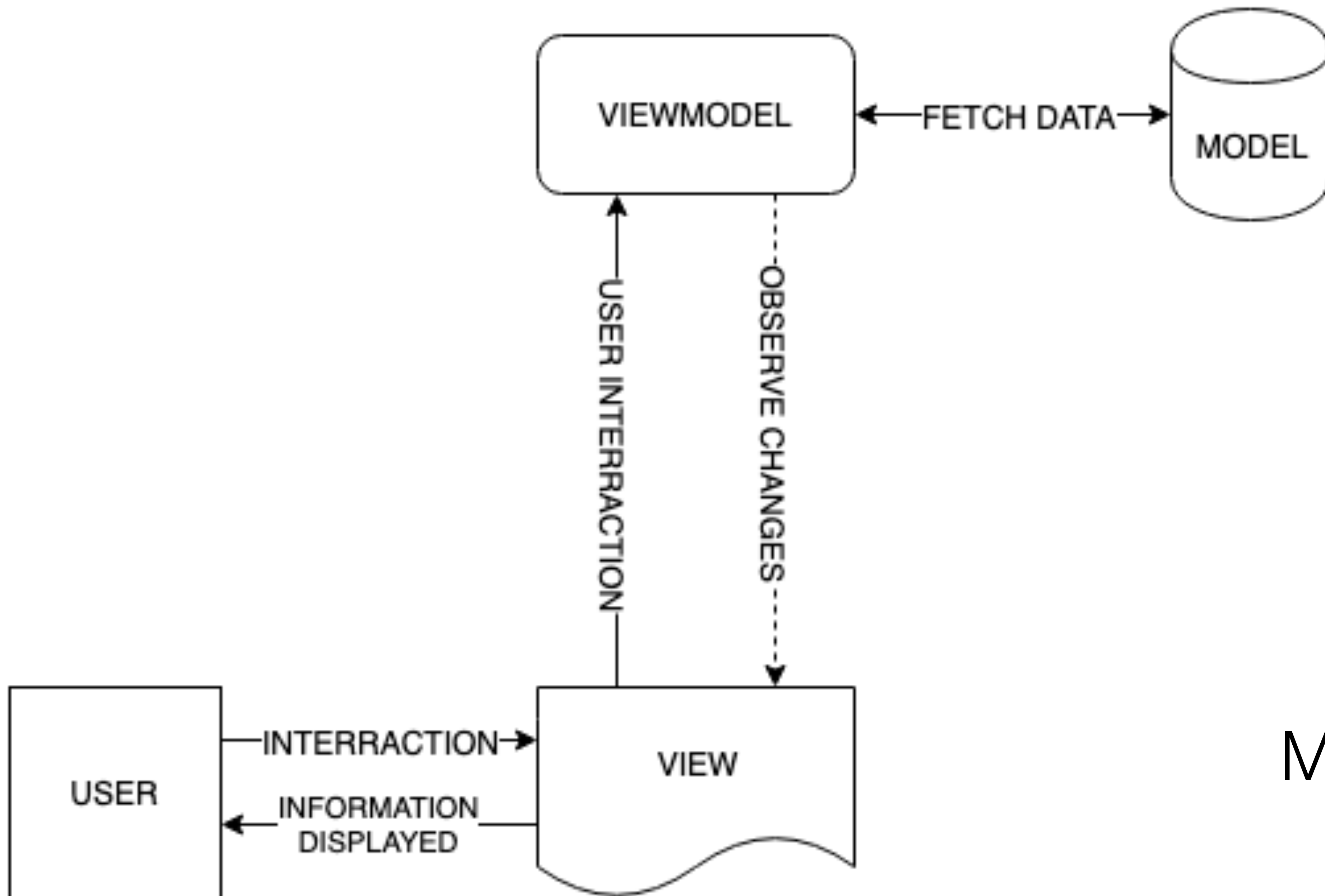
- MVC      - *Model View Controller*
- MVP      - *Model View Presenter*
- MVVM     - *Model View View Model*



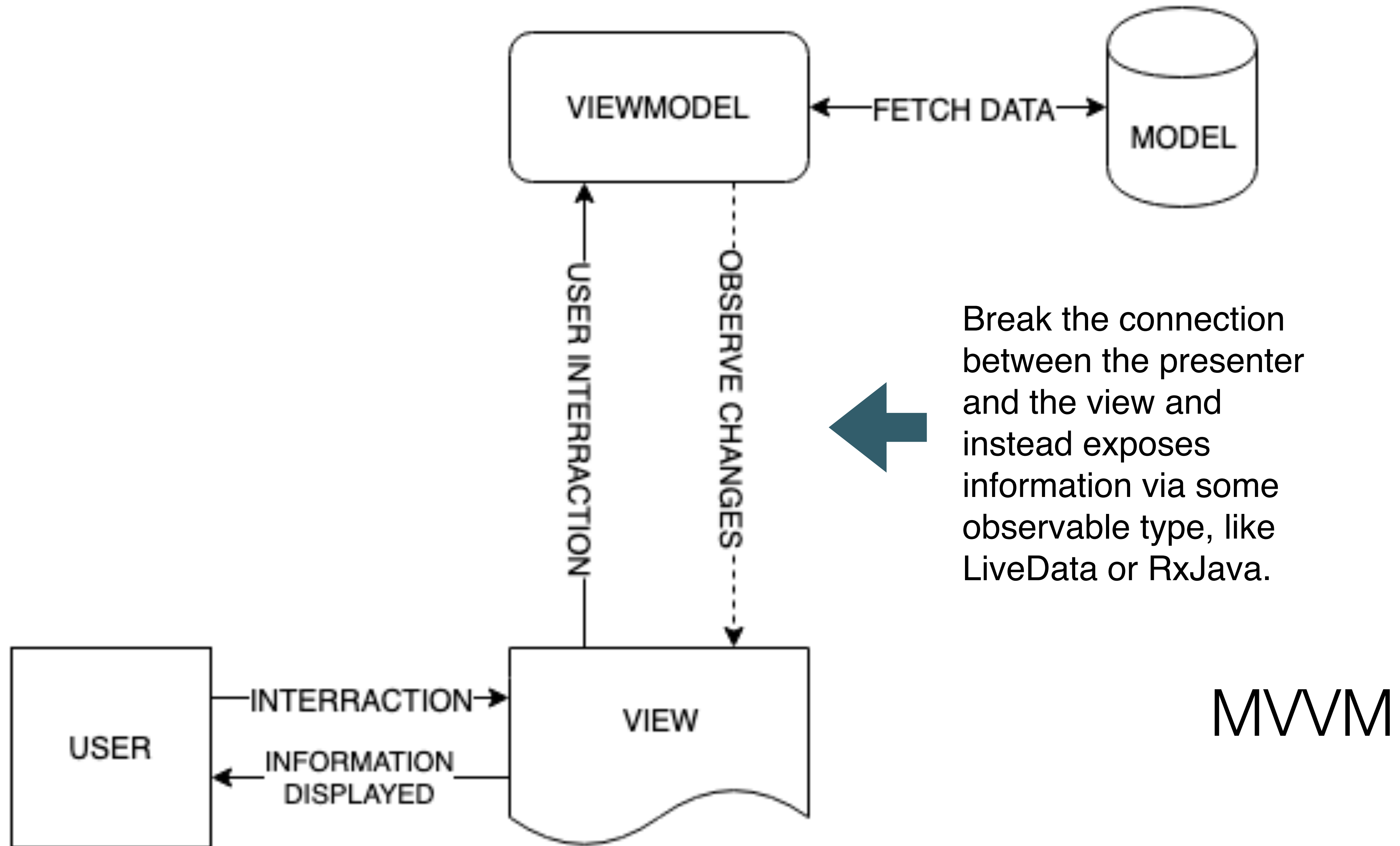
MVC



MVP



MVVM



Break the connection between the presenter and the view and instead exposes information via some observable type, like LiveData or RxJava.

MVVM