Notifications

Notifications sind Benachrichtigungen, die in der Statuszeile am mobilen Endgerät angezeigt werden. Die Einsatzmöglichkeiten für derartige Notifications sind vielfältig. Meist handelt es sich um Push-Nachrichten. In Verbindung mit GPS können so etwa positionsabhängige Nachrichten angezeigt werden.

Nutzer können diese Nachrichten anklicken, um zu lesen bzw. um direkt mit der Nachricht zu interagieren.

Versendet werden Notifications mithilfe eines `NotificationManager`-Objekts.

Eine Notification besteht aus: - einem Icon - einem Titel und - einem Text

Zusätzlich kann man auch noch ein Tonsignal bzw. eine Vibration versenden. Diese können vom User jedoch auch als störend empfunden werden und sollten nur bei absoluter Wichtigkeit eingesetzt werden.

Um auch die Notifications abwärts kompatibel zu gestalten, empfiehlt es sich, die entsprechende `NotificationCompat` Bibliothek zu verwenden. Diese muss im gradle File entsprechend eingebunden werden:

```
dependencies {
    ...
    implementation "com.android.support:support-compat:28.0.0"
    ...
}
```

Notification Builder

Um Notifications zu erzeugen, ist zunächst ein `NotificationBuilder` erforderlich:

```
NotificationCompat.Builder builder  = new NotificationCompat.Builder(
                                                                this, CHANNEL_ID)
                .setSmallIcon(android.R.drawable.star_big_on)
                .setColor(Color.YELLOW)
                .setContentTitle(getString(R.string.app_name))
                .setContentText("This is just a small Notification")
                .setWhen(System.currentTimeMillis())
                .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

Dem Konstruktor muss neben dem Kontext auch eine *Channel Id* mitübergeben werden. Dieser Parameter wird allerdings erst ab Android API-Level 26 verwendet. Frühere API-Versionen ignorieren diese Channel_ID.

Typischerweise passt der Inhalt der Notification in eine einzelne Zeile. Jedoch kann in einer Notification auch ein umfangreicherer Text angezeigt werden. Dafür kann die Notification mittels `setStyle()` entsprechend modifiziert werden:

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(
                                                    this, CHANNEL_ID)
                .setSmallIcon(android.R.drawable.star_big_on)
                .setContentTitle("My notification")
                .setContentText("Much longer text that cannot fit one line...")
                .setStyle(new NotificationCompat.BigTextStyle()
                                .bigText("Much longer text ......"))
                .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

Neben längerem Text können auch weitere Elemente, wie Bilder oder Bedieneinheit für Musik- und Videowiedergabe integriert werden. Siehe dazu: https://developer.android.com/training/notify-user/expanded.html

Anlegen eines Notification Channels

Da der Notification Channel angelegt werden muss, bevor Notifications gepostet werden können, sollte dies gleich beim Starten der App erfolgen. Das wiederholte Anlegen eines Channels ist problemlos möglich, da dieser einfach überschrieben wird.

Die Priorität der Notification gibt an, wie der Benutzer durch die Notification aufmerksam gemacht wird. Da Channels für Notifications erst ab Android 8 unterstützt werden, muss zusätzlich mit `setPriority` die Priorität für Android 7.1 und darunter gesetzt werden.

Die verschiedenen Prioritätslevels sind hier angeführt: https://developer.android.com/training/notify-user/channels.html#importance

```
// Create the NotificationChannel, but only on API 26+ because
// the NotificationChannel class is new and not in the support library
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    CharSequence name = getString(R.string.channel_name);
    String description = getString(R.string.channel_description);
    int importance = NotificationManager.IMPORTANCE_DEFAULT;
    NotificationChannel channel = new NotificationChannel(
                                        CHANNEL_ID, name, importance);
    channel.setDescription(description);
    // Register the channel with the system; you can't change the importance
    // or other notification behaviors after this
    NotificationManager notificationManager = getSystemService(
                                                    NotificationManager.class);
    notificationManager.createNotificationChannel(channel);
}
```

Erzeugen der Tap-Action für die Notifications

Für jede Notification kann das entsprechende Tap-Verhalten definiert werden. Typischerweise wird eine neue Activity geöffnet, die mit der Notification in

Zusammenhang steht.

Angenommen, die Notification gibt Auskunft über neue Mails, dann könnte mittels Tap auf die Notification eine entsprechende Activity geöffnet werden, die die neuen Nachrichten anzeigt.

Zuerst muss ein entsprechender `PendingIntent` angelegt werden.

```
// Create an explicit intent for an Activity in your app
Intent intent = new Intent(this, NotificationDetails.class);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);
```

Dieser wird nun dem NotificationBuilder hinzugefügt:

```
NotificationCompat.Builder builder  = new NotificationCompat.Builder(
                                                              this, CHANNEL_ID)
                .setSmallIcon(android.R.drawable.star_big_on)
                .setColor(Color.YELLOW)
                .setContentTitle(getString(R.string.app_name))
                .setContentText("This is just a small Notification")
                .setWhen(System.currentTimeMillis())
                .setPriority(NotificationCompat.PRIORITY_DEFAULT)
                // Set the intent that will fire when the user taps the notification
                .setContentIntent(pendingIntent)
                .setAutoCancel(true);
```

Obiges CodeSnippet ruft die Methode `setAutoCancel()` auf. Dies führt dazu, dass die Notification automatisch gelöscht wird, nachdem sie vom User angeklickt wurde.

Der Aufruf von `setFlags()` sorgt dafür, dass die vom Benutzer erwartete Navigation entsprechend beibehalten wird, nachdem die App durch die Notification geöffnet wurde. Ob dieses Verhalten gewünscht wird, hängt von folgenden Faktoren ab:

- **Die Activity existiert nur als Antwort auf die Notification.** Es macht also keinen Sinn, dass der User aus einem anderen Grund auf diese Activity navigiert. In diesem Fall wird die Activity als eigener Task gestartet, anstatt zum aktuellen Task und BackStack der geöffneten App hinzugefügt zu werden.
- **Die Activity existiert auch im regulären App-Flow.** In diesem Fall sollte der Aufruf der Activity einen BackStack anlegen, damit die vom User erwartete Navigation mittels BackButton und UpButton funktioniert.

Anzeigen der Notification

Durch einen Aufruf von `NotificationManagerCompat.notify()` wird die Notification angezeigt. Diesem Aufruf muss eine eindeutige ID mitgegeben werden, die vom Entwickler festgelegt wird.

```
NotificationManagerCompat notificationManager =
                                        NotificationManagerCompat.from(this);
// notificationId is a unique int for each notification that you must define
int notificationId = 1;
notificationManager.notify(notificationId, builder.build());
```

Die ID, die dem Aufruf von `notify` mitübergeben wird, wird später benötigt, falls die Notification geändert oder gelöscht werden sollte!

Action Buttons zur Notification hinzufügen

Maximal 2 Buttons können einer Notification direkt hinzugefügt werden. Mithilfe dieser Buttons kann der User schnell auf die Notification reagieren bzw. antworten.

Im Sinne einer guten User Experience sollten diese Buttons jedoch nicht das Verhalten des Tap duplizieren, sondern anstatt bzw. mit ergänzendem Verhalten verwendet werden.

Um einen Action Button zur Notification hinzuzufügen, verwendet man wieder einen `PendingIntent`. Dieser wird der `addAction` Methode hinzugefügt. Die Herangehensweise ist ähnlich, wie beim Tap-Verhalten. Es wird jedoch keine neue Activity gestartet. Es steht aber ein breites Spektrum an Möglichkeiten zur Verfügung, wie etwa das Starten eines BroadcastReceivers, der im Hintergrund arbeitet.

```
Intent snoozeIntent = new Intent(this, MyBroadcastReceiver.class);
snoozeIntent.setAction(ACTION_SNOOZE);
snoozeIntent.putExtra(EXTRA_NOTIFICATION_ID, 0);
PendingIntent snoozePendingIntent =
       PendingIntent.getBroadcast(this, 0, snoozeIntent, 0);

NotificationCompat.Builder builder = new NotificationCompat.Builder(
       this, CHANNEL_ID)
       .setSmallIcon(android.R.drawable.alert_dark_frame)
       .setContentTitle("My notification with snooze button")
       .setContentText("This notification can be snoozed!")
       .setPriority(NotificationCompat.PRIORITY_DEFAULT)
       .setContentIntent(pendingIntent)
       .addAction(android.R.drawable.ic_menu_zoom
               , getString(R.string.snooze),
               snoozePendingIntent);
```

Hier finden sich Informationen, wie Steuerelemente für die Medienwiedergabe direkt in die Notification eingebunden werden können: https://developer.android.com/training/notify-user/expanded.html#media-style

Hinzufügen eines Direkt-Antwort-Buttons

Seit Android 7.0 (API Level 24) können User auch direkt auf Notifications antworten, ohne dass dafür eine eigene Activity geöffnet werden müsste.

Die Funktionalität der Direktantwort ist als zusätzlicher Button in der Notification implementiert. Über diesen wird eine Texteingabemöglichkeit geöffnet über welche der User seine Antwort eingibt. Die Eingabe vom User wird dem Intent, der mit der Notification verbunden wurde, beigefügt und an die App gesendet.

Pending Intent

Wie bei anderen Notifications auch, benötigt eine Notification mit direkter Antwortmöglichkeit natürlich auch ein Objekt der Klasse `PendingIntent`:

```
PendingIntent helpPendingIntent = PendingIntent.getBroadcast(
                MainActivity.this,
                REQUEST_CODE_HELP,
                new Intent(MainActivity.this, MyBroadcastReceiver.class)
                        .putExtra(KEY_INTENT_HELP, REQUEST_CODE_HELP),
                PendingIntent.FLAG_UPDATE_CURRENT
        );
```

Dieser PendingIntent enthält eine Referenz auf einen MyBroadcastReceiver, der mit der Abarbeitung der Konversation betraut wird (in diesem Fall die Klasse `MyBroadcastReceiver`, die wir später noch betrachten werden).

RemoteInputer Builder

Für die Texteingabe ist eine Instance der Klasse `RemoteInput.Builder` erforderlich, die der Notification hinzugefügt werden kann. Der Konstruktor verlangt einen `String`, welcher als Id für den eingegebenen Text dient. Um den eingegebenen Text zu erhalten, ist diese Id erforderlich.

```
RemoteInput remoteInput = new RemoteInput.Builder(NOTIFICATION_REPLY)
                .setLabel("Please enter your question")
                .build();
```

Action Objekt

Das ActionObjekt ist für das Handling der Texteingabe erforderlich und erhält deshalb eine Referenz auf das `remoteInput` Objekt sowie den PendingIntent.

```
NotificationCompat.Action action =
                new NotificationCompat.Action.Builder(
                        android.R.drawable.ic_delete,
                        "Reply Now...", helpPendingIntent)
                        .addRemoteInput(remoteInput)
                        .build();
```

Verschicken der Notification

Zuletzt muss die Notification noch erstellt werden und wieder mithilfe vom NotificationManager versendet werden.

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(
                                                                this, CHANNEL_ID)
                .setSmallIcon(android.R.drawable.ic_dialog_email)
                .setContentTitle("Hey, we received your email!
                                                How can we help you?")
                .setContentText("How can we help you?")
                .setAutoCancel(true)
                .setContentIntent(helpPendingIntent)
                .addAction(action)
                .addAction(android.R.drawable.ic_menu_directions,
                            "Help", helpPendingIntent);

NotificationManager notificationManager = (NotificationManager)
                                    getSystemService(NOTIFICATION_SERVICE);
notificationManager.notify(NOTIFICATION_ID, mBuilder.build());
```

MyBroadcastReceiver

Die Klasse `MyBroadcastReceiver` ist für die Verarbeitung der Texteingaben in Bezug auf die Notification im Hintergrund zuständig. BroadcastReceiver werden im weiteren noch gesondert besprochen. In diesem Zusammenhang soll nur die Klasse `MyBroadcastReceiver` besprochen werden.

Zuerst muss die Klasse von `BroadcastReceiver` abgeleitet werden und muss damit die Methode `onReiceive` überschreiben. Diese wird aufgerufen, wernn der Broadcast-Receiver Daten empfängt. In unserem Fall wird die Texteingabe des Benutzers verarbeitet.

```
public void onReceive(Context context, Intent intent) {
    //getting the remote input bundle from intent
    Bundle remoteInput = RemoteInput.getResultsFromIntent(intent);

    //if there is some input
    if (remoteInput != null) {

        //getting the input value
        CharSequence name = remoteInput.getCharSequence(
                                        MainActivity.NOTIFICATION_REPLY);

        //updating the notification with the input value
        NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(
                                                context,
                                                MainActivity.CHANNEL_ID)
```

```
                .setSmallIcon(android.R.drawable.ic_menu_info_details)
                .setContentTitle("Hey Thanks, " + name);
        NotificationManager notificationManager = (NotificationManager)
                context.getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(MainActivity.NOTIFICATION_ID,
                                   mBuilder.build());
    }

    //if help button is clicked
    if (intent.getIntExtra(MainActivity.KEY_INTENT_HELP, -1) ==
                                            MainActivity.REQUEST_CODE_HELP) {
        Toast.makeText(context, "You Clicked Help", Toast.LENGTH_LONG).show();
    }
}
```

Damit der Broadcast Receiver verwendet werden kann, muss dieser - ähnlich wie Activites - im Manifest eingetragen werden:

```
<receiver
        android:name=".MyBroadcastReceiver"
        android:enabled="true"
        android:exported="false"></receiver>
```

Notification mit ProgressBar

Sollte eine Notification den aktuellen Fortschritt einer Operation im Hintergrund darstellen, kann ein ProgressBar für den User wertvolle Informationen liefern.

Wird z.B. im Hintergrund eine größere Datei heruntergeladen, so kann mithilfe vom ProgressBar dem User der aktuelle Fortschritt angezeigt werden.

Im Codebeispiel wird eine derartige längere Hintergrundoperation mittels `Thread.sleep` simuliert:

```
final NotificationManager mNotifyManager = (NotificationManager)
                            getSystemService(Context.NOTIFICATION_SERVICE);
final NotificationCompat.Builder mBuilder =
                            new NotificationCompat.Builder(this, CHANNEL_ID);
mBuilder.setContentTitle("Picture Download")
        .setContentText("Download in progress")
        .setSmallIcon(android.R.drawable.ic_dialog_info);

// Start a lengthy operation in a background thread
new Thread(new Runnable() {
    @Override
    public void run() {
        int incr;
        // Do the "lengthy" operation 20 times
```

```
        for (incr = 0; incr <= 100; incr += 10) {
            // Sets the progress indicator to a max value, the
            // current completion percentage, and "determinate"
            // state
            mBuilder.setProgress(100, incr, false);
            // Displays the progress bar for the first time.
            mNotifyManager.notify(0, mBuilder.build());
            // Sleeps the thread, simulating an operation
            // that takes time
            try {
                // Sleep for 5 seconds
                Thread.sleep(1 * 1000);
            } catch (InterruptedException e) {
                Log.d(TAG, "sleep failure");
            }
        }
        // When the loop is finished, updates the notification
        mBuilder.setContentText("Download complete")
                // Removes the progress bar
                .setProgress(0, 0, false);
        mNotifyManager.notify(999, mBuilder.build());
        }
    // Starts the thread by calling the run() method in its Runnable
}).start();
```

Mehrere Notifications in Gruppen zusammenfassen

Ab Android 7.0 (API-Level 24) ist es möglich, Notifications in einer Queue zu bündeln: `bundled notifications`. Wenn die App zB. Notifications für erhaltene Nachrichten darstellt und mehr als eine Nachricht erhalten wurden, können diese Nachrichten in einer einzigen Notification-Gruppe gebündelt werden.

Eine solche Gruppe von Notifications ist hierarchisch geordnet. Ganz oben steht die Parent-Notification, die eine Zusammenfassung für die gesamte Gruppe anzeigt. Der Benutzer kann die Notification Gruppe nun erweitern. Je weiter der User die Gruppe öffnet, umso mehr Information wird angezeigt.

Gebündelte Notifications sollten immer dann verwendet werden, wenn folgende Bedingungen zutreffen:

- Die Child-Notifications stellen komplette Notifications dar und könnten auch einzeln dargestellt werden.
- Es gibt einen Vorteil, die Child-Notifications anzuzeigen. (z.B. Mit jedem Kind-Element sind individuelle Actions verknüpft.) Bsp.: E-Mail App, Messaging App, etc.

Um Notifications zu gruppieren, gehe wie folgt vor:

Um einen Notification Stack zu erstellen, rufe die Methode `setGroup()` für jede einzelne Notification auf und definieren eine ID für die Gruppe. Danach rufe die Methode `notify()` auf.

```
// Build the notification, setting the group appropriately
Notification notification = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setContentTitle("New mail from Max!")
    .setContentText("Hi out there! This is my mail! Cheers! Max")
    .setSmallIcon(android.R.drawable.ic_dialog_info)
    .setGroup(GROUP_KEY_EMAILS)
    .build();
// Issue the notification
NotificationManagerCompat notificationManager =
                        NotificationManagerCompat.from(this);
notificationManager.notify(111, notification);
```

Wenn du später eine weitere Notification erstellst, definiere wieder eine Notification-Group mit der gleichen ID. Rufe `notify()` auf, um die Notification anzuzeigen.

```
// Sending another notification to the same group
Notification notification2 = new NotificationCompat.Builder(this, CHANNEL_ID)
     .setContentTitle("New mail from Erik!")
     .setContentText("Hi there! This is Erik!")
     .setSmallIcon(android.R.drawable.ic_dialog_info)
     .setGroup(GROUP_KEY_EMAILS)
     .build();
notificationManager.notify(222, notification2);
```

Füge eine Summary Notification ein, die eine Zusammenfassung für die Notificationsgroup anzeigt. Rufe dafür die Methode `setGroupSummary()` auf.

```
Bitmap largeIcon = BitmapFactory.decodeResource(getResources(),
                android.R.drawable.ic_dialog_info);

// Create an InboxStyle notification
Notification summaryNotification = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setContentTitle("2 new messages")
    .setSmallIcon(android.R.drawable.ic_dialog_info)
    .setLargeIcon(largeIcon)
    .setStyle(new NotificationCompat.InboxStyle()
    .addLine("Alex Faaborg   Check this out")
    .addLine("Jeff Chang   Launch Party")
    .setBigContentTitle("2 new messages")
    .setSummaryText("johndoe@gmail.com"))
    .setGroup(GROUP_KEY_EMAILS)
    .setGroupSummary(true)
    .build();
```

```
notificationManager.notify(333, summaryNotification);
```