образование ПОЛИТЕХ одноклассники экосистема

Паттерны проектирования. Навигация

Нуртдинов Артур

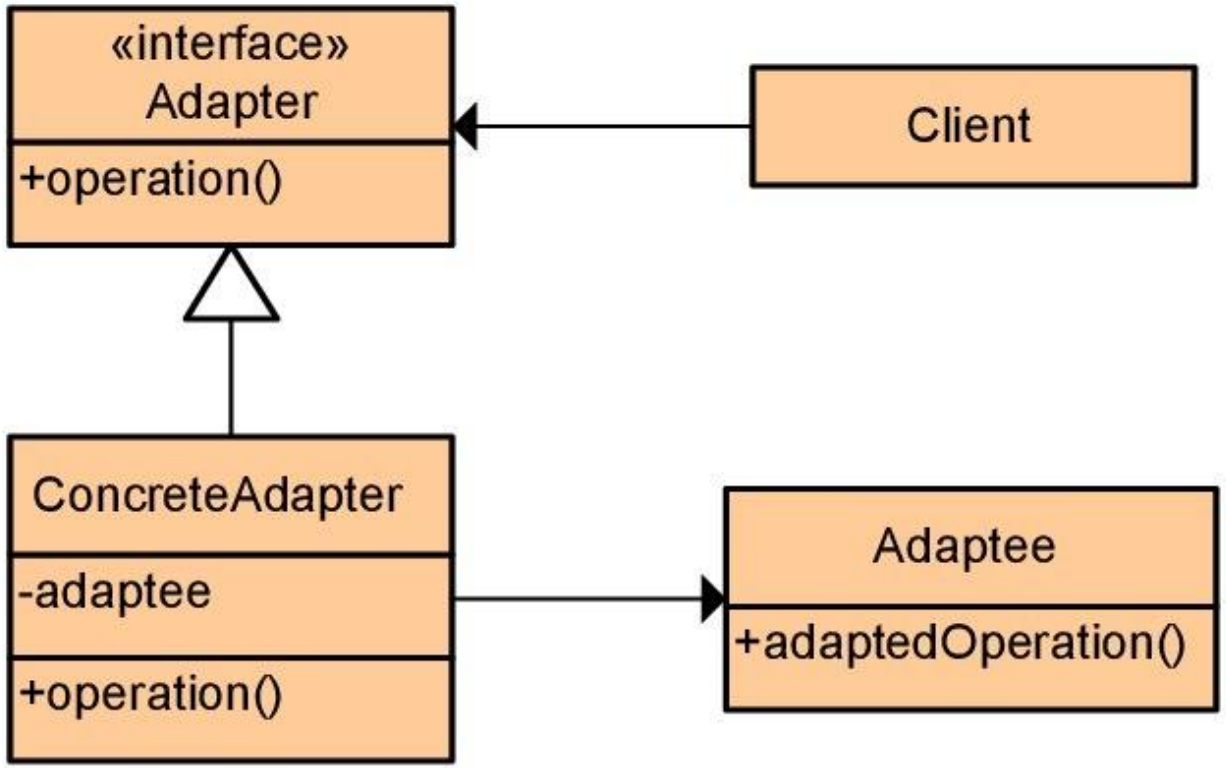
План лекции

1. Общая информация о паттернах в программировании
2. Популярные паттерны в android разработке
3. Dependency Injection
4. MV* паттерны
5. Навигация

Общая информация о паттернах

- Шаблоны проектирования — это допускающие многократное использование оптимизированные решения проблем программирования, с которыми мы сталкиваемся каждый день.
- Есть три основных типа шаблонов проектирования:
- **Структурные** шаблоны, в общем случае, имеют дело с отношениями между объектами, облегчая их совместную работу.
- **Порождающие** шаблоны обеспечивают механизмы инстанциирования, облегчая создание объектов способом, который наиболее соответствует ситуации.
- **Поведенческие** шаблоны используются в коммуникации между объектами, делая её более лёгкой и гибкой.

Структурный паттерн Adapter



Адаптер
Adapter

Тип: Структурный

Что это:

Конвертирует интерфейс класса в другой интерфейс, ожидаемый клиентом. Позволяет классам с разными интерфейсами работать вместе.

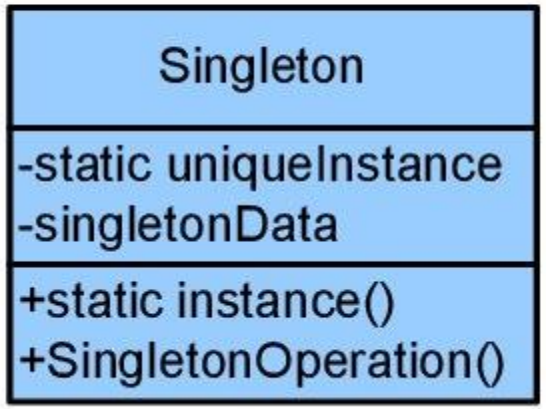
Порождающий паттерн Singleton

Одиночка
Singleton

Тип: Порождающий

Что это:

Гарантирует, что класс имеет только один экземпляр и предоставляет глобальную точку доступа к нему.



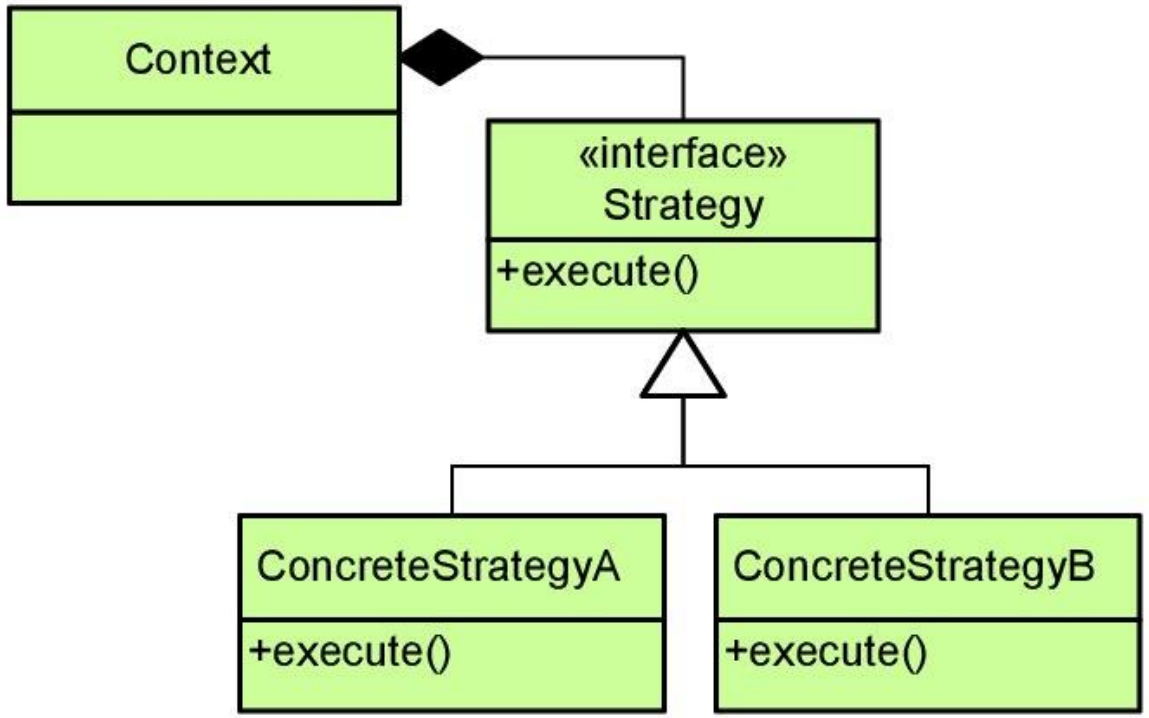
Поведенческий паттерн Strategy

Стратегия
Strategy

Тип: Поведенческий

Что это:

Определяет группу алгоритмов, инкапсулирует их и делает взаимо-заменяемыми. Позволяет изменять алгоритм независимо от клиентов, его использующих.



Для чего нужны паттерны

- Ускоряют и облегчают написание кода.
- Позволяют не «изобретать велосипед», а воспользоваться готовым проверенным принципом.
- При грамотном использовании делают код более читаемым и эффективным.
- Упрощают взаимопонимание между разработчиками.
- Помогают избавиться от типовых ошибок.
- Не зависят от языка программирования и его особенностей.
- Позволяют реализовывать сложные задачи быстрее и проще.

Наиболее важные паттерны для android разработки

1. Dependency Injection - процесс предоставления внешней зависимости программному компоненту
2. MV* паттерны (MVC, MVP, MVVM, MVI)

Dependency Injection (DI)

Внедрение зависимостей — это стиль настройки объекта, при котором поля объекта задаются внешней сущностью. Другими словами, объекты настраиваются внешними объектами. DI — это альтернатива самонастройке объектов.

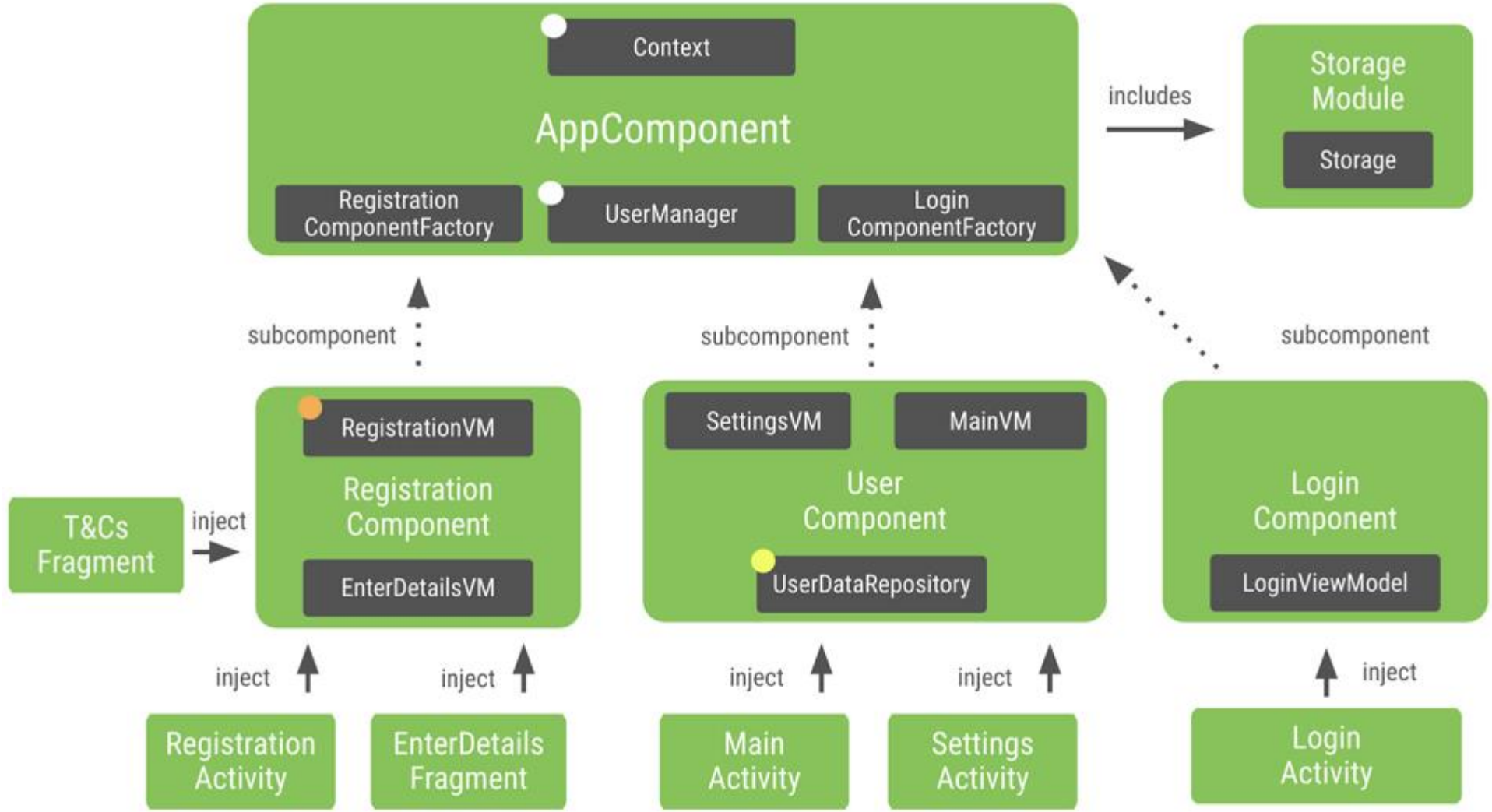
Преимущества:

- Меньше зависимостей
- Меньше «перенос» зависимостей
- Код проще переиспользовать
- Код удобнее тестировать
- Код удобнее читать

Библиотеки Dependency Injection (DI)

- Dagger 2 – на данный момент самая популярная для больших и средних проектов
- Hilt – рекомендация гугла, больше подходит для небольших проектов
- Yatagan – DI фреймворк от Яндекс
- Koin – Полностью написанный на котлине DI фреймворк

Dagger 2


Context
AppComponent
Registration ComponentFactory
UserManager
Login ComponentFactory
includes
Storage Module
Storage
subcomponent
subcomponent
subcomponent
RegistrationVM
Registration Component
EnterDetailsVM
SettingsVM
MainVM
User Component
UserDataRepository
Login Component
LoginViewModel
T&Cs Fragment
inject
inject
inject
inject
inject
inject
Registration Activity
EnterDetails Fragment
Main Activity
Settings Activity
Login Activity

Dagger 2

```kotlin
@Component(modules = [StorageModule::class])
interface AppComponent {

    // Factory to create instances of the AppComponent
    @Component.Factory
    interface Factory {
        // With @BindsInstance, the Context passed in will be available in the graph
        fun create(@BindsInstance context: Context): AppComponent
    }

    fun inject(activity: RegistrationActivity)
}
```

```kotlin
// Tells Dagger this is a Dagger module
// Because of @Binds, StorageModule needs to be an abstract class
@Module
abstract class StorageModule {

    // Makes Dagger provide SharedPreferencesStorage when a Storage type is requested
    @Binds
    abstract fun provideStorage(storage: SharedPreferencesStorage): Storage
}
```

Dagger 2

```
// @Inject tells Dagger how to provide instances of this type
class SharedPreferencesStorage @Inject constructor(context: Context) : Storage { ... }
```

```
class RegistrationActivity : AppCompatActivity() {

    // @Inject annotated fields will be provided by Dagger
    @Inject lateinit var storage: Storage

    override fun onCreate(savedInstanceState: Bundle?) {
        // Ask Dagger to inject our dependencies
        (application as MyApplication).appComponent.inject(this)

        super.onCreate(savedInstanceState)
    }
}
```

Dagger 2

Annotation	Usage
`@Module`	Used on classes which contains methods annotated with `@Provides`.
`@Provides`	Can be used on methods in classes annotated with `@Module` and is used for methods which provides objects for dependencies injection.
`@Singleton`	Single instance of this provided object is created and shared.
`@Component`	Used on an interface. This interface is used by Dagger 2 to generate code which uses the modules to fulfill the requested dependencies.

Hilt

```
@HiltAndroidApp
class LogApplication : Application() {
    ...
}
```

```
@AndroidEntryPoint
class LogsFragment : Fragment() {

    @Inject lateinit var logger: LoggerLocalDataSource
    @Inject lateinit var dateFormatter: DateFormatter

    ...
}
```

Hilt

```
class DateFormatter @Inject constructor() { ... }

class LoggerLocalDataSource @Inject constructor(private val logDao: LogDao) {
    ...
}
```

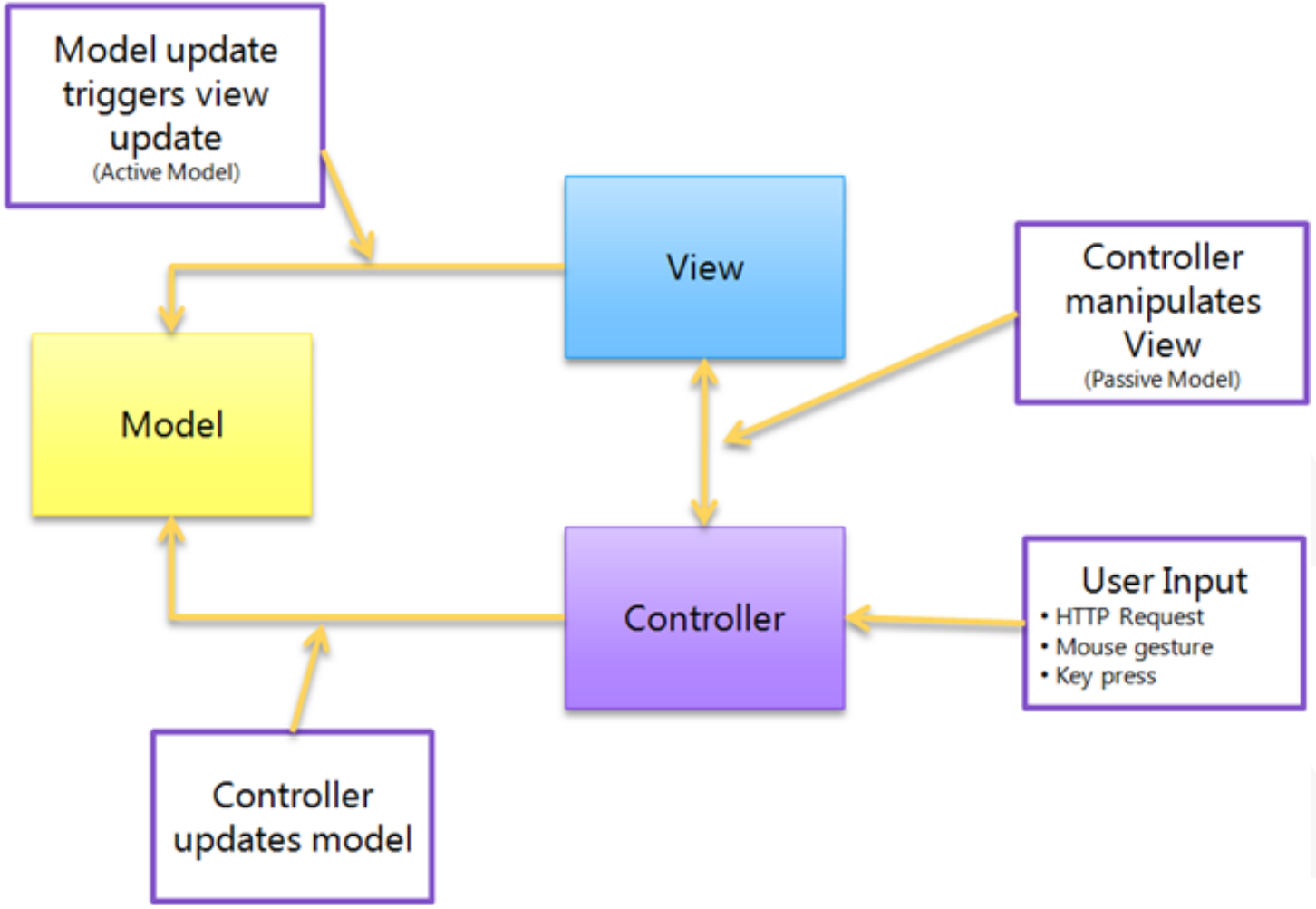
Hilt

```kotlin
@InstallIn(SingletonComponent::class)
@Module
object DatabaseModule {

    @Provides
    fun provideLogDao(database: AppDatabase): LogDao {
        return database.logDao()
    }

    @Provides
    @Singleton
    fun provideDatabase(@ApplicationContext appContext: Context): AppDatabase {
        return Room.databaseBuilder(
            appContext,
            AppDatabase::class.java,
            "logging.db"
        ).build()
    }
}
```

MV* паттерны

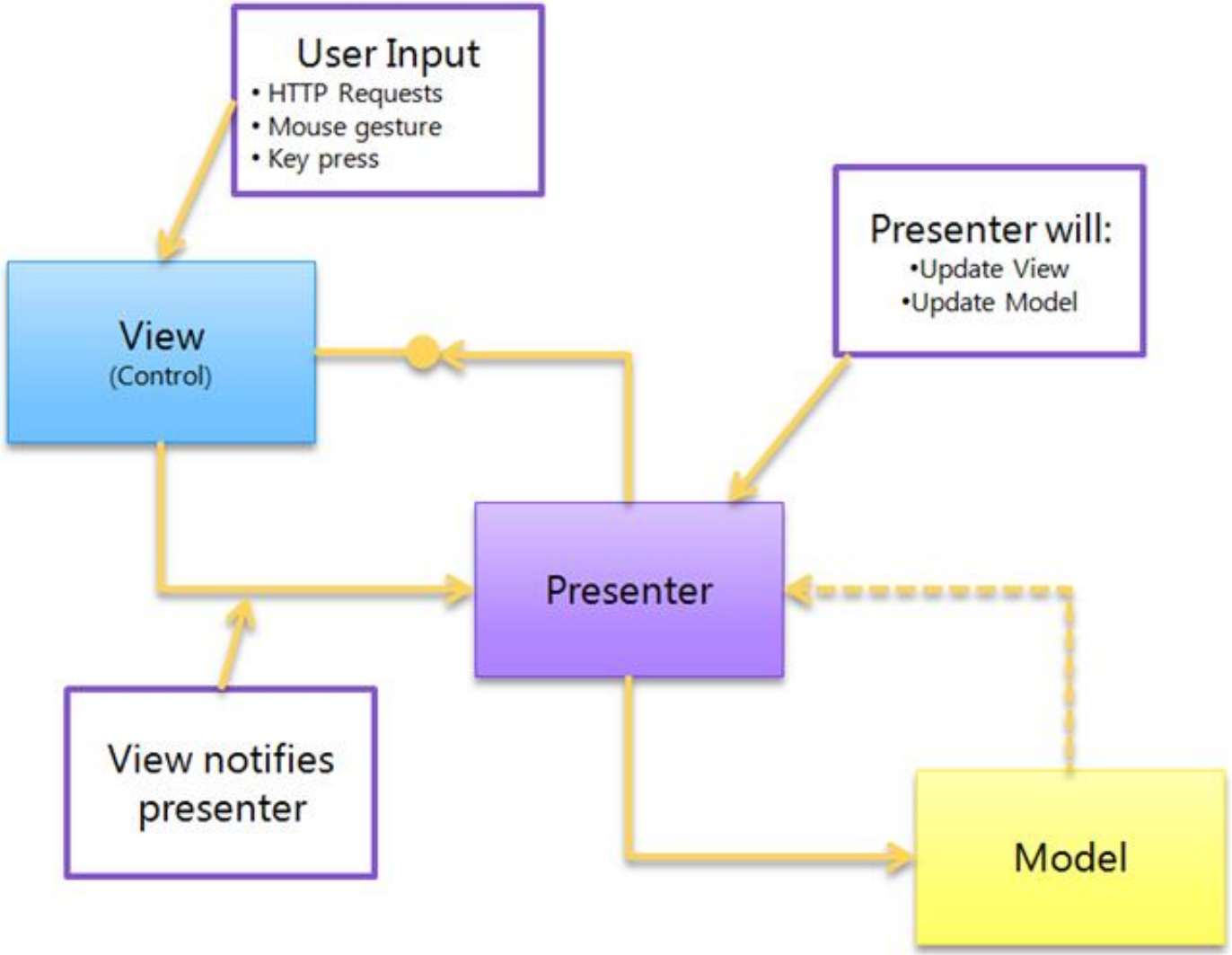
- Разделение логики (отделение работы с UI от работы с данными)
- Удобнее тестировать
- Удобнее поддерживать и расширять
- Удобнее переиспользовать код

MVC



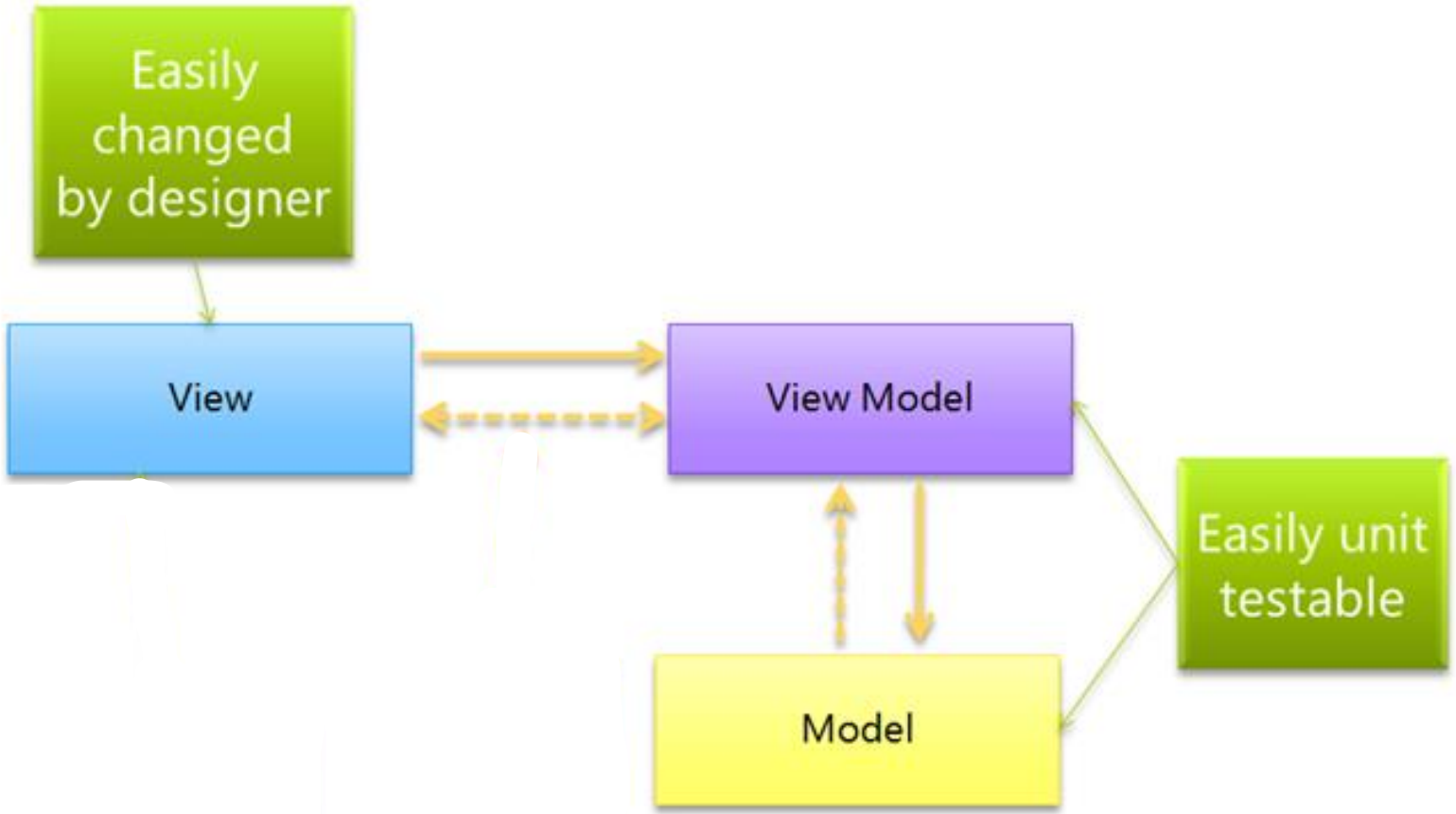
- Необходимость использования большего количества ресурсов
- Усложнен механизм разделения программы на модули
- Усложнен процесс расширения функционала

MVP



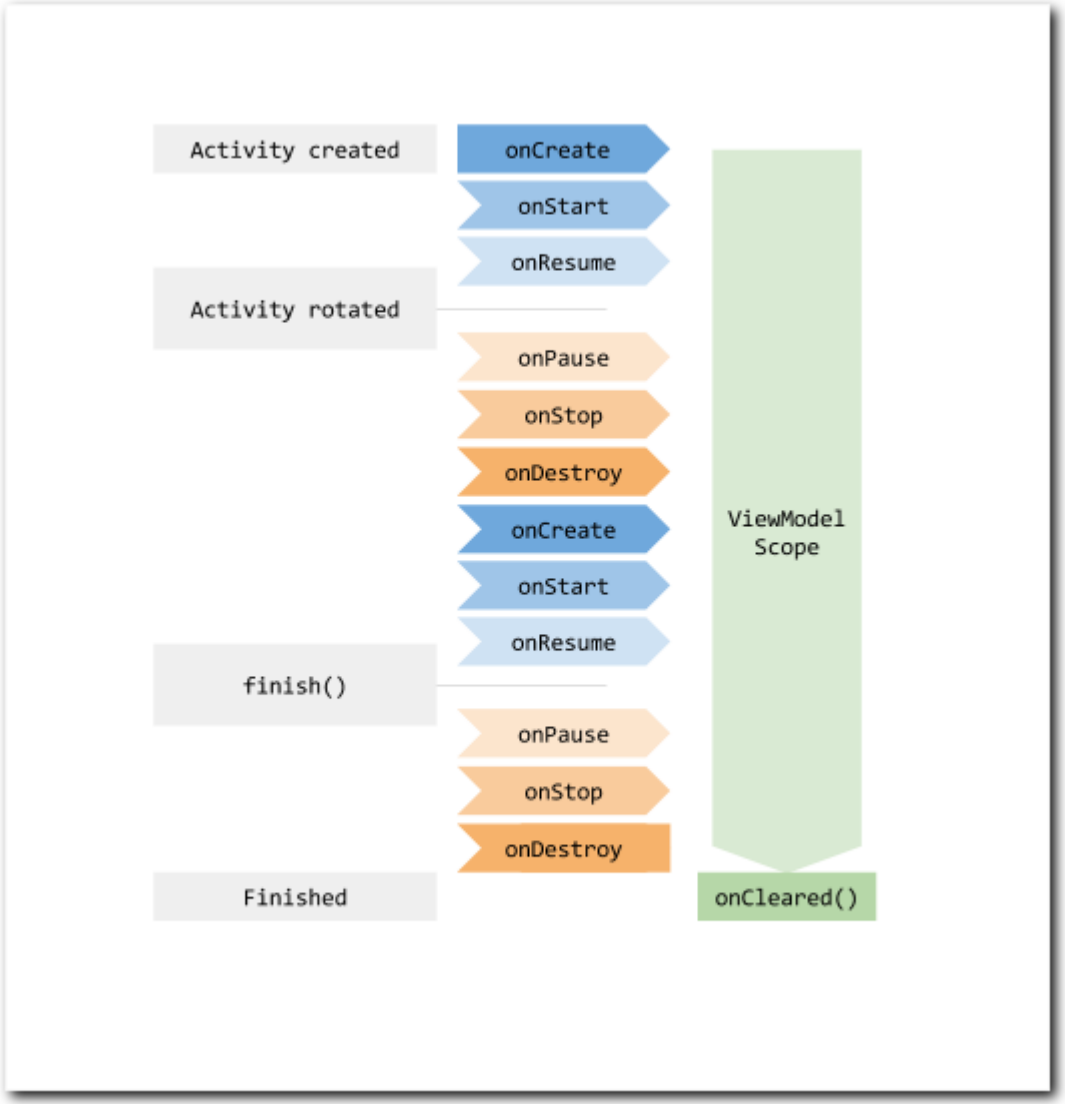
- Циклическая зависимость между Presenter и View
- Presenter не знает о жизненном цикле View
- Состояние View необходимо хранить в Model

MVVM



- ViewModel не знает о View
- View может быть несколько
- Состояние View хранится во ViewModel

MVVM. ViewModel Lifecycle



Cleared when:

- In the case of an activity, when it finishes.
- In the case of a fragment, when it detaches.
- In the case of a Navigation entry, when it's removed from the back stack.

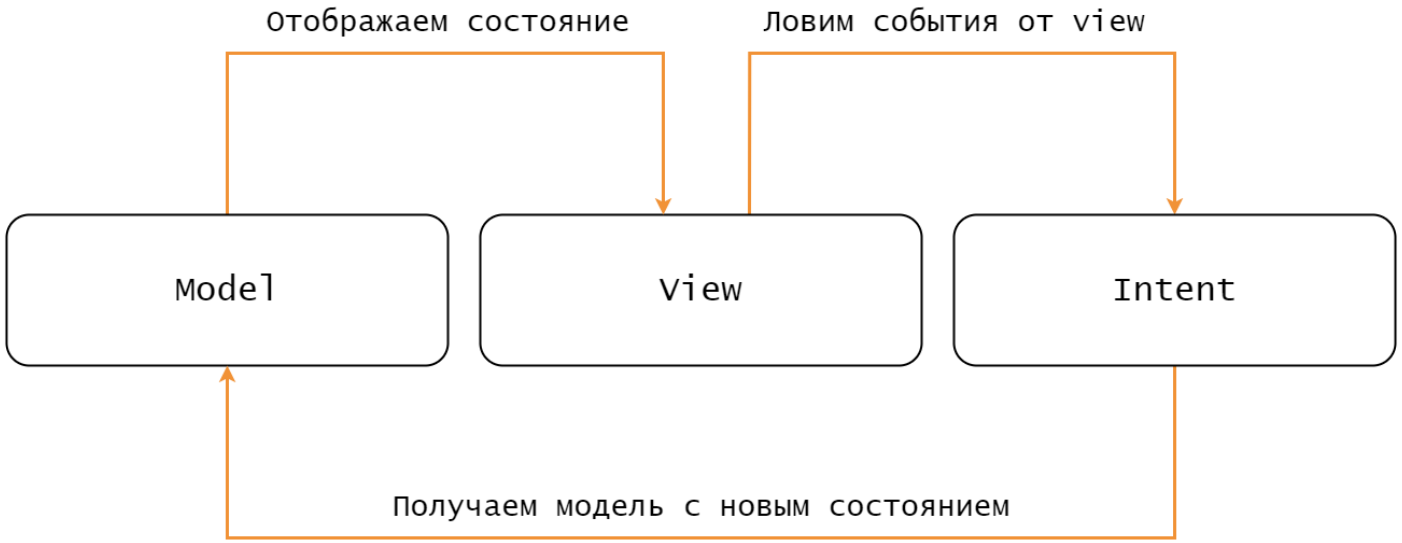
MVVM. ViewModel

```
class SampleViewModel : ViewModel() {
    private val _state = MutableStateFlow(State(emptyList()))
    val state = _state.asStateFlow()

    data class State(val titles: List<String>)
}
```

MVVM. View

```kotlin
class SampleFragment : Fragment() {
    private lateinit var viewModel: SampleViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        viewModel = ViewModelProvider( owner: this)[SampleViewModel::class.java]
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return inflater.inflate(R.layout.fragment_sample, container,  attachToRoot: false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        lifecycleScope.launch { this: CoroutineScope
            repeatOnLifecycle(Lifecycle.State.STARTED) { this: CoroutineScope
                viewModel.state.collect { it: SampleViewModel.State
                    // handle state
                }
            }
        }
    }
}
```

MVI



- **Однонаправленность**. С данными работает только одна сущность, и мы знаем, кто изменяет данные, зачем и почему.
- **Неизменяемость состояния**. Новое состояние складывается из предыдущего и какого-то действия. Изменить данные мы не можем, можем только получить новые.

Навигация

1. Использовать Jetpack Navigation Component
2. Вручную управлять стэком фрагментов и создавать транзакции

Транзакции

FragmentManager - компонент для управления фрагментами

FragmentTransaction - транзакция для внесения изменения стека фрагментов

- add()
- remove()
- replace()
- addToBackStack()

Транзакции

Как начать транзакцию (в конце нужно не забыть commit()):

```
val fragmentManager = ...
val fragmentTransaction = fragmentManager.beginTransaction()
```

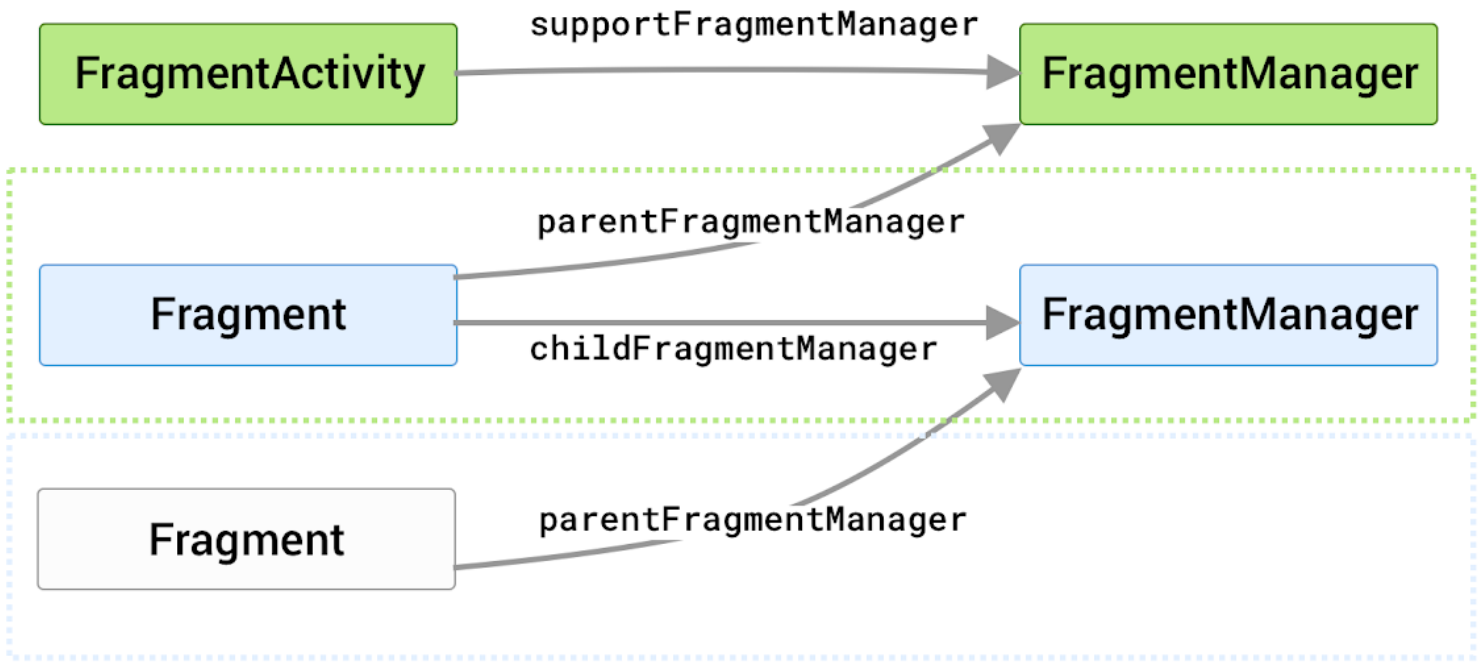
Или (с помощью extensions):

```
// Create new fragment
val fragmentManager = // ...

// Create and commit a new transaction
fragmentManager.commit {
    setReorderingAllowed(true)
    // Replace whatever is in the fragment_container view with this fragment
    replace<ExampleFragment>(R.id.fragment_container)
}
```

```
supportFragmentManager.commit {
    replace<ExampleFragment>(R.id.fragment_container)
    setReorderingAllowed(true)
    addToBackStack("name") // name can be null
}
```

Fragment Manager

Level	Instance of	Fragment Manager
Host Activity	FragmentActivity	FragmentManager
Host Fragment	Fragment	FragmentManager
Child Fragment(s)	Fragment	



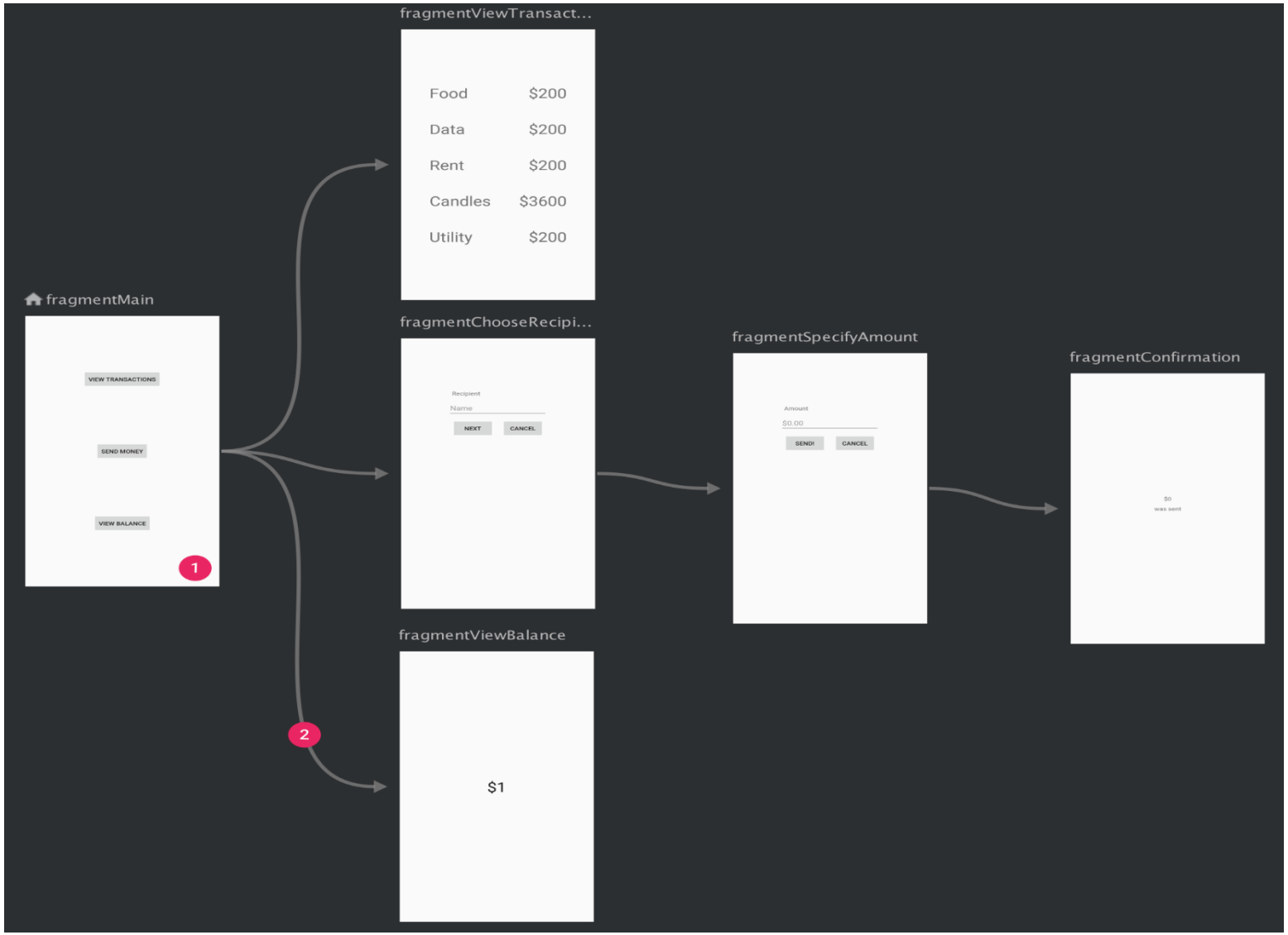
Each FragmentManager manages child fragments of the host

Navigation Component

```
dependencies {
  val nav_version = "2.5.3"

  // Java language implementation
  implementation("androidx.navigation:navigation-fragment:$nav_version")
  implementation("androidx.navigation:navigation-ui:$nav_version")

  // Kotlin
  implementation("androidx.navigation:navigation-fragment-ktx:$nav_version")
  implementation("androidx.navigation:navigation-ui-ktx:$nav_version")

  // Feature module Support
  implementation("androidx.navigation:navigation-dynamic-features-fragment:$nav_version")

  // Testing Navigation
  androidTestImplementation("androidx.navigation:navigation-testing:$nav_version")

  // Jetpack Compose Integration
  implementation("androidx.navigation:navigation-compose:$nav_version")
}
```

https://developer.android.com/guide/navigation/navigation-getting-started

Navigation Component. Navigation Graph



Navigation Component. Navigation Graph

```xml
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:android="http://schemas.android.com/apk/res/android"
    app:startDestination="@id/blankFragment">
    <fragment
        android:id="@+id/blankFragment"
        android:name="com.example.cashdog.cashdog.BlankFragment"
        android:label="@string/label_blank"
        tools:layout="@layout/fragment_blank" >
        <action
            android:id="@+id/action_blankFragment_to_blankFragment2"
            app:destination="@id/blankFragment2" />
    </fragment>
    <fragment
        android:id="@+id/blankFragment2"
        android:name="com.example.cashdog.cashdog.BlankFragment2"
        android:label="@string/label_blank_2"
        tools:layout="@layout/fragment_blank_fragment2" />
</navigation>
```

Navigation Component. FragmentContainerView

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.appcompat.widget.Toolbar
        .../>

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/nav_host_fragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"

        app:defaultNavHost="true"
        app:navGraph="@navigation/nav_graph" />

    <com.google.android.material.bottomnavigation.BottomNavigationView
        .../>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Navigation Component. Navigation Graph

После создания графа необходимо использовать id у action для непосредственной навигации между экранами:

Kotlin:

- Fragment.findNavController()
- View.findNavController()
- Activity.findNavController(viewId: Int)

Java:

- NavHostFragment.findNavController(Fragment fragment)
- Navigation.findNavController(Activity activity, @IdRes int viewId)
- Navigation.findNavController(View view)

Navigation Component. NavController

Для обращения к NavController в активити, которая содержит NavHostFragment) необходимо использовать следующий код:

```
val navHostFragment =
        supportFragmentManager.findFragmentById(R.id.nav_host_fragment) as NavHostFragment
val navController = navHostFragment.navController
```

Имея экземпляр NavController, используем action id из навигационного графа для навигации:

```
val navController = ...
navController.navigate(R.id.action_blankFragment_to_blankFragment2)
```

Navigation Component. Safe Args

Safe Args генерируют класс для кажого destination. Сгенерированный класс содержит слово Directions в имени и имеет статическую функцию для каждого action:

```
override fun onClick(v: View) {
    val amount: Float = ...
    val action =
        SpecifyAmountFragmentDirections
            .actionSpecifyAmountFragmentToConfirmationFragment(amount)
    v.findNavController().navigate(action)
}
```

Navigation Component. Возможности графа

```
<fragment android:id="@+id/a"
          android:name="com.example.myapplication.FragmentA"
          android:label="a"
          tools:layout="@layout/a">
    <action android:id="@+id/action_a_to_b"
            app:destination="@id/b"
            app:enterAnim="@anim/nav_default_enter_anim"
            app:exitAnim="@anim/nav_default_exit_anim"
            app:popEnterAnim="@anim/nav_default_pop_enter_anim"
            app:popExitAnim="@anim/nav_default_pop_exit_anim"/>
</fragment>

<fragment android:id="@+id/b"
          android:name="com.example.myapplication.FragmentB"
          android:label="b"
          tools:layout="@layout/b">
    <action android:id="@+id/action_b_to_a"
            app:destination="@id/a"
            app:enterAnim="@anim/nav_default_enter_anim"
            app:exitAnim="@anim/nav_default_exit_anim"
            app:popEnterAnim="@anim/nav_default_pop_enter_anim"
            app:popExitAnim="@anim/nav_default_pop_exit_anim"
            app:popUpTo="@+id/a"
            app:popUpToInclusive="true"/>
</fragment>
```

Материалы

1. https://habr.com/ru/post/307942/ - Что такое шаблоны проектирования?
2. https://habr.com/ru/post/210288/ - Шпаргалки по паттернам
3. https://habr.com/ru/post/151219/ - Шпаргалка по MV* паттернам
4. https://habr.com/ru/post/350068/ - Про DI
5. https://developer.android.com/training/dependency-injection/dagger-android - Dagger 2 в Android
6. https://developer.android.com/training/dependency-injection/hilt-android - Hilt в Android
7. https://developer.android.com/guide/fragments/transactions - Fragment Transactions
8. https://developer.android.com/guide/fragments/fragmentmanager - Fragment Manager
9. https://developer.android.com/guide/navigation/navigation-navigate - Navigation component guide

Спасибо!

образование

ПОЛИТЕХ

одноклассники
экосистема