ПРАКТИЧЕСКАЯ РАБОТА №3

3.1. Фрагменты

Fragment – часть приложения, представляющая повторно используемую часть пользовательского интерфейса. Фрагмент имеет собственную разметку и управляет ей, обладает собственным жизненным циклом. Фрагменты не могут жить сами по себе — они должны размещаться в Activity или другом фрагменте. Иерархия представлений фрагмента становится частью иерархии представлений хоста или присоединяется к ней.

Фрагменты привносят модульность и возможность повторного использования в пользовательском интерфейсе Activity, позволяя разделить пользовательский интерфейс на отдельные компоненты. Activity — это идеальное место для размещения глобальных элементов пользовательского интерфейса приложения, таких как панель навигации. И наоборот, фрагменты лучше подходят для определения пользовательского интерфейса отдельного экрана или части экрана и управления им.

Рассмотрим приложение, которое реагирует на различные размеры экрана. На больших экранах приложение должно отображать статическую панель навигации и список в виде сетки. На небольших экранах приложение должно отображать нижнюю панель навигации и линейный список. Управление всеми этими вариациями внешнего вида может быть громоздким. Отделение элементов навигации от содержимого может сделать этот процесс более управляемым. При таком подходе Activity отвечает за отображение правильного пользовательского интерфейса навигации, а Fragment отображает список с правильным макетом.

Пример можно увидеть на рис. 3.1, слева большой экран содержит панель навигации, управляемую Activity, и список в формате сетки, управляемый Fragment’ом. Справа небольшой экран содержит нижнюю панель навигации, управляемую Activity, и линейный список, управляемый Fragment’ом.

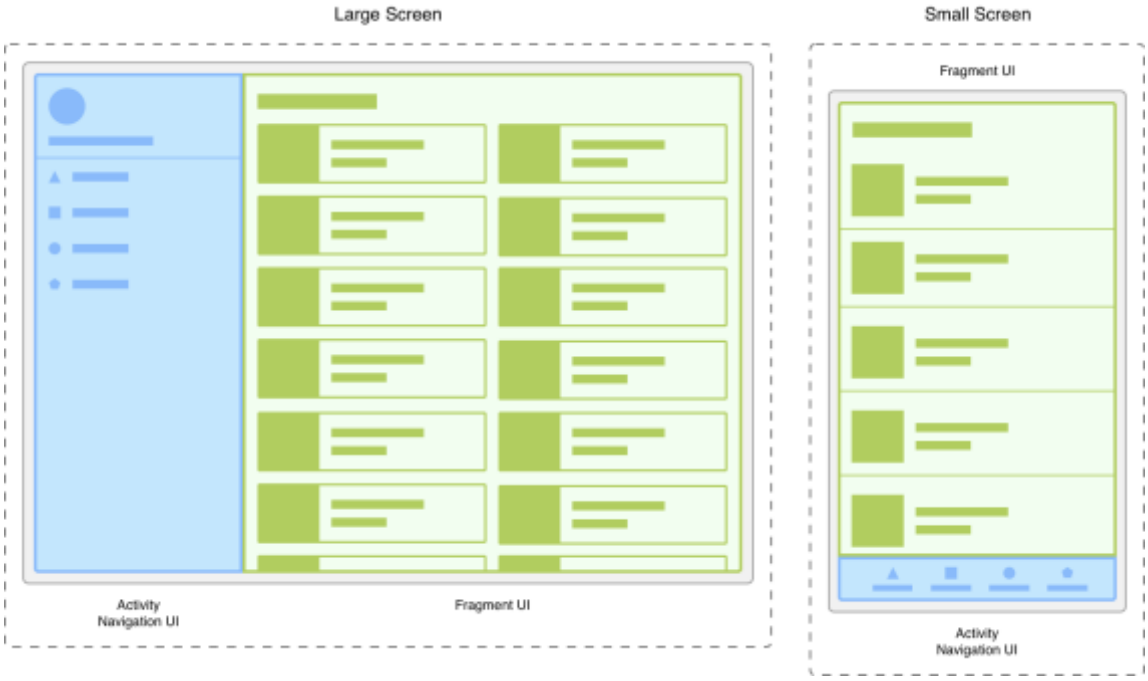

Рисунок 3.1. — Две версии одного и того же экрана на экранах разного размера.

Разделение пользовательского интерфейса на фрагменты упрощает изменение внешнего вида Activity во время работы приложения. Пока Activity находится в состоянии STARTED жизненного цикла или выше, фрагменты можно добавлять, заменять или удалять. Можно хранить записи об этих изменениях в резервном стеке, которым управляет Activity, что позволяет отменить изменения.

Можно использовать несколько экземпляров одного и того же класса фрагментов в одном и том же Activity, в нескольких Activity или даже в качестве дочернего элемента другого фрагмента. Имея это в виду, необходимо создавать фрагмент только с логикой, необходимой для управления собственным пользовательским интерфейсом. Следует избегать зависимости или манипулирования одним фрагментом из другого.

3.2. Создание фрагмента

Фрагмент представляет собой отдельную часть пользовательского интерфейса внутри Activity. Фрагменты можно добавлять или удалять в зависимости от совершаемого действия.

3.2.1. Создание классов фрагмента

Чтобы создать фрагмент, необходимо создать класс-наследник от класса Fragment, находящегося внутри библиотеки AndroidX и переопределить его методы, чтобы реализовать логику приложения, аналогично тому, как при создании класса Activity.

Чтобы создать минимальный фрагмент, определяющий собственную разметку из ресурса, необходимо написать следующее:

```
class ExampleFragment extends Fragment {
    public ExampleFragment() {
        super(R.layout.example_fragment);
    }
}
```

Здесь в родительский класс передается аргумент с разметкой, в результате чего будет создан фрагмент с разметкой example_fragment.

Существуют несколько вариаций фрагментов, представленных в библиотеке AndroidX: DialogFragment и PreferenceFragmentCompat.

Библиотека фрагментов также предоставляет более специализированные базовые классы фрагментов:

DialogFragment

Отображает плавающий диалог. Использование этого класса для создания диалога — хорошая альтернатива использованию вспомогательных методов диалога в классе Activity, поскольку фрагменты автоматически обрабатывают создание и очистку файла Dialog.

PreferenceFragmentCompat

Отображает иерархию Preference объектов в виде списка. Можно использовать PreferenceFragmentCompat для создания экрана настроек для вашего приложения.

3.2.2. Добавление фрагмента через XML

Чтобы добавить фрагмент в разметку Activity используется элемент FragmentContainerView. Вот пример макета активности, содержащего один FragmentContainerView:

```
<!-- res/layout/example_activity.xml -->
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.example.ExampleFragment" />
```

Атрибут android:name указывает имя класса Fragment для создания экземпляра. Когда макет Activity используется приложением, создается конкретный фрагмент, вызывается метод onInflate() для вновь созданного фрагмента и FragmentTransaction создается для добавления фрагмента в FragmentManager.

3.2.3. Добавить фрагмент программно

Чтобы программно добавить фрагмент в макет вашей активности, макет должен включать в себя FragmentContainerView, который будет служить контейнером фрагментов, как показано в следующем примере:

```
<!-- res/layout/example_activity.xml -->
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

В отличие от XML-подхода, атрибут android:name здесь не используется, поэтому фрагмент автоматически не создается. Вместо этого используется FragmentTransaction для создания экземпляра фрагмента и добавления его в макет действия.

Во время выполнения логики Activity можно совершать транзакции фрагментов, такие как добавление, удаление или замена фрагмента. В Activity, у которого в родителях есть FragmentActivity можно получить экземпляр FragmentManager, который можно использовать для создания файла FragmentTransaction. Затем можно создать экземпляр фрагмента в методе onCreate() Activity, используя FragmentTransaction.add(), передав ViewGroup идентификатор контейнера в макете и класс фрагмента, который необходимо добавить, а затем зафиксировать транзакцию, как показано в следующем примере:

```
public class ExampleActivity extends AppCompatActivity {
    public ExampleActivity() {
        super(R.layout.example_activity);
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState == null) {
```

```
            getSupportFragmentManager().beginTransaction()
                .setReorderingAllowed(true)
                .add(R.id.fragment_container_view,
ExampleFragment.class, null)
                .commit();
        }
    }
}
```

Примечание. **Всегда** необходимо использовать setReorderingAllowed(true) при выполнении FragmentTransaction.

Следует обратить внимание, что в предыдущем примере транзакция фрагмента создается только тогда, когда savedInstanceState передается значение null. Это делается для того, чтобы фрагмент добавлялся только один раз при первом создании Activity. При изменении конфигурации и пересоздании Activity savedInstanceState больше не null, и фрагмент не нужно добавлять второй раз, так как фрагмент автоматически восстанавливается из файла savedInstanceState.

Если фрагменту требуются некоторые начальные данные, аргументы могут быть переданы при помощи указания Bundle в вызове FragmentTransaction.add(), как показано ниже:

```
public class ExampleActivity extends AppCompatActivity {
    public ExampleActivity() {
        super(R.layout.example_activity);
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState == null) {
            Bundle bundle = new Bundle();
            bundle.putInt("some_int", 0);

            getSupportFragmentManager().beginTransaction()
                .setReorderingAllowed(true)
                .add(R.id.fragment_container_view,
ExampleFragment.class, bundle)
                .commit();
        }
    }
}
```

Затем аргументы Bundle можно получить во фрагменте, вызвав Bundle.getArguments(), и для извлечения каждого аргумента можно использовать соответствующие методы получения. Например, таким образом можно получить добавленный выше аргумент, к примеру, в методе onCreate() фрагмента:

```
Bundle bundle = this.getArguments();
if (bundle != null) {
    int myInt = bundle.getInt("some_int", <defaultValue>);
}
```

3.3. Жизненный цикл фрагмента

Каждый экземпляр Fragment имеет свой жизненный цикл. Когда пользователь перемещается по приложению и взаимодействует с ним, фрагменты проходят через различные состояния в своем жизненном цикле.

Управления жизненным циклом Fragment реализует класс LifecycleOwner, предоставляя объект Lifecycle, к которому можено получить доступ через getLifecycle() метод.

Каждое возможное состояние Lifecycle представлено в перечислении(enum) Lifecycle.State:

- INITIALIZED;
- CREATED;
- STARTED;
- RESUMED;
- DESTROYED.

Создавая Fragment поверх Lifecycle, можно использовать методы и классы, доступные для обработки стадий жизненного цикла с помощью компонентов, учитывающих жизненный цикл. Например, можно отобразить местоположение устройства на экране с помощью компонента. Этот компонент может автоматически запускать слушатель, когда фрагмент становится активным, и останавливаться, когда фрагмент переходит в неактивное состояние.

В качестве альтернативы использованию LifecycleObserver класс Fragment включает методы обратного вызова, соответствующие каждой из стадий жизненного цикла фрагмента. К ним относятся onCreate(), onStart(), onResume(), onPause(), onStop(), и onDestroy().

Представление (view) фрагмента имеет отдельный Lifecycle, который управляется независимо от Lifecycle фрагмента. Фрагменты поддерживают LifecycleOwner для своего представления, доступ к которому можно получить с помощью getViewLifecycleOwner(). Доступ к Lifecycle предствления полезен в ситуациях, когда компонент, поддерживающий жизненный цикл, должен выполнять работу только тогда, когда существует представление фрагмента.

3.3.1. Фрагменты и менеджер фрагментов

При создании экземпляра фрагмента, он находится в состоянии INITIALIZED. Чтобы фрагмент прошел оставшуюся часть своего жизненного цикла, его необходимо добавить в FragmentManager. Объект FragmentManager отвечает за определение состояния, в котором должен находиться фрагмент, а затем переводит его в это состояние.

Помимо жизненного цикла фрагмента, FragmentManager он также отвечает за присоединение фрагментов к хостовой Activity и их отсоединение, когда фрагмент больше не используется. Класс Fragment имеет два метода обратного вызова onAttach()и onDetach(), которые можно переопределить для выполнения логики при возникновении любого из этих событий.

Callback-метод onAttach() вызывается, когда фрагмент был добавлен в FragmentManager и присоединен к его хостовой Activity. В этот момент фрагмент активен, и FragmentManager управляет состоянием его жизненного цикла. На данном этапе такие методы, как findFragmentById() FragmentManager’а возвращают этот фрагмент.

onAttach()всегда вызывается перед любым изменением состояния жизненного цикла .

Callback-метод onDetach() вызывается, когда фрагмент был удален из FragmentManager и отсоединен от своей хостовой Activity. Фрагмент больше не активен и больше не может быть получен с помощью метода findFragmentById().

onDetach()всегда вызывается после любого изменения состояния жизненного цикла.

Обратите внимание, что эти обратные вызовы не связаны с методами FragmentTransaction attach() и detach().

Внимание: Необходимо избегать повторного использования экземпляров Fragment после их удаления из FragmentManager. Хотя фрагмент обрабатывает собственную очистку внутреннего состояния, можно непреднамеренно перенести его состояние в повторно используемый экземпляр.

3.3.2. Состояния жизненного цикла фрагмента и обратные вызовы

При определении состояния жизненного цикла фрагмента FragmentManager необходимо учитывать следующее:

- Максимальное состояние фрагмента определяется его FragmentManager. Фрагмент не может выйти за пределы состояния своего FragmentManager.
- Можно установить максимальное состояние жизненного цикла для фрагмента, используя setMaxLifecycle(), как часть FragmentTransaction.
- Состояние жизненного цикла фрагмента никогда не может быть больше, чем у его родителя. Например, родительский фрагмент или Activity должны быть активированы до его дочерних фрагментов. Точно так же дочерние фрагменты должны быть остановлены до их родительского фрагмента или Activity.

Внимание: необходимо избегать использования <fragment> тега для добавления фрагмента с помощью XML, так как тег <fragment> позволяет фрагменту выйти за пределы состояния его FragmentManager'a. Вместо этого всегда следует использовать FragmentContainerView для добавления фрагмента с помощью XML.

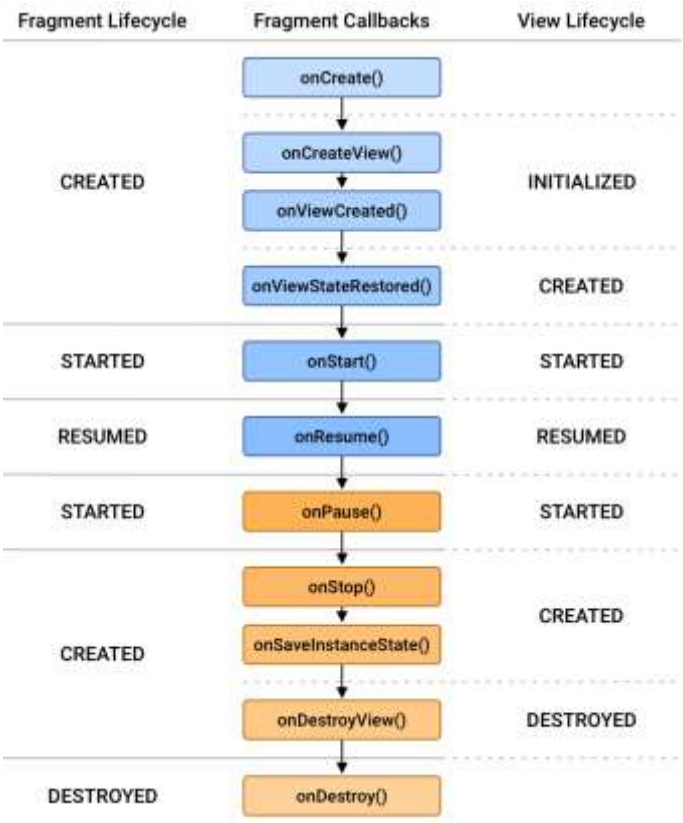

Рисунок 3.2. Состояния фрагмента Lifecycleи их отношение как к обратным вызовам жизненного цикла фрагмента, так и к представлению фрагмента Lifecycle.

На рис. 3.2 показано каждое из состояний фрагмента Lifecycle и их связь как с обратными вызовами жизненного цикла фрагмента, так и с представлением фрагмента Lifecycle.

По мере того, как фрагмент проходит свой жизненный цикл, он перемещается вверх и вниз по своим состояниям. Например, фрагмент, добавленный в начало backstack'a, перемещается вверх от CREATED к STARTED к RESUMED. И наоборот, когда фрагмент извлекается из backstack', он перемещается вниз по этим состояниям, переходя от RESUMED к STARTED к CREATED и наконец DESTROYED.

3.3.2.1. Восходящие переходы состояний

При перемещении вверх по состояниям жизненного цикла фрагмент сначала вызывает соответствующий callback жизненного цикла для своего нового состояния. Как только этот обратный вызов завершится, релевантное значение Lifecycle.Event передается наблюдателям Lifecycle фрагмента, за которым следует Lifecycle представления фрагмента, если оно было создано.

Fragment CREATED

Когда фрагмент достигает состояния CREATED, он добавлен в FragmentManager и onAttach() метод уже был вызван.

Это было бы подходящим местом для восстановления любого сохраненного состояния, связанного с самим фрагментом, через SavedStateRegistry. Обратите внимание, что представление фрагмента еще не создано в это время, и любое состояние, связанное с представлением фрагмента, должно быть восстановлено только после создания представления.

Этот переход вызывает callback onCreate(). Он также получает savedInstanceState аргумент класса Bundle, содержащий любое состояние, ранее сохраненное с помощью onSaveInstanceState(). Обратите внимание, что savedInstanceState имеет null значение при первом создании фрагмента, но всегда не равно нулю для последующих воссозданий, даже если вы не переопределяете onSaveInstanceState().

Fragment CREATED и View INITIALIZED

Представление фрагмента Lifecycle создается только тогда, когда во Fragment предоставляется действительный экземпляр View. В большинстве случаев можно использовать конструкторы фрагментов, которые принимают @LayoutId, что автоматически создает представление в нужное время. Вы

также можете переопределить onCreateView() , чтобы программно создать представление фрагмента.

Если и только если представление фрагмента создано с ненулевым значением View, это View установлено для фрагмента и может быть получено с помощью getView(). Затем обновляется getViewLifecycleOwnerLiveData() новый вид, INITIALIZED LifecycleOwner соответствующий представлению фрагмента. Обратный вызов жизненного цикла onViewCreated() также вызывается в это время.

Это подходящее место для настройки начального состояния представления.

Fragment и View CREATED

После того, как представление фрагмента было создано, предыдущее состояние представления, если оно было, восстанавливается, а затем Lifecycle представления перемещается в CREATED состояние. Владелец жизненного цикла представления также передает ON_CREATE событие своим наблюдателям. Здесь вы должны восстановить любое дополнительное состояние, связанное с представлением фрагмента.

Этот переход также вызывает callback onViewStateRestored().

Fragment и View STARTED

Настоятельно рекомендуется привязывать компоненты, поддерживающие жизненный цикл, к STARTED состоянию фрагмента, так как это состояние гарантирует доступность представления фрагмента, если оно было создано, и безопасность выполнения операций FragmentTransaction над дочерним FragmentManager'ом фрагмента. Если представление фрагмента не равно нулю, Lifecycle представления фрагмента перемещается в состояние STARTED сразу после перемещения состояния фрагмента в Lifecycle.STARTED.

Когда фрагмент переходит в состояние STARTED, вызывается обратный вызов onStart().

Fragment и View RESUMED

Когда фрагмент виден, все эффекты Animator и Transition завершены, и фрагмент готов для взаимодействия с пользователем. Lifecycle фрагмента переходит в состояние RESUMED, и вызывается обратный вызов onResume().

Переход к состоянию RESUMED является подходящим сигналом, указывающим, что теперь пользователь может взаимодействовать с вашим фрагментом. Фрагменты, которые не находятся в состоянии RESUMED не

должны вручную устанавливать фокус на своих представлениях или пытаться обрабатывать ввод данных.

3.3.2.2. Переходы состояний вниз

Когда фрагмент перемещается вниз к более низкому состоянию жизненного цикла, релевантное значение Lifecycle.Event передается наблюдателям Lifecycle представления фрагмента, если оно создано, следуя за Lifecycle фрагмента. После активации события жизненного цикла фрагмента этот фрагмент вызывает соответствующий обратный вызов жизненного цикла.

Fragment и View STARTED

Когда пользователь начинает покидать фрагмент, и пока фрагмент все еще виден, Lifecycle фрагмента и его представления возвращаются в состояние STARTED и передают событие ON_PAUSE своим наблюдателям. Затем фрагмент вызывает свой обратный вызов onPause().

Fragment и View CREATED

Как только фрагмент больше не виден, Lifecycle фрагмента и его представления перемещаются в состояние CREATED и передают событие ON_STOP своим наблюдателям. Этот переход состояния инициируется не только остановкой родительского Activity или фрагмента, но и сохранением состояния родительским Activity или фрагментом. Такое поведение гарантирует, что событие ON_STOP будет вызвано до сохранения состояния фрагмента. Это делает событие ON_STOP последней точкой, в которой безопасно выполнять FragmentTransaction над дочерним элементом FragmentManager.

Как показано на рис. 3.3, порядок обратного вызова onStop() и сохранения состояния onSaveInstanceState() различается в зависимости от уровня API. Для всех уровней API до API 28 onSaveInstanceState()вызывается до onStop(). Для уровней API 28 и выше порядок вызовов обратный.

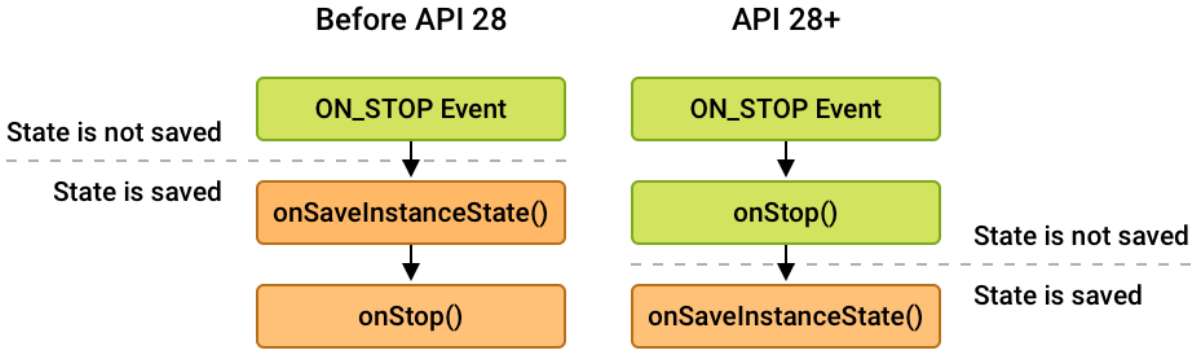

Рисунок 3.3. Различия в порядке вызовов для onStop()и onSaveInstanceState().

Fragment CREATED и View DESTROYED

После завершения всех завершающих анимаций и транзакций и отсоединения представления фрагмента от окна, Lifecycle представления фрагмента перемещается в состояние DESTROYED и передает событие ON_DESTROY своим наблюдателям. Затем фрагмент вызывает свой обратный вызов onDestroyView(). В этот момент представление фрагмента достигло конца своего жизненного цикла и getViewLifecycleOwnerLiveData() возвращает null значение.

На этом этапе все ссылки на представление фрагмента должны быть удалены, что позволит удалить представление фрагмента сборщиком мусора.

Fragment DESTROYED

Если фрагмент удаляется или уничтожается FragmentManager, Lifecycle фрагмента перемещается в состояние DESTROYED и отправляет событие ON_DESTROY своим наблюдателям. Затем фрагмент вызывает свой обратный вызов onDestroy(). В этот момент фрагмент достиг конца своего жизненного цикла.

3.4. Передача данных между фрагментами

В некоторых случаях может понадобиться передать одноразовое значение между двумя фрагментами или между фрагментом и его хостовой Activity. Например, может существовать фрагмент, который считывает QR-коды, передавая данные предыдущему фрагменту. В Fragment версии 1.3.0 и выше каждый FragmentManager реализует FragmentResultOwner. Это означает, что FragmentManager может действовать как центральное хранилище результатов исполнения фрагментов. Это изменение позволяет компонентам взаимодействовать друг с другом, устанавливая результаты фрагментов и ожидая получение слушателем этих результатов, не требуя, чтобы эти компоненты имели прямые ссылки друг на друга.

3.4.1. Передача результатов между фрагментами

Чтобы передать данные обратно к фрагменту A из фрагмента B, сначала необходимо установить слушатель результатов на фрагмент A, который получает результат. После этого необходимо вызвать метод setFragmentResultListener() FragmentManager’a фрагмента A, как показано в следующем примере:

```
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

getParentFragmentManager().setFragmentResultListener("requestKey
", this, new FragmentResultListener() {
        @Override
        public void onFragmentResult(@NonNull String requestKey,
@NonNull Bundle bundle) {
            // We use a String here, but any type that can be
put in a Bundle is supported
            String result = bundle.getString("bundleKey");
            // Do something with the result
        }
    });
}
```

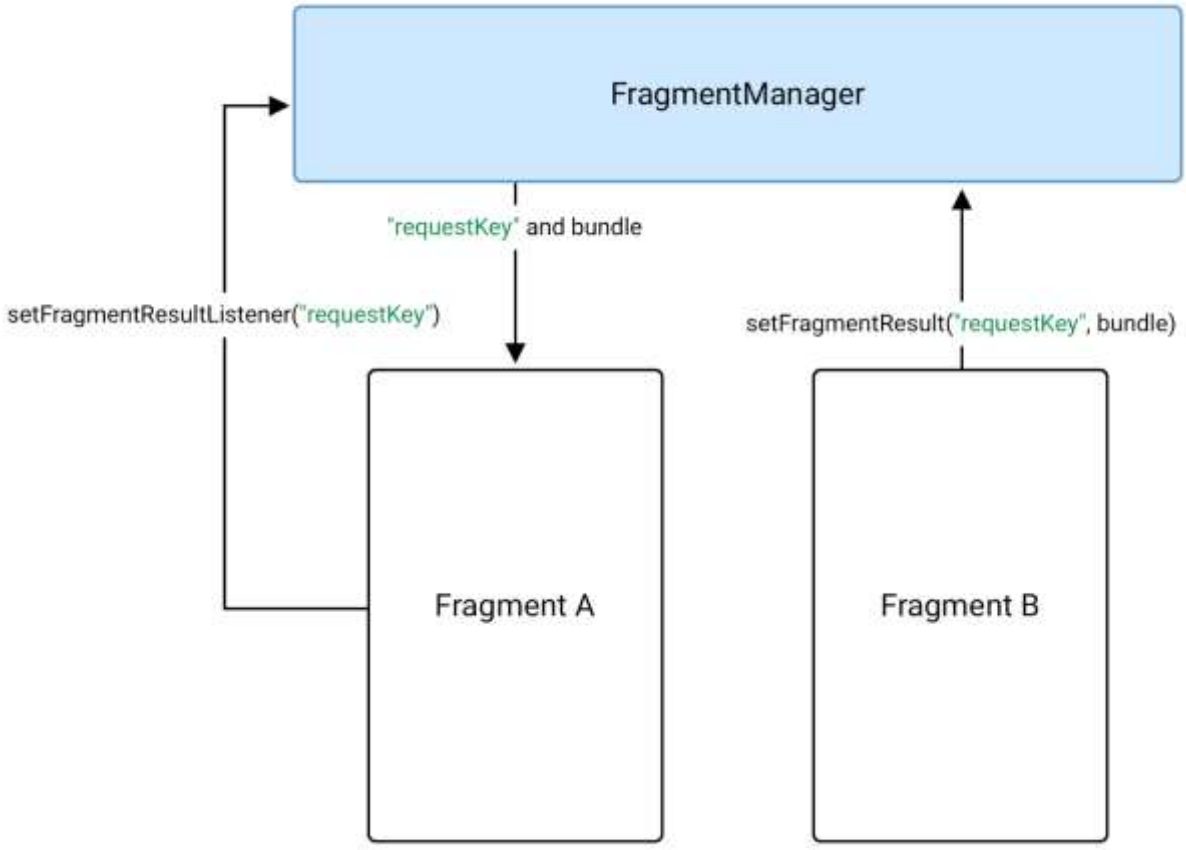

Рисунок 3.4. Фрагмент B отправляет данные фрагменту A с помощью файла FragmentManager.

Во фрагменте B, фрагменте, отдающем результат выполнения, необходимо получить результат того же FragmentManager используя тот же requestKey. Это можно сделать с помощью setFragmentResult() API:

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Bundle result = new Bundle();
        result.putString("bundleKey", "result");
        getParentFragmentManager().setFragmentResult(
            "requestKey", result);
    }
});
```

Затем фрагмент A получает результат и выполняет обратный вызов слушателя, как только фрагмент будет в состоянии STARTED.

Может существовать только один слушатель и результат для данного ключа. Если при помощи setFragmentResult() один и тот же ключ задается более одного раза, и если слушатель не находится в состоянии STARTED, система заменяет все ожидающие результаты обновленным результатом. Если результат устанавливается без соответствующего слушателя для его получения, результат сохраняется в до тех пор, пока во FragmentManager не будет установлен слушатель с тем же ключом. Как только слушатель получает результат и запускает обратный вызов onFragmentResult(), результат очищается. Такое поведение имеет два основных последствия:

- Фрагменты в backstack’е не получают результатов до тех пор, пока они не будут извлечены и не будут в состоянии STARTED.
- Если состояние фрагмента равно STARTED, и он ожидает появление результата, обратный вызов слушателя запустится немедленно, при установке результата.

Примечание. Поскольку результаты фрагмента хранятся на уровне FragmentManager, фрагмент должен быть подключен к вызову setFragmentResultListener() или setFragmentResult() с родительским FragmentManager.

3.4.2. Передача результатов между родительским и дочерним фрагментами

Чтобы передать результат из дочернего фрагмента в родительский, родительский фрагмент должен использовать getChildFragmentManager() вместо getParentFragmentManager() при вызове setFragmentResultListener().

```
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // We set the listener on the child fragmentManager
    getChildFragmentManager()
        .setFragmentResultListener("requestKey", this, new
FragmentResultListener() {
            @Override
            public void onFragmentResult(@NonNull String
requestKey, @NonNull Bundle bundle) {
                String result = bundle.getString("bundleKey");
                // Do something with the result
            }
        });
}
```

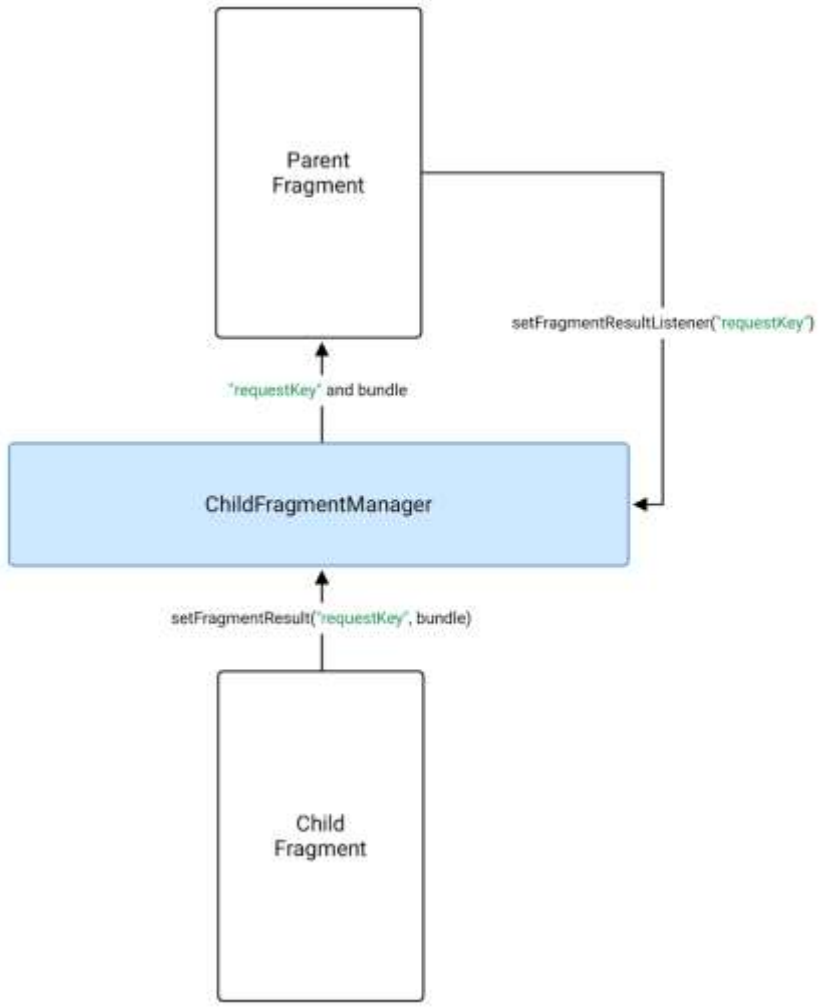

Рисунок 3.5. Дочерний фрагмент может использовать FragmentManager для отправки результата своему родителю.

Дочерний фрагмент устанавливает результат своему FragmentManager'у. Затем родитель получает результат, когда фрагмент переходит в состояние STARTED:

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Bundle result = new Bundle();
        result.putString("bundleKey", "result");
        // The child fragment needs to still set the result on
its parent fragment manager

getParentFragmentManager().setFragmentResult("requestKey",
result);
    }
});
```

3.4.3. Получение результатов в хостовой Activity

Чтобы получить результат фрагмента в хостовом Activity, необходимо установить слушатель результатов в менеджере фрагментов с помощью getSupportFragmentManager().

```
class MainActivity extends AppCompatActivity {
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

getSupportFragmentManager().setFragmentResultListener("requestKe
y", this, new FragmentResultListener() {
            @Override
            public void onFragmentResult(@NonNull String
requestKey, @NonNull Bundle bundle) {
                // We use a String here, but any type that can
be put in a Bundle is supported
                String result = bundle.getString("bundleKey");
                // Do something with the result
            }
        });
    }
}
```

Задание

1. Перенести разметки и логику работы экранов из практической работы №2, которые были представлены на Activity во Fragment. Разметка экранов должна остаться неизменной.
2. При помощи отображения Toast и сообщений в Log обработать переходы состояний фрагментов в соответствии с их жизненным циклом.
3. Согласно выбранной предметной области организовать обмен данными между несколькими Fragment’ами.