Лабораторная работа № 4 по теме:
«Создание и вызов Activity. Жизненный цикл Activity»

Теоретическая часть

В предыдущих работах создавались приложения, содержащие всего одно окно активити, но большинство реальных приложений состоит из множества активити. Каждое активити в приложении независимо от других. При открытии нового активити работа предыдущего приостанавливается, и оно вносится и сохраняется в стек активити.

Активити используются для взаимодействия с пользователем. Класс Activity отвечает за создание окна, в котором можно разместить свой пользовательский интерфейс с помощью метода setContentView (View). Хотя активити часто представляются пользователю как полноэкранные окна, их также можно использовать и другими способами: как плавающие окна, в многооконном режиме или встроенными в другие окна.

В отличие от приложений в большинстве других систем, у приложений Android нет единственной точки входа для запуска всего приложения, аналогичной, например, функции main () в С-подобных языках программирования. Программа может иметь несколько точек входа. Активити могут запускать независимо друг от друга.

Как правило, одно активити в приложении указывается в качестве основного, которое появляется первым, когда пользователь запускает приложение. Затем каждое активити может запускать другое для выполнения различных действий. Сторонние приложения могут запускать произвольные активити вашего приложения, если выдать им соответствующие разрешения.

Для запуска компонентов приложения, в том числе активити, используются асинхронные сообщения, называющиеся намерениями (объекты Intent), которые определяют имя запускаемого компонента и, при необходимости, могут передавать набор параметров, определяющих

условия запуска этого компонента. В системе Android все коммуникации между приложениями, службами и отдельными компонентами происходят только с использованием объектов Intent.

Компоненты приложения имеют жизненный цикл — начало, когда Android инициализирует их, активный период работы, неактивный период, когда они бездействуют, и конец, когда компоненты уничтожаются и освобождают ресурсы для запуска других компонентов.

Особенность жизненного цикла приложения Android состоит в том, что система управляет большой частью цикла. Все приложения Android выполняются только в пределах своего собственного процесса. Все выполняющиеся процессы наблюдаются системой, и в зависимости от приоритета компонента и от того, насколько интенсивно компонент (например, активити) работает, Android может закончить его работу и передать освободившиеся ресурсы другому компоненту из этого процесса.

Решая, должен ли компонент быть закрыт, Android принимает во внимание активность пользователя в работе с этим компонентом, использование памяти и т.д.

Чтобы использовать активити в приложении, требуется зарегистрировать информацию о них в манифесте приложения и соответствующим образом управлять жизненными циклами активити.

При создании проекта в Android Studio с пустым активити (англ. Empty Activity) по умолчанию создаётся класс MainActivity, наследуемый от AppCompatActivity. AppCompatActivity ― это непрямой наследник класса Activity, это базовый класс для активити, который позволяет использовать некоторые из новых функций платформы на старых устройствах Android.

Конфигурирование манифеста приложения

Для объявления активити в манифесте приложения (AndroidManifest.xml) требуется добавить элемент <activity> в качестве дочернего элемента <application> (листинг 1).

Листинг 1. Объявление активити в манифесте.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.activitiesapp">
    <application  ...>
        <activity android:name=".MainActivity">
            ...
        </activity>
    </application>
</manifest>
```

Единственный обязательный атрибут для элемента <activity> — это android:name, который указывает имя класса активити. Также можно добавить атрибуты, определяющие характеристики активити, такие как: название, значок или тема оформления. После публикации приложения изменять названия активити нельзя, так как это может привести к ошибкам.

Внутри элемента <activity> могут быть объявлены фильтры намерений (Intent-фильтры). Фильтры намерений — это очень мощная функция платформы Android. Они предоставляют возможность запускать активити на основе не только явного запроса, но и неявного. Например, явный запрос может указать системе: «Запустить активити отправки электронной почты в приложении Gmail». А неявный запрос сообщает системе: «Запустить активити отправки электронной почты из любого приложения, которое может выполнить эту работу», после чего система спросит пользователя, какое приложение следует использовать для выполнения задачи.

Элемент <intent-filter> может располагаться в манифесте внутри элемента <activity> (а также внутри элементов <service> и <receiver>, объявляющих компоненты сервис и приемник широковещательных сообщений соответственно). Определение элемента <intent-filter> включает элемент <action> и необязательные элементы <category> и <data>. С помощью этих элементов указывается тип намерения, на которое может реагировать данное активити. В листинге 2 приведён пример фильтра намерений, задающего что это активити будет «главным» и будет запускаться при запуске приложения.

Листинг 2. Объявление фильтра намерений.

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

В листинге 3 показано, как настроить фильтр намерений активити, которое отправляет текстовые данные и получает для этого запросы от других активити.

Листинг 3. Объявление фильтра намерений.

```
<activity android:name=".MainActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

В этом примере элемент <action> указывает, что это активити отправляет данные. Объявление элемента <category> как DEFAULT позволяет активити получать запросы на запуск. Элемент <data> указывает тип данных, которые может отправлять это активити. В листинг 4 показано, как вызвать активити, описанное выше.

Листинг 4. Работа с объектом Intent.

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.setType("text/plain");
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
startActivity(sendIntent);
```

В элементе <activity> может быть указан атрибут android:exported. Он позволяет или запрещает запускать это активити другим приложениям. По умолчанию значение этого атрибута равно false, если в данном активити не объявлен фильтр намерений, и другие приложения не могут запустить это активити. Но если у активити есть фильтр намерений, то по умолчанию

значение android:exported="true". Поэтому, если нужно запретить запуск через сторонние приложения, то следует явно указывать значение false.

Внутри элемента <activity> с помощью атрибута android:permission можно контролировать, какие разрешения должны быть у приложений, чтобы они могли запускать это активити.

Например, если ваше приложение хочет использовать некое активити приложения с именем SocialApp для публикации постов в социальных сетях, в приложении SocialApp должно быть определено разрешение, которое должны иметь вызывающие его приложения:

<activity android:name="...."

android:permission="com.google.socialapp.permission.SHARE_POST"/>

Затем, чтобы разрешить вызов активити из SocialApp, в вашем приложении должно быть определено разрешение, установленное в манифесте SocialApp, внутри элемента <uses-permission>, вложенного в корневой элемент <manifest>:

<uses-permission

android:name="com.google.socialapp.permission.SHARE_POST"/>

Запуск Activity с использованием явных и неявных намерений

Как отмечалось раннее, объекты Intent могут быть разделены на две группы:

- Явное намерение — определяет целевой компонент по имени. Явный Intent обычно используют для сообщений внутри приложения, например, когда одно активити запускает другое из этого приложения.
- Неявное намерение — не называет адресата. Неявные Intent обычно используются для запуска компонентов других приложений.

Для запуска активити с помощью явного намерения нужно в объект Intent передать имя этого активити. Имя устанавливается методами setComponent(), setClass() или setClassName().

Далее, чтобы запустить активити, объект Intent передают в метод Context.startActivity(). Этот метод принимает единственный параметр — объект Intent, описывающий активити, который будет запускаться (листинг 5). В метод setClass () передаётся контекст приложения и класс активити, которое надо запустить.

Листинг 5. Запуск активити с помощью явного намерения.

```
Intent intent = new Intent();
intent.setClass(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

В листинге 6 приведён пример запуска активити стороннего приложения с помощью явного намерения. В метод setClassName () передаётся имя пакета приложения и строка с именем активити. Приведённый код будет работать в случае, если у целевого активити в манифесте явно объявлен атрибут android:exported="true" или есть фильтр намерений, а атрибут не указан, или в случае, если настроены разрешения.

Листинг 6. Запуск активити с помощью явного намерения.

```
Intent intent = new Intent();
intent.setClassName("com.example.resources",
"com.example.resources.MainActivity");
startActivity(intent);
```

Системные компоненты запускаются с помощью неявных намерений. Для вызова системных компонентов в классе Intent определено множество констант Standard Activity Actions для запуска стандартных активити. Например, действие ACTION_DIAL запускает активити, представляющее собой окно для набора телефонного номера. Это окно можно вызвать следующим образом (листинг 7):

Листинг 7. Запуск системного активити с помощью неявного намерения.

```
Intent intent = new Intent(Intent.ACTION_DIAL);
startActivity(intent);
```

В случае с действием ACTION_DIAL, если не передавать никакой номер для совершения звонка автоматически, то никакие разрешения приложению не потребуются, если же сразу указать номер (листинг 8), по которому будет совершен звонок, то потребуется добавить разрешение.

Листинг 8. Запуск системного активити с помощью неявного намерения с передачей параметров.

```
Intent dialIntent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:" +
"123456789"));
startActivity(dialIntent);
```

Кроме класса Intent, в других классах также определены действия для вызова специализированных активити. Например, в классе MediaStore и MediaStrore.Audio.Media определены константы действий для запуска окон с медиаплеером, поиска музыки и записи с микрофона. Также действия определены в классах DownloadManager для запуска активити, управляющего загрузкой файлов через Интернет, и т. д.

Работа с разрешениями

Начиная с первой версии Android и вплоть до выхода Android 6.0 (уровень API 23) для добавления разрешения было достаточно прописать его в манифесте приложения. Например, разрешение для возможности совершать звонки объявляется в манифесте следующим образом:

<uses-permission android:name="android.permission.CALL_PHONE" />

При установке приложений система запрашивала разрешения, необходимые приложениям для работы. Таких разрешений могло потребоваться довольно много.

Начиная с Android 6.0 механизм работы с разрешениями изменился. В новой модели разрешений пользователи могут напрямую управлять разрешениями приложений во время выполнения. Эта модель дает пользователям улучшенный контроль над разрешениями, оптимизирует процессы установки и автоматического обновления для разработчиков

приложений. Пользователи могут предоставлять или отзывать разрешения индивидуально для установленных приложений.

В приложениях, предназначенных для Android 6.0 (уровень API 23) или выше, нужно обязательно проверять и запрашивать разрешения во время выполнения. Для определения предоставлено ли приложению разрешение используется метод checkSelfPermission (). Чтобы запросить разрешение, вызывается метод requestPermissions ().

Разрешения делятся на два основных типа:

- обычные (normal);
- опасные (dangerous).

Обычные разрешения будут получены приложением при установке автоматически, без подтверждения от пользователя. В дальнейшем отозвать их у приложения невозможно. Опасные разрешения должны быть запрошены в процессе работы приложения и в любой момент могут быть отозваны. В таблице 1 приведены некоторые опасные и обычные разрешения.

Таблица 1. Опасные и обычные разрешения.

Разрешение	**Тип**	**Описание**
ACCESS_BACKGROUND _LOCATION	**опасное**	Позволяет приложению получать доступ к местоположению в фоновом режиме.
ACCESS_NETWORK_ STATE	обычное	Позволяет приложению получать доступ к информации о сетях.
ACCESS_WIFI_STATE	обычное	Позволяет приложению получать доступ к информации о сетях Wi-Fi.
ANSWER_PHONE_ CALLS	**опасное**	Приложение сможет отвечать на входящий телефонный звонок.
CALL_PHONE	**опасное**	Позволяет приложению инициировать телефонный звонок, минуя пользовательский интерфейс Dialer, в котором пользователь мог бы подтвердить звонок.
CAMERA	**опасное**	Требуется для доступа к камере.
READ_CONTACTS	**опасное**	Позволяет приложению читать данные контактов пользователя.
SET_WALLPAPER	обычное	Позволяет приложению устанавливать обои.
USE_BIOMETRIC	обычное	Приложение сможет использовать биометрические датчики, поддерживаемые устройством.

Для доступа к компоненту, требующему разрешения типа «опасное», необходимо сначала проверить, предоставил ли пользователь это разрешение. Для этого из класса Activity вызывается метод checkSelfPermission (). Он вернет константы PERMISSION_GRANTED, если разрешение предоставлено, или PERMISSION_DENIED, если разрешение отклонено. Если проверка указывает, что разрешение не было предоставлено, необходимо явно запросить его, вызвав метод requestPermissions ().

Поскольку этот метод будет взаимодействовать с пользователем, он представляет собой асинхронный запрос. Чтобы получить уведомление об ответе, необходимо переопределить метод onRequestPermissionsResult () в классе Activity. Этот метод принимает целочисленный код запроса, запрошенное разрешение и константу PackageManager. PERMISSION_GRANTED или PackageManager.PERMISSION_DENIED.

Для каждого разрешения, для которого будет выполняться проверка необходимо объявить целочисленную константу кода запроса, например:

private static final int CALL_PHONE_PERMISSION_CODE = 100;

В листинг 9 приведён пример запроса и проверки разрешения CALL_PHONE. В методе checkPermission () осуществляется проверка на наличие разрешения, если разрешение отсутствует вызывается запрос на предоставление разрешения. Если разрешение есть, вызывается метод callPhone (), который запускает активити для совершения звонка с помощью неявного намерения с передачей номера телефона в качестве параметра.

Листинг 9. Запрос и проверка разрешения.

```
public class MainActivity extends AppCompatActivity {
    private static final int CALL_PHONE_PERMISSION_CODE = 100;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        checkPermission(Manifest.permission.CALL_PHONE,
        CALL_PHONE_PERMISSION_CODE);
        ...
```

```
}

public void callPhone() {
    Intent dialIntent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:" +
"123456789"));
    startActivity(dialIntent);

}
public void checkPermission(String permission, int requestCode)
{
    if (ContextCompat.checkSelfPermission(MainActivity.this, permission)
== PackageManager.PERMISSION_DENIED) {
        ActivityCompat.requestPermissions(MainActivity.this, new String[]
{ permission }, requestCode);
    }
    else {
        callPhone();
    }
}

@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String[] permissions,
                                       @NonNull int[] grantResults)
{
    super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
    if (requestCode == CALL_PHONE_PERMISSION_CODE) {
        if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            Toast.makeText(MainActivity.this,"Разрешение
предоставлено",Toast.LENGTH_SHORT).show();
            callPhone();
        }
        else {
            Toast.makeText(MainActivity.this,"Разрешение
отклонено",Toast.LENGTH_SHORT).show();
        }
    }
}

}
```

В файле манифеста приложения должно быть объявлено соответствующее разрешение:

<uses-permission android:name="android.permission.CALL_PHONE" />

При первом вызове checkPermission (), пользователь увидит на экране запрос с двумя кнопками: «Разрешить» и «Отклонить» (рис. 1).

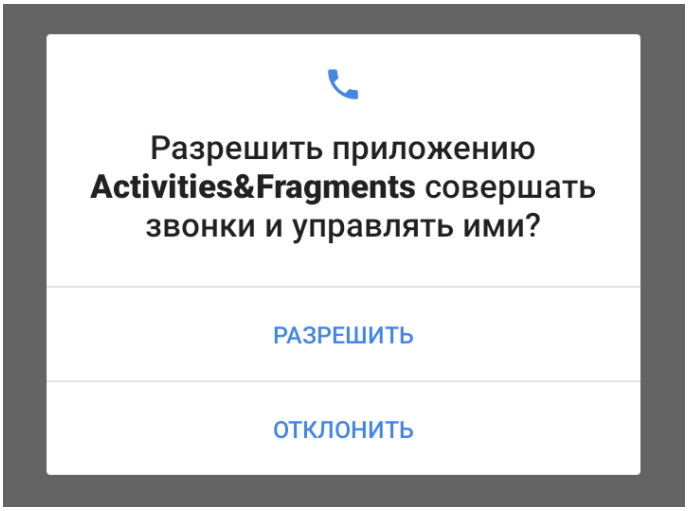

Рисунок 1. Первый запрос на получение разрешения.

Если отклонить разрешение, то при повторном вызове данного запроса, появится кнопка «Запретить и больше не спрашивать» (рис. 2). Если пользователь нажмет на эту кнопку запросы на разрешения больше появляться не будут. Однако, пользователь сможет включить данное разрешение вручную в окне «О приложении», которое можно открыть, нажав и подержав палец на иконке приложения (рис. 3).

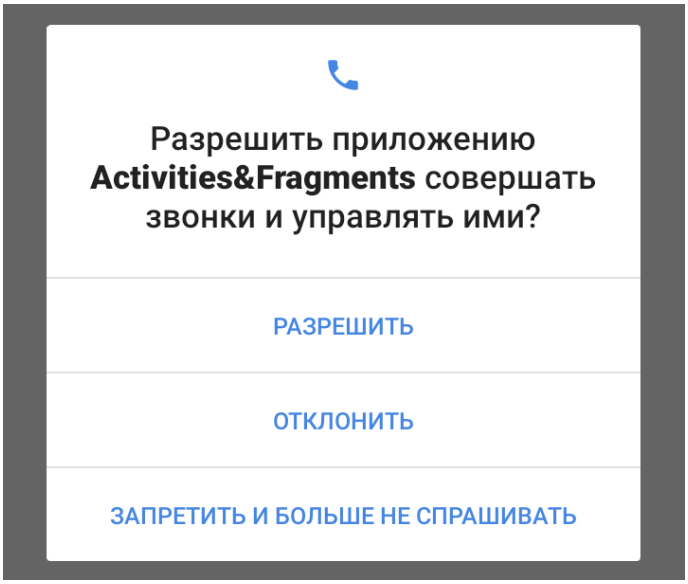

Рисунок 2. Повторный запрос на получение разрешения после отклонения.

Следует учитывать, что разрешения, критически важные для работы приложения, следует запрашивать при запуске приложения, а не настолько важные — можно запрашивать по мере возникновения необходимости в них.

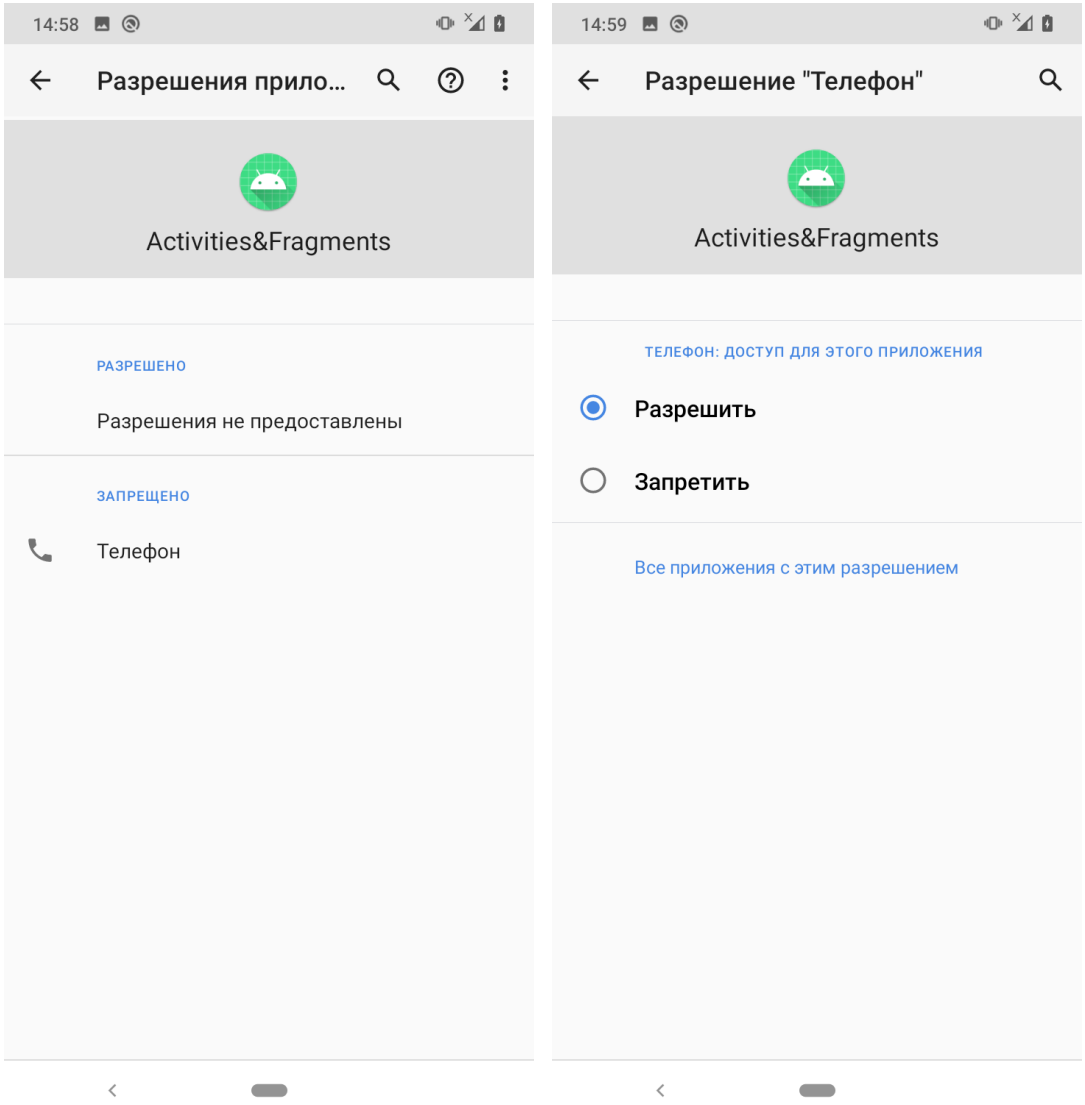

Рисунок 3. Добавление разрешений вручную.

Обычно рекомендуется также предоставлять пользователю информацию о том, зачем нужно разрешение. Для этого из класса Activity вызывается метод shouldShowRequestPermissionRationale (). Этот метод вызывается, если метод ContextCompat.checkSelfPermission () возвращает PERMISSION_DENIED.

Если метод shouldShowRequestPermissionRationale () возвращает true, следует отобразить образовательный интерфейс, в котором описывается, почему функция, которую пользователь хочет включить, требует определенного разрешения.

Жизненный цикл Activity

В течение своего жизненного цикла активити проходит через несколько состояний. Для обработки переходов между этими состояниями используется серия методов обратного вызова.

Активити в системе организуются в стеки. Когда новое активити запускается, оно обычно помещается в верхнюю часть текущего стека и становится «выполняемым» (состояние running), при этом предыдущее активити всегда остается в стеке активити под ним и не переходит на передний план, пока новое активити не завершится. На экране может быть один или несколько стеков активити.

Активити имеет четыре состояния:

- **Активное или выполняемое** (англ. active, running) ― это активити находится на переднем плане экрана (в наивысшей позиции верхнего стека). Обычно это активити, с которым в данный момент взаимодействует пользователь.
- **Видимое** (англ. visible) ― это активити потеряло фокус, но все еще отображается на экране. Это возможно, если новое не полноразмерное или прозрачное активити запустилось поверх данного активити, другое активити имеет более высокую позицию в многооконном режиме или само активити не может получить фокус ввода в текущем оконном режиме.
- **Остановленное или скрытое** (англ. stopped, hidden) ― это активити, которое полностью скрыто другим активити. Оно по-прежнему сохраняется в стеке, однако больше не видно пользователю. Такие активити часто уничтожаются системой, когда память оказывается востребована другие более приоритетными активити (находящимися выше в стеке).
- **Уничтоженное** (англ. destroyed) ― это активити, которое было удалено для освобождения места в памяти. Когда это активити снова

отображается пользователю, оно должно быть полностью перезапущено и восстановлено до предыдущего состояния.

На рисунке 4 показана диаграмма жизненного цикла активити. Серые прямоугольники представляют методы обратного вызова, которые можно реализовать, чтобы выполнять определенные действия при изменении состояния данного активити.

Рисунок 4. Диаграмма жизненного цикла активити.

Эти методы не вызывают смену состояния активити. Наоборот, смена состояния является триггером, который вызывает эти методы. С помощью этих методов нас уведомляют о смене состояния, и мы можем реагировать соответственно. Рассмотрим эти методы обратного вызова подробнее.

Метод **onCreate ()** вызывается при создании активити. Это единственный обязательный для реализации в классе Activity метод обратного вызова. Внутри этого метода инициализируют основные компоненты активити: например, настраивают интерфейс активити — создают представления, связывают данные со списками и т.д. Что наиболее важно, здесь вызывается метод setContentView (), который определяет макет пользовательского интерфейса (компоновку) активити.

Метод onCreate () принимает один параметр — объект Bundle, содержащий предыдущее состояние активити (если это состояние было сохранено).

Когда onCreate () завершается, следующим обратным вызовом всегда будет onStart (). При выходе из onCreate () активити переходит в состояние «выполняемое» и становится видимым для пользователя.

Метод **onStart ()** содержит те действия, которые требуются для последней подготовки активити к выходу на передний план и получению фокуса ввода. Например, здесь можно начать отрисовку визуальных элементов, запустить анимацию и т.д.

Метод **onResume ()** вызывается непосредственно перед тем, как пользователь сможет начать взаимодействовать с активити. На этом этапе активити находится наверху стека активити и фиксирует все вводимые пользователем данные. Большая часть основных функций приложения реализуется в методе onResume (). За onResume () всегда следует обратный вызов onPause ().

Метод **onPause ()** система вызывает, когда активити теряет фокус и переходит в состояние паузы. Это состояние возникает, когда, например,

пользователь нажимает кнопку «Назад». Когда система вызывает onPause () для вашей активити, это технически означает, что активити все ещё частично видно, но чаще всего это указывает на то, что пользователь покидает активити, и оно вскоре перейдет в состояние остановлено или возобновлено (выполняется).

Активити в состоянии приостановленное (видимое) может продолжать обновлять пользовательский интерфейс, если пользователь ожидает подобного поведения, например, при показе окна навигационной карты или проигрывателя мультимедиа. Даже если такие активити теряют фокус, пользователь ожидает, что их пользовательский интерфейс продолжит обновляться.

Когда активити B запускается перед активити A, метод onPause () будет вызван для A. Активити B не будет создано до тех пор, пока A не вернет onPause (), поэтому в этом методе не следует реализовывать ничего, требующего длительного времени для выполнения.

В этом методе также можно останавливать задачи, которые потребляют существенное количество ресурсов центрального процессора, чтобы как можно быстрее переключиться на следующее активити.

После завершения выполнения onPause () следующим обратным вызовом будет onStop () или onResume ().

Метод **onStop ()** система вызывает, когда активити больше не отображается на экране. Это может произойти из-за того, что активити уничтожается, запускается новое активити или существующее активити переходит в состояние возобновления и перекрывает остановленное активити.

Следующий обратный вызов, который вызывает система, ─ это либо onRestart (), если активити возвращается для взаимодействия с пользователем, либо onDestroy (), если это активити полностью завершается.

Метод **onRestart ()** система вызывает, когда активити в состоянии остановлено собирается перезапуститься. onRestart () восстанавливает состояние активити с момента его остановки. За этим обратным вызовом всегда следует onStart ().

Метод **onDestroy ()** система вызывает до того, как активити будет уничтожено. Этот обратный вызов является последним, который получает активити. onDestroy () обычно реализуется, чтобы гарантировать, что все ресурсы активити будут освобождены, когда активити или содержащий его процесс уничтожается.

Чтобы зафиксировать состояние активити перед его полным уничтожением (onDestroy), в классе, реализующем Activity, необходимо реализовать метод onSaveInstanceState ().

Система передает методу объект Bundle, в который можно записать параметры, динамическое состояние активити как пары имя-значение. Когда активити будет снова вызван, объект Bundle передается системой в качестве параметра в метод onCreate () и в метод onRestoreInstanceState (), который вызывается после onStart (), чтобы один из них или они оба могли установить активити в предыдущее состояние.

Начиная с Android 3.0 (уровень API 11) метод onSaveInstanceState () может быть безопасно вызван после onPause ().

Для приложений, ориентированных на Android 9 (уровень API 28) и выше, метод onSaveInstanceState () всегда будет вызываться после onStop ().

Следует учитывать, что на устройствах с уровнем до Android 3.0 (уровень API 11) при нехватке ресурсов активити может быть уничтожено, минуя методы onStop () и onDestroy (). Поэтому важные данные следовало сохранять в методе onPause (), чтобы никакие данные не потерялись при внезапном уничтожении активити.

Однако начиная с Android 3.0 (уровень API 11) активити не может быть уничтожено до возврата onStop (). Это влияет на то, когда можно безопасно сохранять данные. Теперь важные данные можно сохранять в onStop (), а не в onPause ().

Начиная с Android 10 (уровень API 29) в системе может быть несколько выполняемых активити одновременно в многооконном и многоэкранном режимах, поэтому был введён метод onTopResumedActivityChanged (). Метод onTopResumedActivityChanged () вызывается, когда активити достигает или теряет верхнюю выполняемую позицию в системе.

Этот обратный вызов следует использовать вместо onResume () как указание на то, что активити может попытаться открыть устройства с эксклюзивным доступом, например, камеру. Этот метод всегда вызывается после перехода активити в выполняемое состояние и до его приостановки.

Фрагменты

Организация приложения на основе нескольких активити не всегда оптимальна. Существует большое количество различных устройств под управлением Android (смартфоны, планшеты, телевизоры). И если для мобильных аппаратов с небольшими экранами взаимодействие между разными активити выглядит довольно неплохо, то на больших экранах ─ планшетах, телевизорах, окна активити смотрелись бы плохо в силу большого размера экрана. Из-за этого возникла концепция фрагментов.

Фрагмент существует в контексте активити и имеет свой жизненный цикл, вне активити обособлено он не может существовать. Каждая активити может иметь несколько фрагментов.

Разработчик может объединить несколько фрагментов в одно активити для построения многопанельного пользовательского интерфейса и повторного использования фрагмента в нескольких активити. Фрагмент можно рассматривать как модульную часть активити. Такая часть имеет

свой жизненный цикл и самостоятельно обрабатывает события ввода. Кроме того, ее можно добавить или удалить непосредственно во время выполнения активити. Это нечто вроде вложенной активити, которую можно многократно использовать в различных активити.

Фрагмент всегда должен быть встроен в активити, и на его жизненный цикл напрямую влияет жизненный цикл активити. Например, когда активити приостановлена, в том же состоянии находятся и все фрагменты внутри нее, а когда активити уничтожается, уничтожаются и все фрагменты.

Однако пока активити выполняется, можно манипулировать каждым фрагментом независимо, например, добавлять или удалять их. Когда разработчик выполняет такие транзакции с фрагментами, он может также добавить их в стек переходов назад, которым управляет активити. Каждый элемент стека переходов назад в активити является записью выполненной транзакции с фрагментом. Стек переходов назад позволяет пользователю обратить транзакцию с фрагментом (выполнить навигацию в обратном направлении), нажимая кнопку Назад.

Когда фрагмент добавлен как часть компоновки активити, он находится в объекте ViewGroup внутри иерархии представлений активити и определяет собственную компоновку представлений. Разработчик может вставить фрагмент в компоновку активити двумя способами. Для этого следует объявить фрагмент в файле компоновки как элемент <fragment> или добавить его в существующий объект ViewGroup в коде приложения. Впрочем, фрагмент не обязан быть частью компоновки активити. Можно использовать фрагмент без интерфейса в качестве невидимого рабочего потока активити.

Для создания фрагмента необходимо создать подкласс класса Fragment (или его существующего подкласса). Класс Fragment имеет код, во многом схожий с кодом Activity. Он содержит методы обратного вызова, аналогичные методам Activity, такие как onCreate (), onStart (), onPause () и

onStop (). На практике, если требуется преобразовать существующее приложение Android так, чтобы в нем использовались фрагменты, достаточно просто переместить код из методов обратного вызова активити в соответствующие методы обратного вызова фрагмента.

У фрагментов также есть несколько дополнительных методов обратного вызова жизненного цикла, которые обеспечивают уникальное взаимодействие с Activity для выполнения таких действий, как создание и уничтожение пользовательского интерфейса фрагмента (onAttach (), onCreateView (), onActivityCreated (), onDestroyView (), onDetach ()).

Существует ряд подклассов, которые может потребоваться расширить вместо использования базового класса Fragment:

- FragmentActivity может использовать класс Fragment, чтобы создавать более сложные пользовательские интерфейсы, разбитые на модули, для больших экранов и помогать масштабировать приложение между маленькими и большими экранами.
- DialogFragment ― диалоговое окно. Использование этого класса для создания диалогового окна является хорошей альтернативой вспомогательным методам диалогового окна в классе Activity. Он дает возможность вставить диалоговое окно фрагмента в управляемый Activity стек переходов назад для фрагментов, что позволяет пользователю вернуться к закрытому фрагменту.
- ListFragment ― список элементов, управляемый адаптером, аналогично классу ListActivity. Этот класс предоставляет несколько методов для управления списком представлений, например, метод обратного вызова onListItemClick () для обработки нажатий.
- PreferenceFragmentCompat ― отображение иерархии объектов Preference в виде списка. Этот класс полезен, когда в приложении создается экран «Настройки».

Чтобы создать компоновку для фрагмента, разработчик должен реализовать метод обратного вызова onCreateView (), который система Android вызывает, когда для фрагмента наступает время отобразить свою компоновку. Реализация этого метода должна возвращать объект View, который является корневым в компоновке фрагмента. Чтобы возвратить компоновку из метода onCreateView (), можно использовать объект LayoutInflater.

Задание

Разработайте приложение, состоящие из нескольких активити, между которыми можно переключаться. Добавьте в одно из них элемент TextView, в котором будет прописываться, какие методы обратного вызова активити сработали в процессе жизнедеятельности приложения. Также добавьте в приложение возможность совершать звонок по определённому номеру телефона и, дополнительно, читать книгу контактов.

Контрольные вопросы:

1. Как создать несколько активити в приложении?
2. Есть ли в Android-приложении единственная точка входа?
3. Что такое намерение (объект Intent) в Android-приложении?
4. Что такое фильтр намерений, и где он задаётся?
5. Зачем в манифесте приложения задаются разрешения?
6. Чем отличается явное намерение (Intent) от неявного?
7. Как осуществляется явный и неявный вызов активити?
8. Как работает механизм разрешение на устройствах с Android 6.0 и выше?
9. Какие два основных типа разрешений выделяют?
10. Какие четыре состояния имеет активити?
11. Когда происходит вызов метода onCreate?
12. Когда происходит вызов метода onResume?
13. Когда происходит вызов метода onPause?

14.Когда происходит вызов метода onStop?

15.Когда происходит вызов метода onDestroy?

16.Что такое фрагменты, и для чего они используются?