ПРАКТИЧЕСКАЯ РАБОТА №6

6.1. Уведомления

Уведомление – это сообщение, которое Android отображает за пределами пользовательского интерфейса вашего приложения, чтобы предоставить пользователю напоминания, сообщения от других людей или другую своевременную информацию из вашего приложения. Пользователи могут нажать на уведомление, чтобы открыть ваше приложение или выполнить действие непосредственно из уведомления.

6.1.1. Анатомия уведомлений

Дизайн уведомления определяется системными шаблонами, а ваше приложение определяет содержимое для каждой части шаблона. Некоторые детали уведомления отображаются только в развернутом виде.

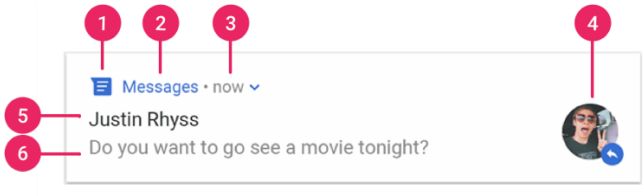

Рисунок 5.1. *Уведомление с основными деталями.*

Наиболее распространенные части уведомления показаны на рис. 5.1. следующим образом:

1. Маленький значок: требуется; устанавливается с помощью setSmallIcon().
2. Название приложения: предоставляется системой.
3. Отметка времени: предоставляется системой, но вы можете переопределить ее с помощью setWhen() или скрыть ее с помощью setShowWhen(false).
4. Большой значок: необязательно; обычно используется только для фотографий контактов. Не используйте его для значка вашего приложения. Установите с помощью setLargeIcon().
5. Заголовок: необязательно; устанавливается с помощью setContentTitle().
6. Текст: необязательно; устанавливается с помощью setContentText().

Настоятельно рекомендуется использовать системные шаблоны для обеспечения надлежащей совместимости дизайна на всех устройствах. При необходимости вы можете создать пользовательский макет уведомлений.

6.1.2. Действия с уведомлениями

Хотя это и не обязательно, рекомендуется, чтобы каждое уведомление открывало соответствующее действие приложения при его нажатии. В дополнение к этому действию уведомления по умолчанию, вы можете добавить кнопки действий, которые завершают задачу, связанную с приложением, из уведомления — часто без открытия действия — как показано на рис. 5.2.

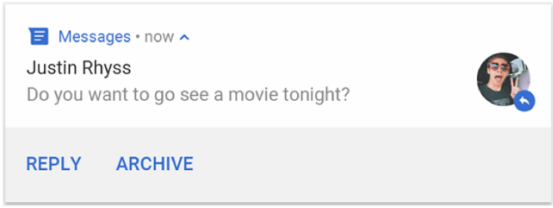

Рисунок 5.2. Уведомление с кнопками действий.

Начиная с Android 7.0 (уровень API 24), вы можете добавить действие для ответа на сообщения или ввода другого текста непосредственно из уведомления.

Начиная с Android 10 (уровень API 29), платформа может автоматически генерировать кнопки действий с предлагаемыми действиями на основе намерений.

6.1.3. Действия на зблокированном устройство

Пользователи могут видеть действия с уведомлениями на экране блокировки устройства. Если действие уведомления заставляет приложение запускать действие или отправлять прямой ответ, пользователи должны разблокировать устройство, прежде чем приложение сможет вызвать это действие уведомления.

На Android 12 (уровень API 31) и выше вы можете настроить действие уведомления таким образом, что устройство должно быть разблокировано, чтобы ваше приложение могло вызвать это действие, независимо от того, какой рабочий процесс запускает действие. Эта опция добавляет дополнительный уровень безопасности к уведомлениям на заблокированных устройствах.

Чтобы потребовать разблокировки устройства до того, как ваше приложение вызовет определенное действие уведомления, перейдите true в setAuthenticationRequired() раздел при создании действия уведомления, как показано в следующем фрагменте кода:

```
val moreSecureNotification = Notification.Action.Builder(...)
    // This notification always requests authentication when
invoked
    // from a lock screen.
    .setAuthenticationRequired(true)
    .build()
```

6.1.4. Каналы уведомлений

Начиная с Android 8.0 (уровень API 26), все уведомления должны быть назначены каналу, иначе они не отображаются. Это позволяет пользователям отключать определенные каналы уведомлений для вашего приложения вместо отключения *всех* ваших уведомлений. Пользователи могут управлять визуальными и звуковыми параметрами для каждого канала из системных настроек Android, как показано на рисунке 5.3. Пользователи также могут нажимать и удерживать уведомление, чтобы изменить поведение связанного канала.

На устройствах под управлением Android 7.1 (уровень API 25) и ниже пользователи могут управлять уведомлениями только для каждого приложения. Каждое приложение фактически имеет только один канал на Android 7.1 и ниже.

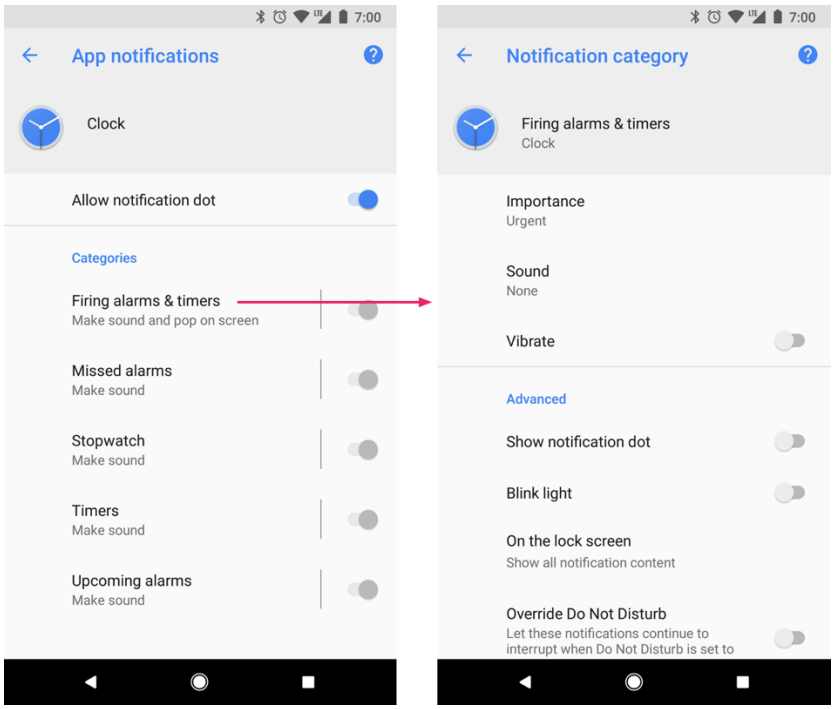

Рисунок 5.4. Настройки уведомлений для приложения Clock и одного из его каналов

Примечание: Пользовательский интерфейс ссылается на каналы как на "категории".

Приложение может иметь отдельные каналы для каждого типа уведомлений, которые выдает приложение. Приложение также может создавать каналы уведомлений в ответ на выбор, сделанный пользователями. Например, вы можете настроить отдельные каналы уведомлений для каждой группы разговоров, созданной пользователем в приложении для обмена сообщениями.

В канале также указывается уровень важности для ваших уведомлений на Android 8.0 и выше, поэтому все уведомления, отправляемые на один и тот же канал уведомлений, имеют одинаковое поведение.

6.1.5. Разрешения для уведомлений

В Android 13 для отображения уведомления требуется получение разрешения на выполнение. Такой тип разрешений может быть запрошен и для других методов, которые требуют обращения к некоторым системным компонентам Android.

Разрешения делятся на два типа (есть и другие, но они нас не интересуют):

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

Разрешение на показ уведомлений относится к опасным. Обычные разрешения будут получены приложением при установке, никакого подтверждения от пользователя не потребуется (немного спорный момент, на мой взгляд, стоило бы уведомлять пользователя об обязательных разрешениях). В дальнейшем отозвать их у приложения будет невозможно. Опасные же должны быть запрошены в процессе работы приложения и в любой момент могут быть отозваны.

Разрешения можно добавить с помощью adb в процессе отладки

```
adb shell pm grant <app package> <permission name>
adb shell pm revoke <app package> <permission name>
```

Но для обычного пользователя такой вариант не подходит. Для этого, были созданы методы checkSelfPermission, которые входят в ActivityCompat, FragmentCompat, ContextCompat. Каждый раз, когда необходимо использовать метод, требующий опасного разрешения, необходимо проверить есть ли оно у пользователя. Для этого используем метод ContextCompat.checkSelfPermission(Context context, String permission), который возвращает нам одно из int значений: PackageManager.PERMISSION_GRANTED в случае, если разрешение есть или

PackageManager.PERMISSION_DENIED если его нет. Именем разрешения является одна из констант класса Manifest.permission.

Далее, если разрешение есть, выполняем нужное нам действие, а если нет, то его нужно запросить. Одновременно можно запросить несколько разрешений (пользователю по очереди будет показан запрос на каждое из них), если это необходимо.

```
if (ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_EXTERNAL_STORAGE) ==
PackageManager.PERMISSION_GRANTED) {
    //Выполняем действие
}

public void requestPermissions() {
    ActivityCompat.requestPermissions(this,
        new String[] {
            Manifest.permission.READ_EXTERNAL_STORAGE
        },
        PERMISSION_REQUEST_CODE);
}
```

Для запроса используется метод ActivityCompat.requestPermissions(Activity activity, String[] permissions, int requestCode). Массив permissions соответственно содержит названия разрешений, которые необходимо запросить. Отсюда видно, что одновременно можно запрашивать несколько разрешений. requestCode — значение, по которому в дальнейшем можно будет определить, на какой запрос разрешения придет ответ, подобно тому как результат получается от activity, используя startActivityForResult.

Результат запроса разрешения следует обрабатывать в onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults). Параметры requestCode и permissions содержат данные, которые передавались при запросе разрешений. Основные данные здесь несет массив grantResults, в котором находится информация о том, получены разрешения или нет. Каждому i-му элементу permissions соответствует i-ый элемент из grantResults. Их возможные значения аналогичны результату checkSelfPermission.

```
@Override
public void onRequestPermissionsResult(int requestCode,
    @NonNull String[] permissions,
    @NonNull int[] grantResults) {
    if (requestCode == PERMISSION_REQUEST_CODE &&
grantResults.length == 1) {
        if (
            grantResults[0] == PackageManager.PERMISSION_GRANTED
        ) {

        }

    }
    super.onRequestPermissionsResult(
        requestCode, permissions, grantResults
    );
}
```

Размер массива grantResults проверяется для того, чтобы удостовериться, что запрос разрешения не был прерван (в этом случае permissions и grantResults не будут содержать элементов). Такую ситуацию следует рассматривать не как запрет разрешения, а как отмену запроса на него. Схема проверки наличия и запроса разрешений представлена на рис. 5.5.

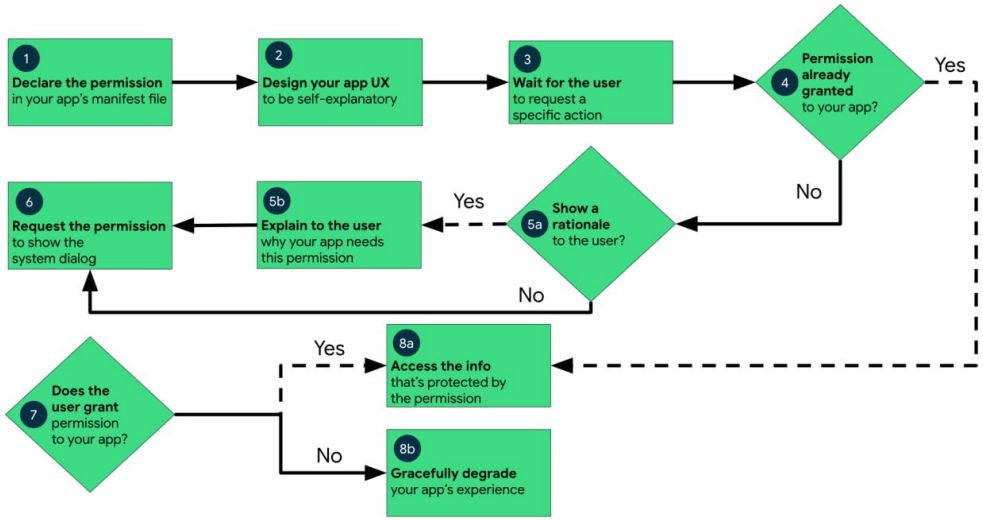

Рисунок 5.5. Схема проверки наличия и запроса разрешений

6.1.6. Создание и отображение уведомлений

Создание уведомление можно реализовать при помощи объекта класса NotificationCompat.Builder.

Примечание. Используемая переменная CHANNEL_ID должна быть задана глобально в классе `private final String CHANNEL_ID="channel_id";`.

```
private void showNotification() {
    NotificationCompat.Builder builder = new
        NotificationCompat.Builder(this, CHANNEL_ID)
            .setSmallIcon(R.drawable.ic_launcher_foreground)
            .setContentTitle(getString(
                R.string.notification_title))
            .setContentText(notificationText)
            .setPriority(NotificationCompat.PRIORITY_DEFAULT);
        NotificationManagerCompat notificationManager =
            NotificationManagerCompat.from(this);
        notificationManager.notify(
            notificationId, builder.build()
        );

    // notificationId - должен быть уникальным для каждого
уведомления в канале
}
```

6.2. Сервисы

A Service – это компонент приложения, который может выполнять длительные операции в фоновом режиме. Он не предоставляет пользовательский интерфейс. После запуска служба может продолжать работать в течение некоторого времени, даже после того, как пользователь переключится на другое приложение. Кроме того, компонент может привязываться к службе для взаимодействия с ней и даже выполнять межпроцессное взаимодействие (IPC). Например, служба может обрабатывать сетевые транзакции, воспроизводить музыку, выполнять ввод-вывод файлов или взаимодействовать с поставщиком контента – и все это в фоновом режиме.

6.2.1. Типы сервисов

Сервисы могут быть запущены как непосредственно в приложении в момент его отображения на экране, так и при его сворачивании и переходе в фоновый режим. Следовательно, существует 2 типа сервисов: фоновые и активные (запущенные).

В Android O (API 26) произошли существенные изменения в регулировании фоновых служб системой. Одно из главных изменений в том,

что запущенная служба, которая не в белом списке (в белый список помещаются службы, работа которых видна пользователю; подробнее смотри в офф. документации) или которая явно не сообщает пользователю о своей работе, не будет запускаться в фоновом потоке после закрытия Activity. Другими словами, вы должны создать уведомление (notification), к которому вы прикрепляете запущенную службу. И вы должны запускать службу с помощью нового метода startForegroundService() (а не с помощью startService()). И, после создания службы, у вас есть пять секунд чтобы вызвать метод startForeground() запущенной службы и показать видимое пользователю уведомление. Иначе система останавливает службу и показывает ANR ("приложение не отвечает").

6.2.2. Основы

Чтобы создать сервис, необходимо создать класс-наследник класса Service или использовать один из его существующих классов-наследников. В вашей реализации вы должны переопределить некоторые методы обратного вызова, которые обрабатывают ключевые аспекты жизненного цикла службы, и предоставить механизм, позволяющий компонентам привязываться к службе, если это необходимо. Это наиболее важные методы обратного вызова, которые вы должны переопределить:

onStartCommand() – система вызывает этот метод, вызывая startService() его, когда другой компонент (например, действие) запрашивает запуск службы. Когда выполняется этот метод, служба запускается и может работать в фоновом режиме бесконечно. Если вы реализуете это, вы несете ответственность за остановку службы по завершении ее работы, позвонив stopSelf()или stopService(). Если вы хотите только обеспечить привязку, вам не нужно реализовывать этот метод.

- onBind() – система вызывает этот метод, вызывая bindService(), когда другой компонент хочет связать со службой (например, для выполнения RPC). В вашей реализации этого метода вы должны предоставить интерфейс, который клиенты используют для связи со службой, возвращая IBinder. Вы всегда должны реализовывать этот метод; однако, если вы не хотите разрешать привязку, вы должны вернуть значение null.
- onCreate() – система вызывает этот метод для выполнения одноразовых процедур настройки при первоначальном создании службы (перед вызовом либо onStartCommand() или onBind()). Если служба уже запущена, этот метод не вызывается.

•onDestroy() – система вызывает этот метод, когда служба больше не используется и уничтожается. Ваша служба должна реализовать это, чтобы очистить любые ресурсы, такие как потоки, зарегистрированные слушатели или получатели. Это последний вызов, который получает служба.

Если компонент запускает службу путем вызова startService() (что приводит к вызову onStartCommand()), служба продолжает работать до тех пор, пока не остановится сама stopSelf()или другой компонент не остановит ее вызовом stopService().

Если компонент вызывает bindService() для создания службы и onStartCommand() *не* вызывается, служба выполняется только до тех пор, пока компонент привязан к ней. После того, как служба отключена от всех своих клиентов, система уничтожает ее.

Система Android останавливает службу только при нехватке памяти и должна восстанавливать системные ресурсы для выполнения действий, ориентированных на пользователя. Если служба привязана к действию, которое ориентировано на пользователя, вероятность ее отключения снижается; если служба объявлена для запуска на переднем плане, она редко отключается. Если служба запущена и работает долго, система со временем понижает свою позицию в списке фоновых задач, и служба становится очень уязвимой для уничтожения - если ваша служба запущена, вы должны спроектировать ее так, чтобы она корректно обрабатывала перезапуски системы. Если система отключает вашу службу, она перезапускает ее, как только ресурсы становятся доступными, но это также зависит от значения, из которого вы возвращаетесь onStartCommand().

6.2.3. Объявление службы в манифесте

Вы должны объявить все службы в файле манифеста вашего приложения так же, как вы делаете для activity и других компонентов.

Чтобы объявить свою службу, добавьте <service> элемент в качестве дочернего элемента <application> элемента. Вот пример:

```
<manifest ... >
    ...
    <application ... >
        <service
            android:name=".NameService"
            android:enabled="true"
            android:exported="true" />
        ...
```

```
    </application>
</manifest>
```

6.2.4. Запуск сервиса

Служба начинают свою работу после вызова метода startService(Intent) в вашей Activity или службе. При этом Intent должен быть явным. Это означает, что вы должны явно указать в Intent имя класса запускаемой вами службы. Служба получает это Intent в onStartCommand() методе, который должен включать в себя основную логику работы сервиса.

Когда служба запускается, ее жизненный цикл не зависит от компонента, который ее запустил. Служба может работать в фоновом режиме бесконечно, даже если компонент, который ее запустил, уничтожен. Таким образом, служба должна останавливать себя, когда ее работа завершается вызовом stopSelf(), или другой компонент может остановить ее вызовом stopService().

```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

Метод startService() возвращается немедленно, и система Android вызывает метод сервиса onStartCommand(). Если служба еще не запущена, система сначала вызывает onCreate(), а затем вызывает onStartCommand().

6.2.5. Остановка сервиса

Запущенная служба должна управлять своим собственным жизненным циклом. То есть система не останавливает и не уничтожает службу, если только она не должна восстановить системную память, и служба продолжает работать после onStartCommand() возврата. Служба должна остановить себя вызовом stopSelf(), или другой компонент может остановить ее вызовом stopService().

После запроса о прекращении работы с stopSelf() или stopService() система уничтожает службу как можно скорее.

6.3. Аппаратное наложение окна приложения поверх других приложений

Показ View поверх других приложений возможен с помощью указания флага WindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY. Флаг передается как параметр во WindowManager.LayoutParams, например: `mWindowManager.updateViewLayout(view, params)`. Для обновления LayoutParams используется менеджер окон и метод updateViewLayout.

```
final WindowManager.LayoutParams params = new
WindowManager.LayoutParams(
    WindowManager.LayoutParams.MATCH_PARENT,
    ViewGroup.LayoutParams.WRAP_CONTENT,
    WindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY,
    WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE,
    PixelFormat.TRANSLUCENT);

    mWindowManager.addView(view, params);
    params.gravity = Gravity.BOTTOM | Gravity.CENTER;

    mWindowManager.updateViewLayout(view, params);
```

Задание

1. Добавить в приложение кнопку, отображающую любое уведомление с текстом.
2. Создать сервис, который будет реализовывать аппаратное наложение и выводить баннер с информацией из приложения и возможностью перехода в запущенное приложение.