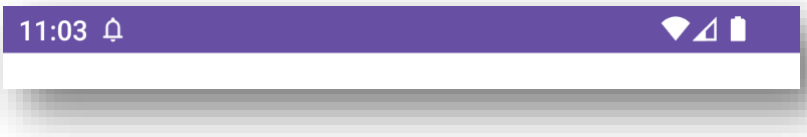
Уведомления

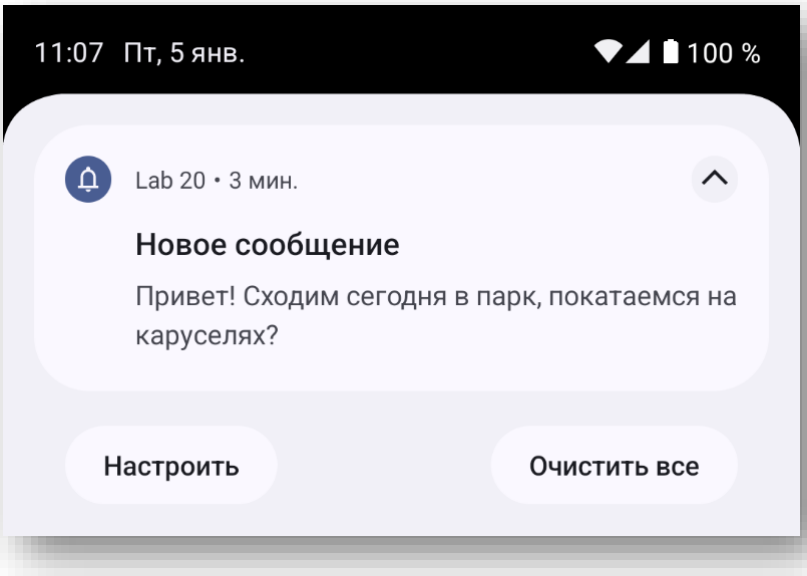
Уведомление — это сообщение, которое Android отображает за пределами пользовательского интерфейса приложения. Оно служит для предоставления пользователю напоминания, сообщения от других людей, или другой своевременной информации из приложения. Пользователи могут коснуться уведомления, чтобы открыть приложение, или выполнить действие непосредственно из уведомления.

Уведомления могут отображаться в разных местах:

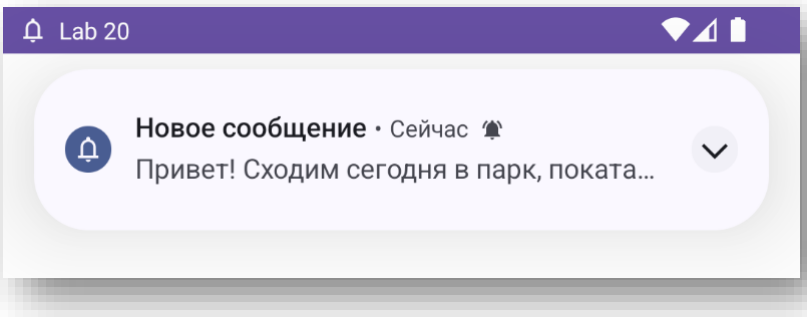
- в виде значка в строке состояния:



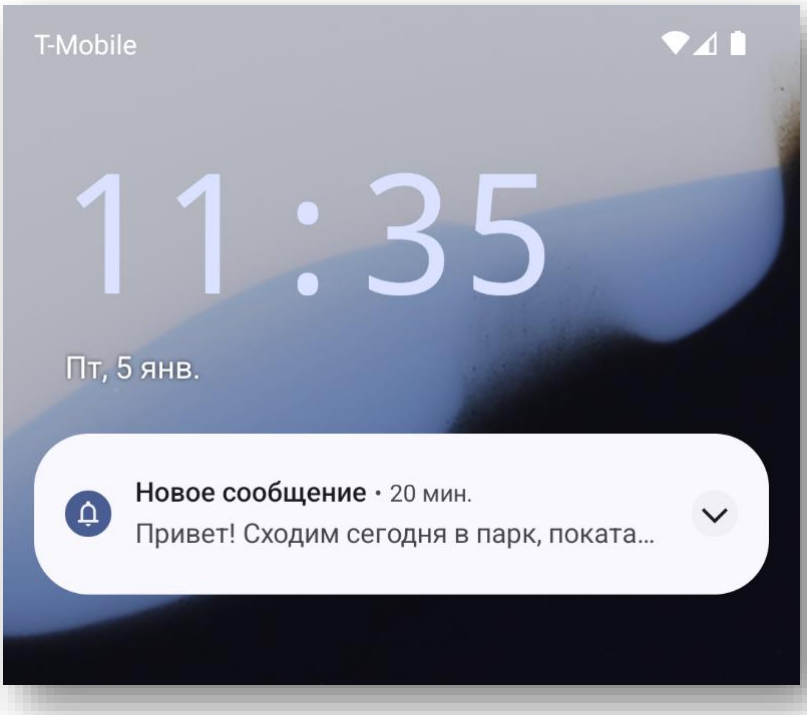
- в виде текста и значка приложения в планке на панели уведомлений:



- во всплывающем окне – так ненадолго появляются важные уведомления, о которых пользователь должен узнать немедленно, после этого они остаются в панели уведомлений:



- на экране блокировки, если пользователь разрешил это в настройках безопасности:



- на значке приложения – так называемся *точка уведомления*:

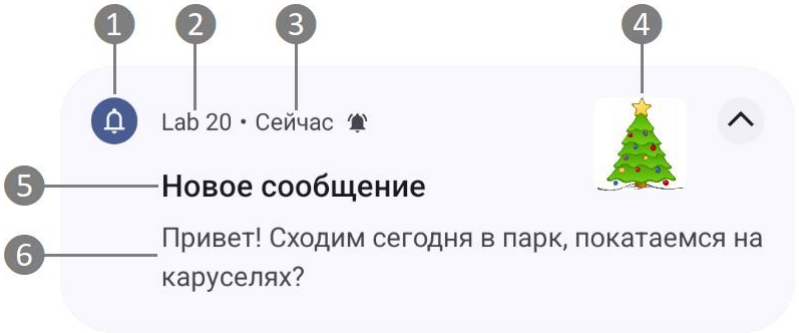
- на сопряженных носимых устройствах.

Планку на панели уведомлений можно потянуть вниз и тогда она отобразится в расширенном формате – с большим количеством информации и кнопками действия.

Уведомление остается видимым на панели уведомлений до тех пор, пока оно не будет закрыто приложением или пользователем.

Структура уведомления

Дизайн уведомления определяется системными шаблонами, приложение лишь определяет содержимое каждой части шаблона.



Самые используемые части уведомления показаны на рисунке выше:

1. **Маленький значок**: обязательно, устанавливается методом `setSmallIcon()`.
2. **Имя приложения**: предоставляется системой.
3. **Метка времени**: предоставляется системой, но можно переопределить ее методом `setWhen()` или скрыть методом `setShowWhen(false)`.
4. **Большой значок**: необязательно, устанавливается методом `setLargeIcon()`. Используется только для фотографий контактов, не следует использовать для значка приложения!
5. **Заголовок**: необязательно, устанавливается методом `setContentTitle()`.
6. **Текст**: необязательно, устанавливается методом `setContentText()`.

Настоятельно рекомендуется использовать системные шаблоны, чтобы обеспечить единый стиль дизайна на всех типах устройствах. Однако, при необходимости можно создать собственную разметку для уведомлений.

Менеджер уведомлений

Для работы с уведомлениями используется специальный системный сервис – менеджер уведомлений. Получить к нему доступ можно так:

```
val notificationManager = getSystemService(NOTIFICATION_SERVICE) as NotificationManager
```

Однако Google рекомендует вместо этого использовать класс `NotificationManagerCompat` из библиотеки Jetpack – это позволит сохранять совместимость с предыдущими версиями Android, в том числе использовать в старых версиях Android некоторые возможности, которые появились лишь в более новых версиях:

```
val notificationManager = NotificationManagerCompat.from(this)
```

Везде в примерах этой темы объект `notificationManager` получен примерно таким образом.

Разрешение

Начиная с Android 13 (уровень API 33) для отображения уведомлений приложение должно запросить разрешение `POST_NOTIFICATIONS` у пользователя. Если разрешение не запрошено, то показ уведомлений не разрешен, и уведомления просто не будут появляться. Предыдущие версии Android не требуют запроса разрешения.

Разрешение необходимо как обычно заявить в манифесте:

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

Запросить разрешение можно обычным образом, однако поскольку такое разрешение появилось только в API 33, то следует выполнять его только для новых API:

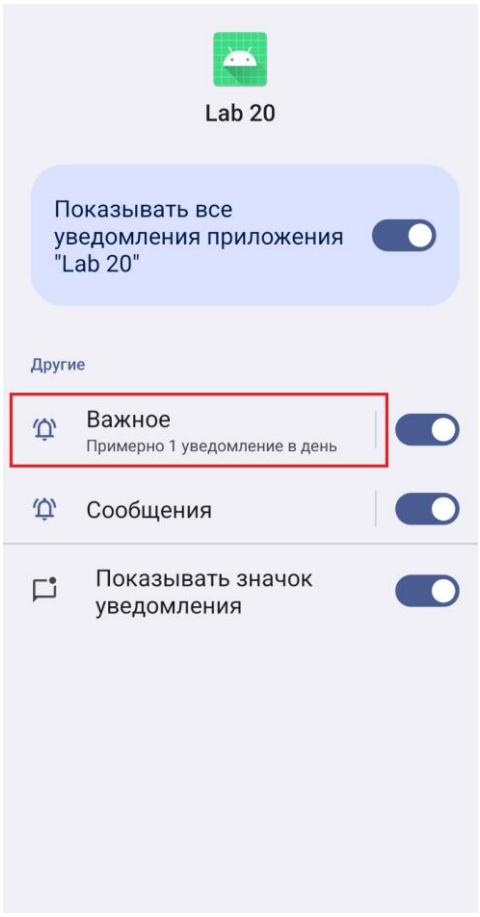
```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) { // Android 13 и выше
    if (ActivityCompat.checkSelfPermission(this,
        Manifest.permission.POST_NOTIFICATIONS) != PackageManager.PERMISSION_GRANTED) {
        requestResult.launch(android.Manifest.permission.POST_NOTIFICATIONS)
    }
}
else
    showNotification() // Более старые версии Android
```

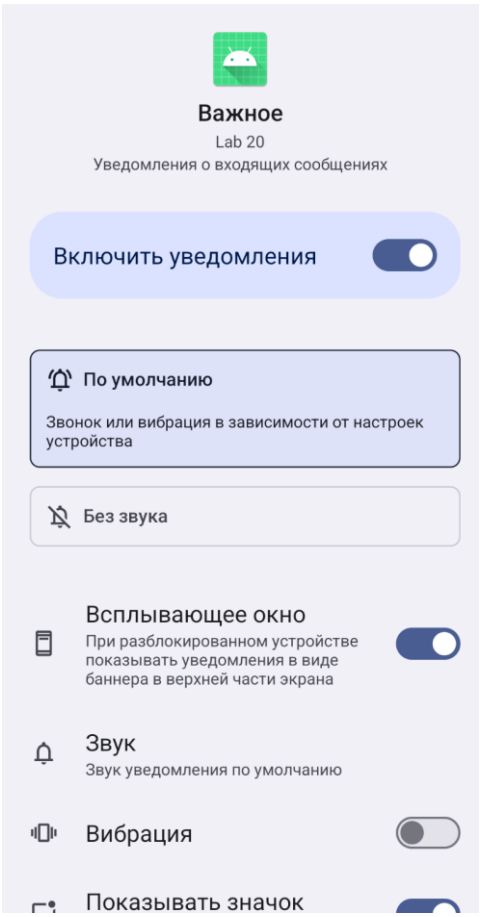
Здесь `requestResult` – это объект, который получит результаты запроса разрешений:

```
private val requestResult =
registerForActivityResult(ActivityResultContracts.RequestPermission()) {
    granted ->
        if (granted)
            showNotification()
}
```

Каналы

Все уведомления необходимо размещать в каналах. Для каждого канала можно задать название, описание, важность уведомлений, а также настроить визуальное и звуковое поведение, которое будет применяться ко всем уведомлениям в этом канале. Пользователь может поменять часть настроек, а также скрыть отдельные каналы, если уведомления в них ему не интересны или слишком навязчивы.





После создания канала программа не сможет изменить поведение уведомлений, над этим полный контроль имеет пользователь. Можно изменить только название и описание канала.

Чтобы создать канал, нужно инициализировать объект `NotificationChannel`, указав ему уникальный идентификатор, название и уровень важности. По желанию можно указать и описание, которое будет выводиться в настройках канала. Затем передать этот объект в функцию `createNotificationChannel()`. Если канал уже существует, то никакие действия выполнены не будут, поэтому можно создавать канал каждый раз при запуске приложения

или перед выводом уведомления – реальное создание канала произойдёт лишь в первый раз.

```
val CHANNEL_MSG_ID = "lab20_channel_msg"
val channel = NotificationChannel(
                  CHANNEL_MSG_ID,
                  "Сообщения", // Название канала
                  NotificationManager.IMPORTANCE_DEFAULT)
channel.description = "Уведомление о входящих сообщениях" // Описание канала
notificationManager.createNotificationChannel(channel)
```

Уровень важности определяет поведение уведомлений, которые будут появляться в этом канале:

- `IMPORTANCE_HIGH`, «срочный» – звуковой сигнал, всплывающее сообщение
- `IMPORTANCE_DEFAULT`, «высокий» – звуковой сигнал
- `IMPORTANCE_LOW`, «средний» – без звука
- `IMPORTANCE_MIN`, «низкий» – без звука, не отображается в строке состояния
- `IMPORTANCE_NONE`, «нет» – без звука, не отображается в строке состояния или шторке.

При желании можно задать и ряд других параметров, например, цвет индикатора уведомления (свойство `lightColor`), тип вибрации (метод `setVibrationPattern()`) и т.д.

С помощью метода `getNotificationChannel()` можно запросить у системы объект `NotificationChannel` с теми изменениями, которые пользователь мог внести в поведение канала (например, запретив вибрацию или вообще скрыв уведомления).

Если канал уведомлений больше не нужен (например, в мессенджере удалена группа контактов, для которой пользователь установил особые настройки уведомлений), то канал можно удалить с помощью метода `deleteNotificationChannel()`.

Уведомление

Уведомление представляется в виде объекта класса `Notification`, однако напрямую его создавать не рекомендуется, вместо этого используется построитель (англ. Builder) из класса `NotificationCompat`:

```
val builder = NotificationCompat.Builder(this, CHANNEL_MSG_ID)
    .setSmallIcon(R.drawable.notifications)
    .setContentTitle("Новое сообщение")
    .setContentText("Привет! Сходим сегодня в парк, покатаемся на каруселях?")
val notification = builder.build()
```

В процессе постройки используются методы для установки значка, заголовка и текстового содержимого, а в качестве параметра построителя указывается идентификатор канала, в котором будет размещено уведомление.

После того как объект уведомления создан, его можно показать пользователю:

```
notificationManager.notify(id, notification)
```

В качестве первого параметра передаётся уникальный числовой идентификатор, с его помощью можно будет впоследствии управлять уведомлением.

Например, чтобы обновить уведомление, нужно снова вызвать метод `notify()`, передав ему тот же идентификатор, который использовался ранее. Если предыдущее уведомление отклонено пользователем, вместо него создается новое уведомление. Можно использовать метод `setOnlyAlertOnce()`, чтобы звук, вибрация или всплывающее окно появлялось только при первом показе уведомления, но не для последующих обновлений.

Если уведомление больше неактуально, можно удалить его вызвав метод `cancel()` с идентификатором. Метод `cancelAll()`удалит все отправленные ранее уведомления. Наконец, можно установить таймаут при создании уведомления с помощью метода `setTimeoutAfter()`, тогда оно исчезнет автоматически после истечения срока жизни.

Касание

Уведомление должно реагировать на касание – как правило, при этом открывается приложение и выполняется подходящее действие. Для этого должно быть создано *отложенное намерение*, его следует установить с помощью метода `setContentIntent()`.

Отложенное намерение – это, фактически, обёртка вокруг обычного намерения, которая позволяет передать это намерение системе или другому приложению, а в нужный момент выполнить указанную в намерении операцию так, словно это приложение является Вашим приложением – с теми же разрешениями и правами. Отложенное намерение создаётся с помощью класса `PendingIntent` и поддерживается системой даже если процесс приложения-владельца был уничтожен.

Добавляется отложенное намерение следующим образом:

```
val intent = Intent(this, AlertActivity::class.java).apply {
    flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
}
val pendingIntent = PendingIntent.getActivity(this, 0, intent,
                                              PendingIntent.FLAG_IMMUTABLE)


builder
    ...
    .setContentIntent(pendingIntent)
    .setAutoCancel(true)
```

В этом примере создаётся обычное намерение, которое будет запускать активность `AlertActivity`, причем с помощью флагов указывается что активность должна выполниться не в контексте запущенного приложения, а как новая задача со своим стеком возврата. Затем намерение оборачивается в отложенное намерение, и уже оно устанавливается в сборщик уведомления. Также с помощью метода `setAutoCancel()` указывается что после касания уведомления и запуска приложение, уведомление должно быть убрано, в противном случае уведомление так и останется активным.

Кнопки действий

Помимо касания можно добавить и действия, которые вызываются нажатием кнопок в уведомлении. Сначала нужно создать намерение, которое сработает при нажатии кнопки:

```
val myButtonActionIntent = Intent(this, MyReceiver::class.java).apply {
    action = "ru.testapp.some_action"
```

```
    putExtra(EXTRA_NOTIFICATION_ID, 1)
}
```

После этого нужно обернуть намерение в отложенное намерение:

```
val myActionPendingIntent = PendingIntent.getBroadcast(
    this,
    0,
    myButtonActionIntent,
    PendingIntent.FLAG_IMMUTABLE)
```

И затем добавить действие к уведомлению, указав значок, текст на кнопке, и отложенное намерение, которое будет вызвано при нажатии на кнопку:

```
.addAction(R.drawable.myIconr, "Название", myActionPendingIntent)
```

Если при касании уведомления всегда запускается какая-либо активность, в случае кнопок действия можно выполнять и другие операции, например, отправлять широковещательную рассылку как примере выше. Нужно указать класс приемника широковещательных рассылок (`MyReceiver` в примере) и/или строку действие, которая будет передана.

Если указан конкретный класс приемника, при нажатии кнопки именно он будет создан системой, и его методу `onReceive` будет передана рассылка. Взаимодействие с активностью в этом случае очень ограничено, поскольку если даже приложение запущено, ссылок на открытую активность во вновь созданном приемнике нет. А ведь приложение может быть и не запущено, в этом случае никакого взаимодействия не получится – сначала нужно запустить приложение, и лишь затем как-то с ним взаимодействовать.

Если указана строка действия (action), то обработать его можно, например, в приемнике, который создан внутри активности – в этом случае доступны все методы и свойства активности:

```
private val broadcastReceiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        when (intent?.action) {
            ACTION_1 -> ...
            ACTION_2 -> ...
        }
    }
}
```

Конечно, нужно не забыть зарегистрировать приемник при запуске приложения, указав строки действия, которые нужно принимать, и выставив флаг `RECEIVER_EXPORTED` чтобы приемник мог принимать рассылки извне приложения:

```
ContextCompat.registerReceiver(this, broadcastReceiver, IntentFilter().apply {
    addAction("com.myapp.action1")
    addAction("com.myapp.action2")
}, ContextCompat.RECEIVER_EXPORTED)
```

Минусом такого подхода является то, что работает динамический приемник только если приложение запущено, а если оно не запущено, то даже запустить его будет некому, и рассылка пропадёт без результата.

Более правильным является вариант, при котором приемник широковещательных рассылок находится в фоновом сервисе, и при получении рассылки сам сервис выполняет требуемое действие, например, удаляет письмо, а активность лишь отображает изменения. Работа сервисов разбирается в отдельной теме.

Поле ввода ответа

Иногда бывает удобно не просто выполнить предопределенное действие по нажатию кнопки, а ввести и отправить текст прямо из уведомления – например, ответ на сообщение в мессенджере. Первым делом нужно создать элемент класса `RemoteInput` – именно он отвечает за текстовое поле в уведомлении:

```
val remoteInput = RemoteInput.Builder(KEY_TEXT).run {
    setLabel("Введите текст... ")
    build()
}
```

Здесь `KEY_TEXT` – это строка-ключ, по которой впоследствии можно будет получить введённый текст. Метод `setLabel()` позволяет установить текст-подсказку, которая находится внутри текстового поля до начала ввода текста.

Далее нужно создать намерение, обернуть его в отложенное намерение – примерно так же как выше при создании кнопки-действия. Однако, для намерения, которое относится к полю ввода, нужно дополнительно задать имя пакета приложения – это требуется в целях безопасности:

```
val setColorIntent = Intent("ru.testapp.my_input_action")
setColorIntent.setPackage(packageName)
```

Отложенное намерение при этом должно содержать флаг, определяющий будет ли оно изменяемым (mutable) или нет (immutable). Проблема в том, что в разных версиях Android требования к изменяемости разные, поэтому придётся добавить проверку версии:

```
val flags = when {
    Build.VERSION.SDK_INT >= Build.VERSION_CODES.S ->
        PendingIntent.FLAG_MUTABLE
    Build.VERSION.SDK_INT >= Build.VERSION_CODES.M ->
        PendingIntent.FLAG_IMMUTABLE
    else ->
        PendingIntent.FLAG_UPDATE_CURRENT
}

val replyPendingIntent: PendingIntent =
    PendingIntent.getBroadcast(applicationContext,
        1,
        setColorIntent,
        flags)
```

На последнем этапе нужно создать и само действие, только вместо отдельных элементов (значок, название и намерение) нужно создать его сразу целиком в виде объекта класса `Action`:

```
val myTextAction = Action.Builder(R.drawable.myIcon, "Название", replyPendingIntent)
```

```
                .addRemoteInput(remoteInput)
                .build()
```

А затем добавить этот объект при построении уведомления:

```
.addAction(myTextAction)
```

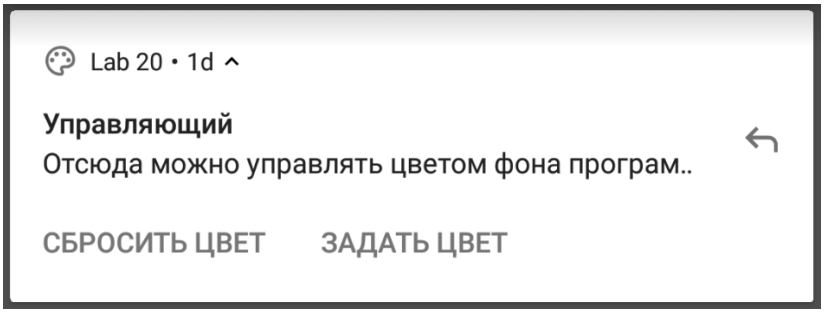
После того как пользователь ввёл текст и нажал кнопку отправки, будет выполнено отложенное намерение и приложение получит в приемнике широковещательных рассылок объект `intent`, из которого можно извлечь введённый текст:

```
val text =
RemoteInput.getResultsFromIntent(intent)?.getCharSequence(KEY_TEXT).toString()
```

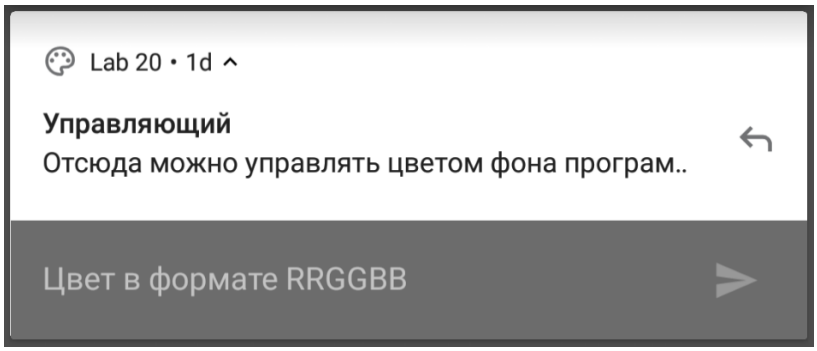
Нужно не забыть обязательно обновить уведомление после обработки текста, например, добавить отправленный текст в уведомление!

Задание

Напишите программу, которая по нажатию кнопки отображает уведомление – конечно, с обязательной проверкой разрешена ли отправка уведомлений, запросом такого разрешения и т.п. формальностями. Уведомление должно выглядеть примерно следующим образом:



При нажатии уведомления должна запускаться программа (даже если она уже запущена, что поделать…). При нажатии кнопки «Сбросить цвет» – фон активности становится белым. При нажатии кнопки «Задать цвет» – появляется поле ввода, в котором можно ввести цвет в формате RRGGBB:



После отправки цвета – фон активности становится выбранного цвета, а в центре появляется надпись вида «Выбран цвет: #RRGGBB», где RRGGBB – это hex-код выбранного цвета:


Выбран цвет: #FFBD1B
Показать уведомление