ТЕХНОТРЕК

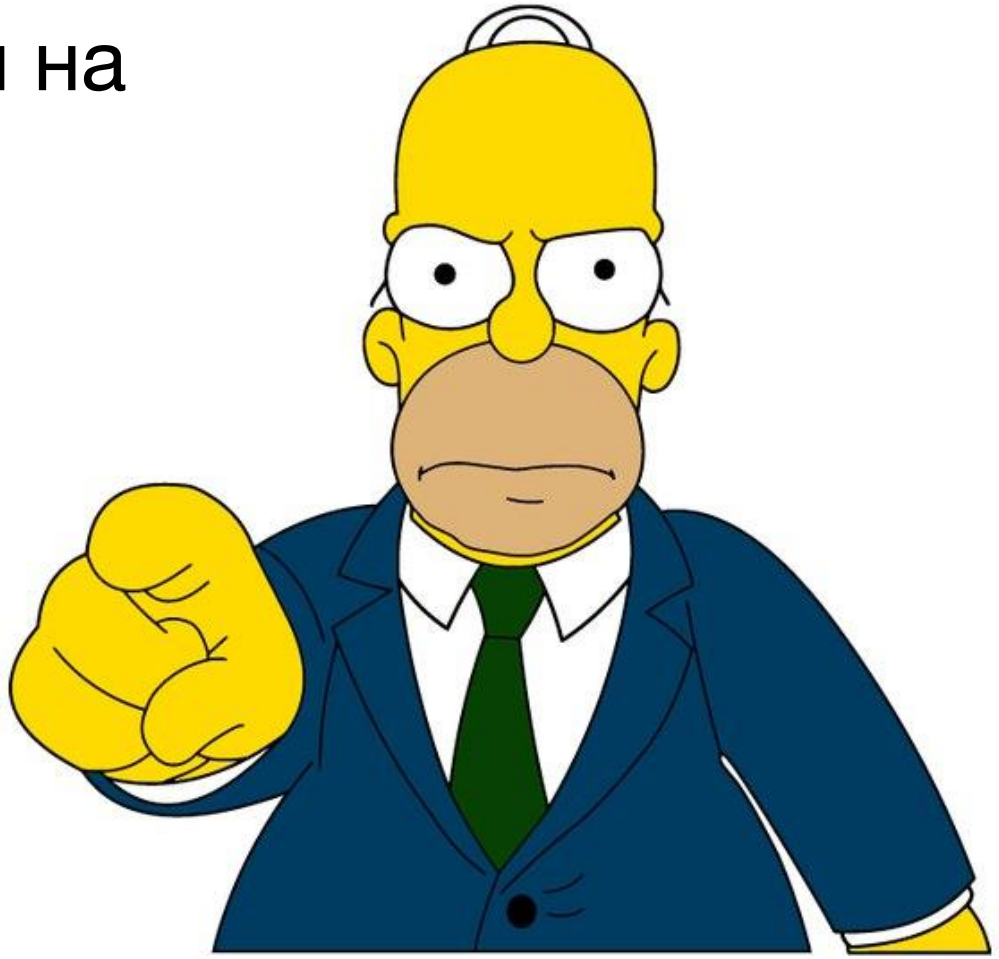
Занятие №7

Разработка приложений на Android

Кирилл Филимонов
Юрий Береза

Напоминание

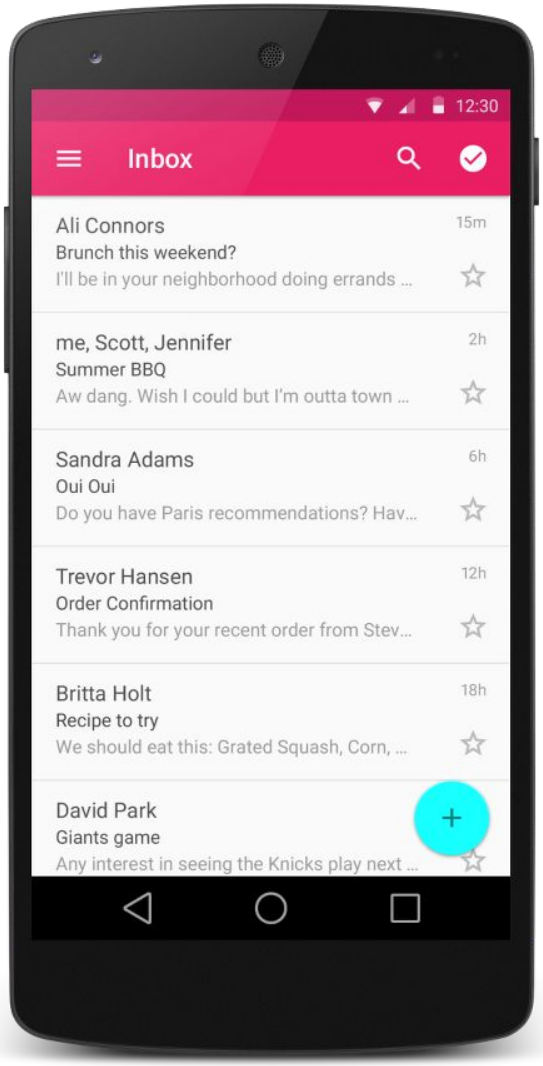
А ты отметился о присутствии на занятии?

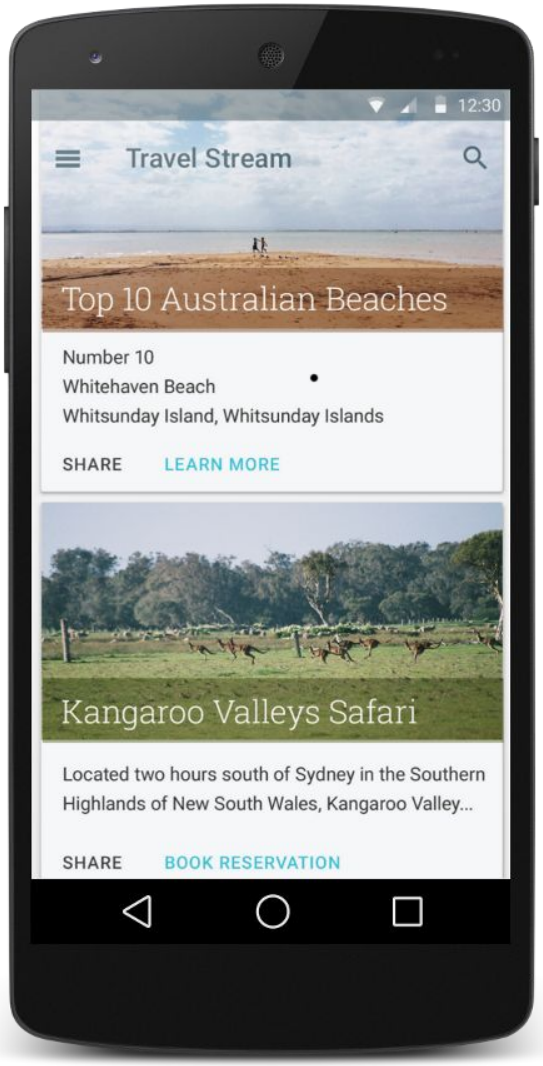


Agenda

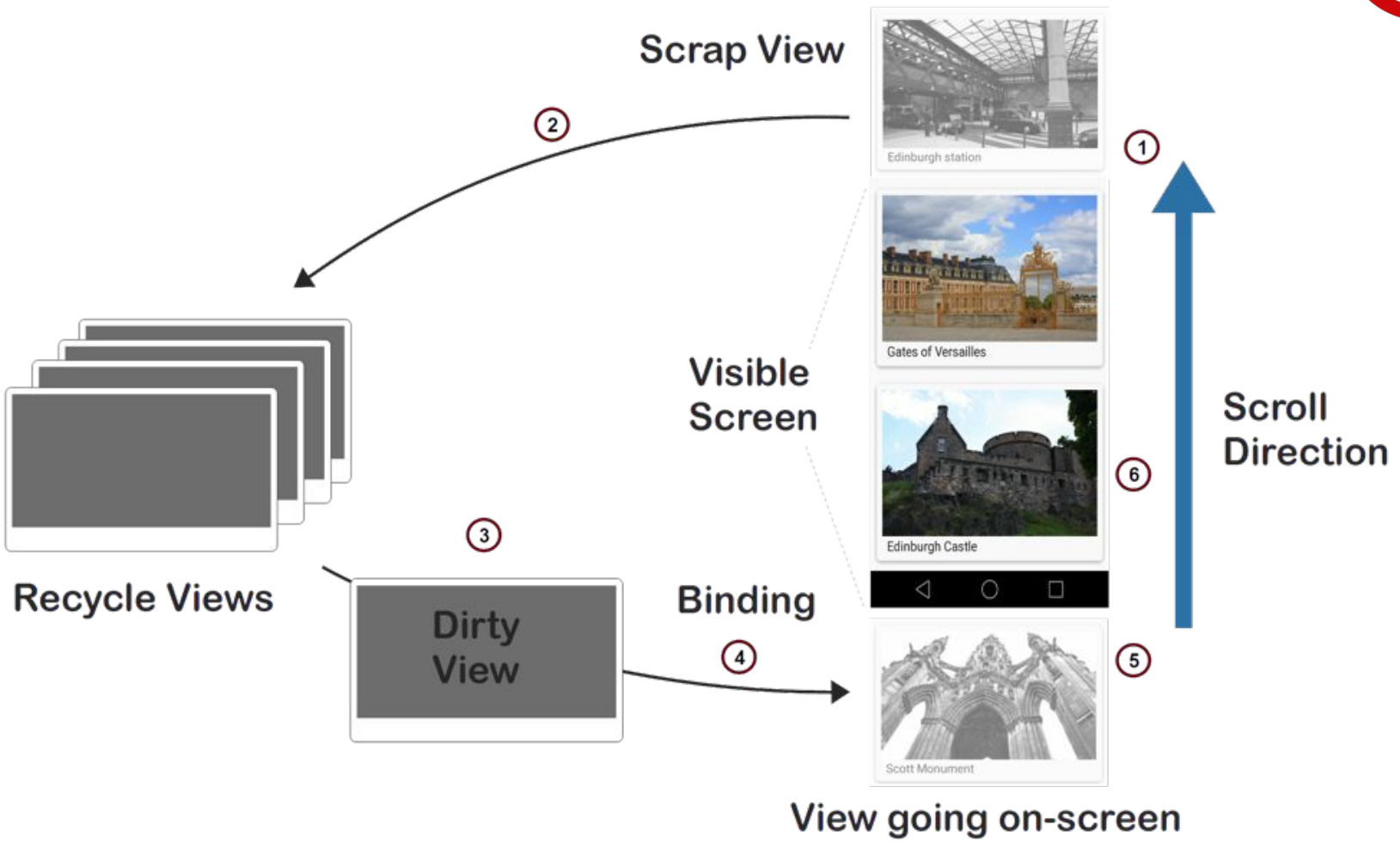
- RecyclerView
- Сервисы
- Job Scheduler
- Work Manager

RecyclerView

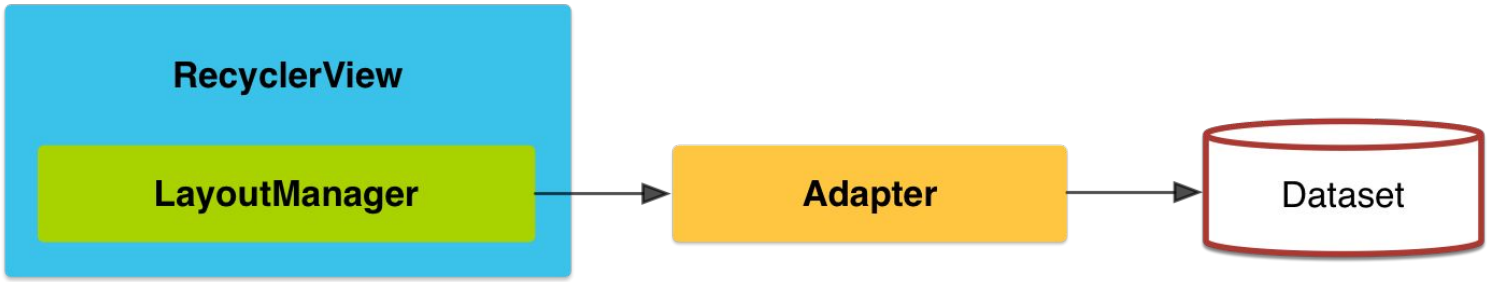




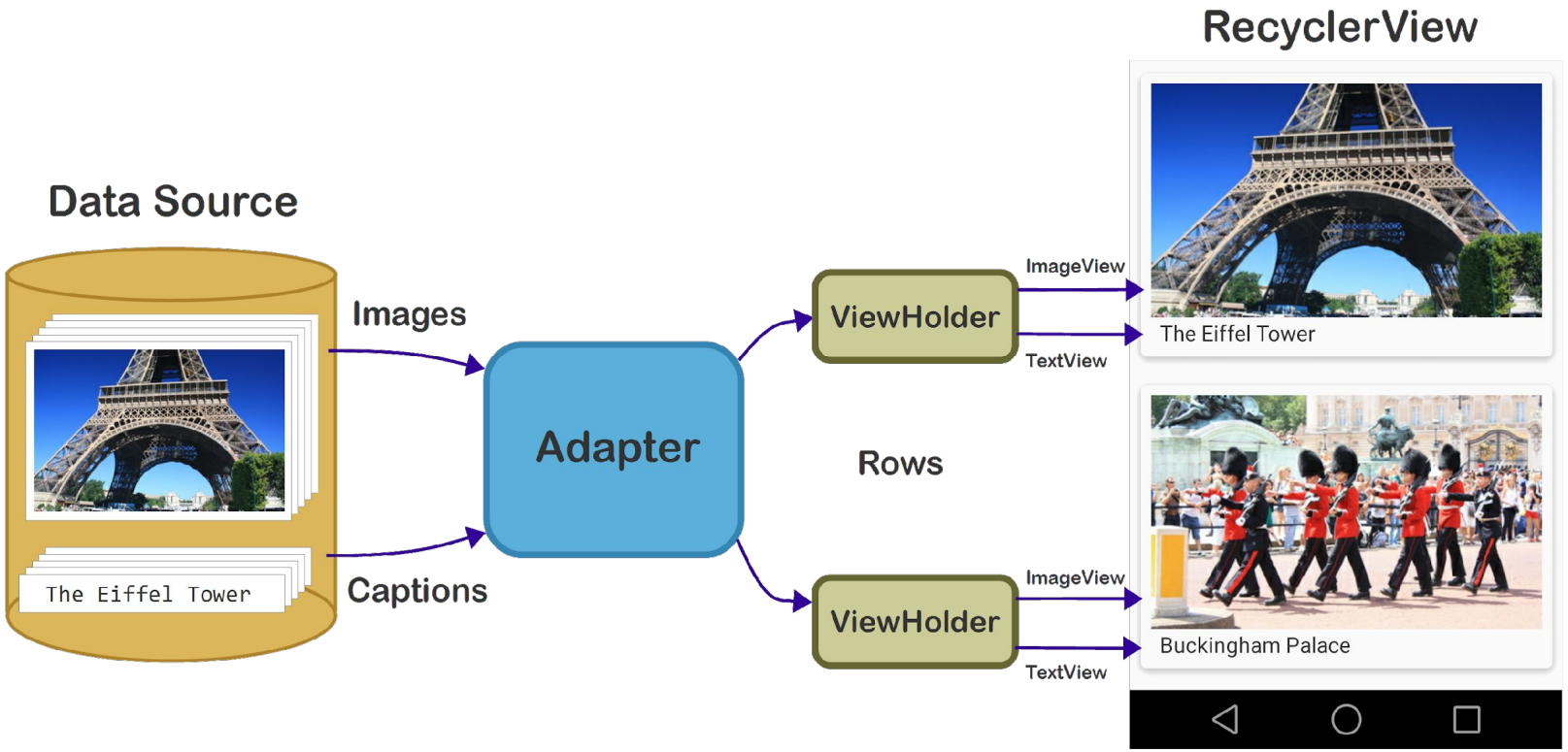
Цикл жизни элемента



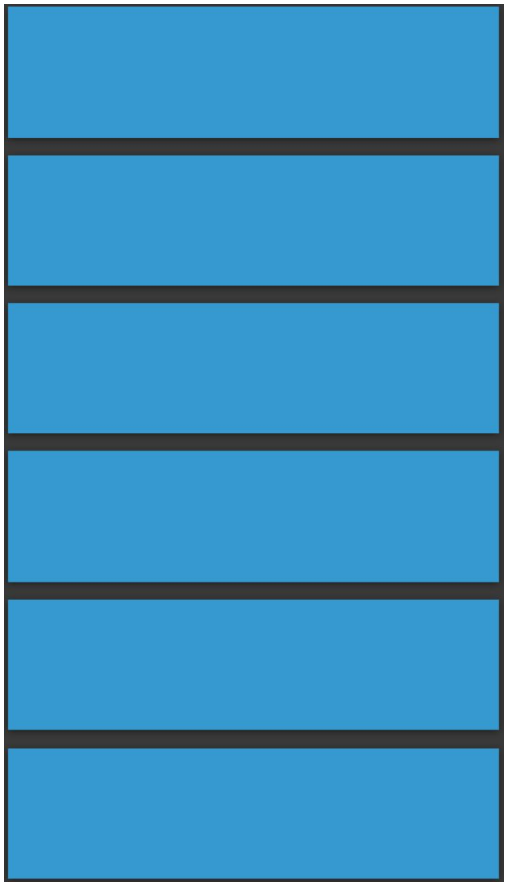
RecyclerView



RecyclerView

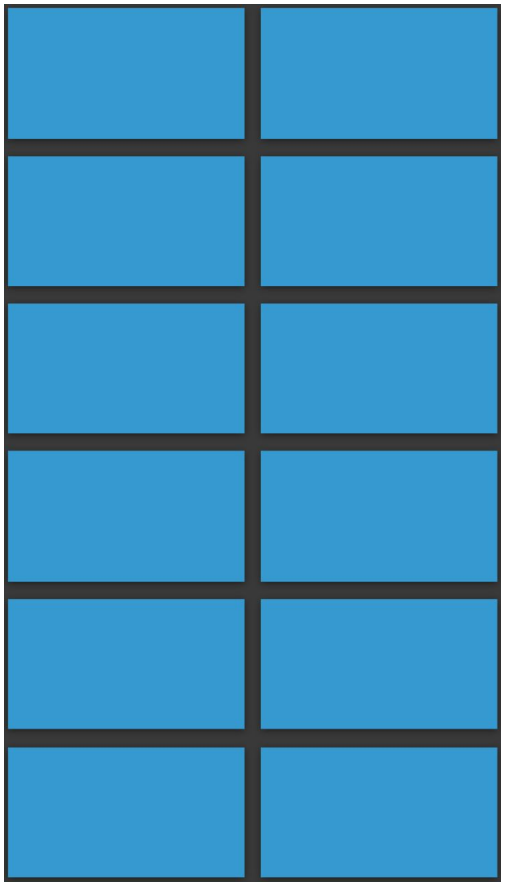


LinearLayoutManager



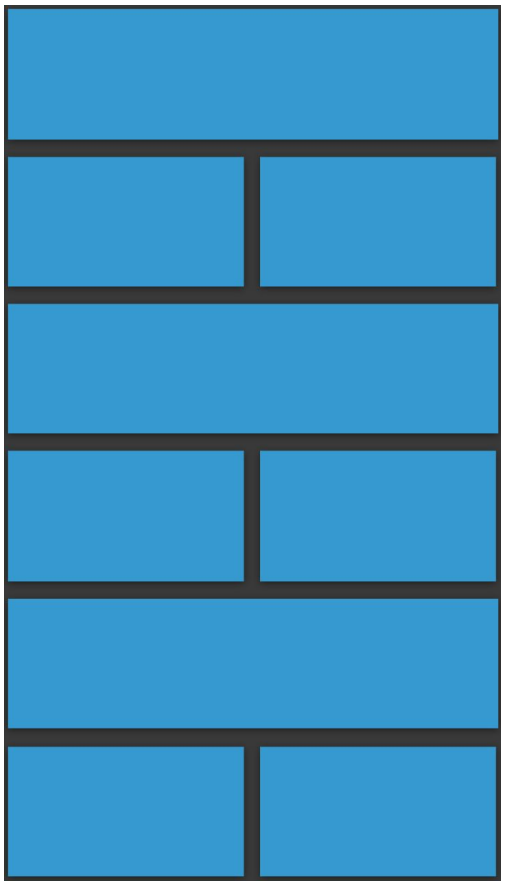
```
recycler.layoutManager = LinearLayoutManager(
    requireContext(),
    RecyclerView.VERTICAL,
    false
)
```

GridLayoutManager



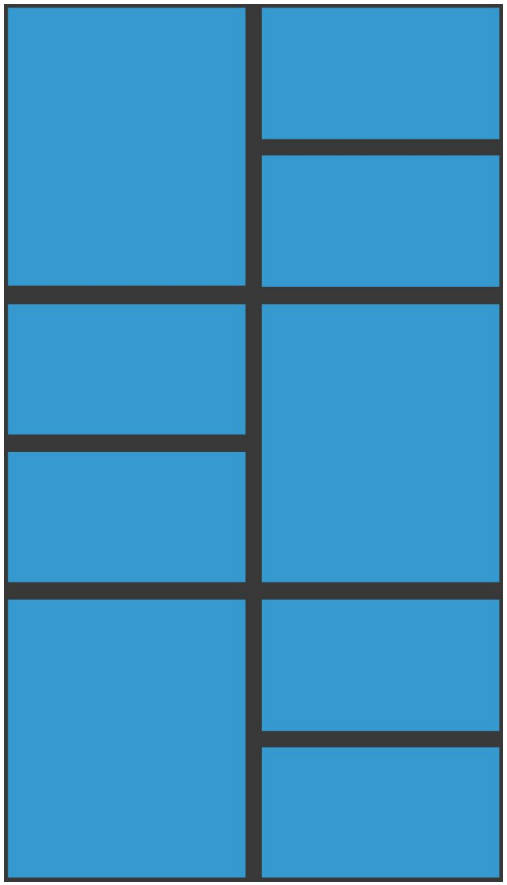
```
recycler.layoutManager = GridLayoutManager(
    requireContext(),
    2,
    RecyclerView.VERTICAL,
    false
)
```

GridLayoutManager



```
val layoutManager = GridLayoutManager(
    requireContext(),
    2,
    RecyclerView.VERTICAL,
    false
)
layoutManager.spanSizeLookup = object : GridLayoutManager.SpanSizeLookup() {
    override fun getSpanSize(position: Int): Int {
        return if (position % 3 == 0) 2 else 1
    }
}
recycler.layoutManager = layoutManager
```

StaggeredGridLayoutManager



```
recycler.layoutManager = StaggeredGridLayoutManager(
    2,
    RecyclerView.VERTICAL
)
```

Контейнеры

Адаптеры

RecyclerView.Adapter<T>

```
class SimpleListAdapter : RecyclerView.Adapter<SimpleViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SimpleViewHolder {

    }

    override fun getItemCount(): Int {

    }

    override fun onBindViewHolder(holder: SimpleViewHolder, position: Int) {

    }
}
```

RecyclerView.ViewHolder

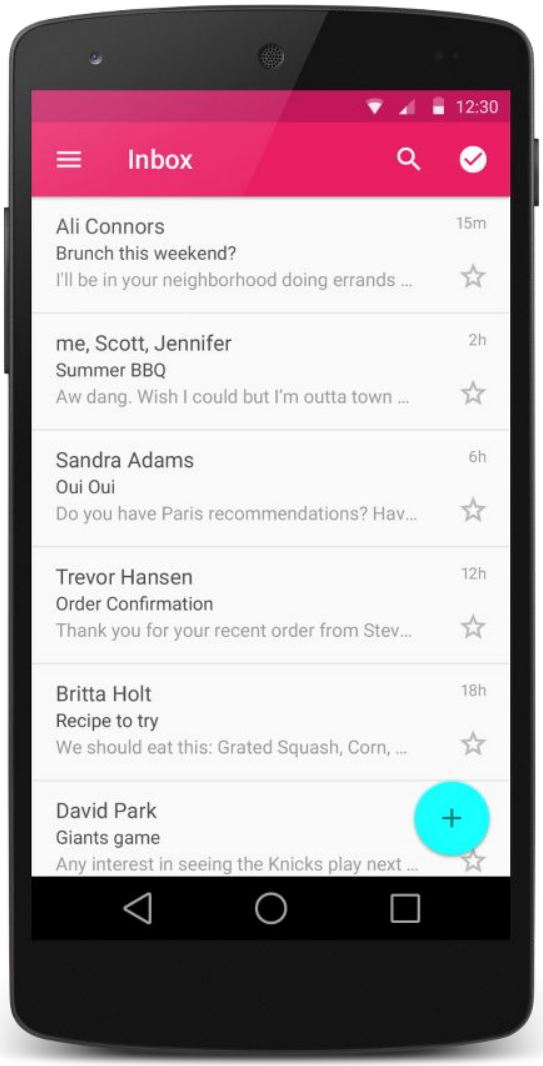
```
class SimpleViewHolder(view : View) : RecyclerView.ViewHolder(view) {
    private val title: TextView = view.findViewById(R.id.title)
    private val text: TextView = view.findViewById(R.id.text)

    fun setText(text : String) {
        this.text.text = text
    }

    fun setTitle(title : String) {
        this.title.text = title
    }
}
```

RecyclerView.Adapter - обновление данных

```
fun notifyDataSetChanged()

fun notifyItemChanged(position : Int)

fun notifyItemChanged(position : Int, payload : Any?)

fun notifyItemRangeChanged(start : Int, count : Int)

fun notifyItemRangeChanged(start : Int, count : Int, payload : Any?)

fun notifyItemInserted(position : Int)

fun notifyItemMoved(oldPosition : Int, newPosition : Int)

fun notifyItemRangeInserted(start : Int, count : Int)

fun notifyItemRemoved(position : Int)

fun notifyItemRangeRemoved(start : Int, count : Int)
```

Контейнеры

- GridView
- ListView
- Spinners



Demo

https://github.com/ybereza/technotrack

Сервис

- Что такое сервис
- Чем сервис отличается от потока
- Какие бывают сервисы
- Как с ними работать
- Что такое AIDL

Сервис

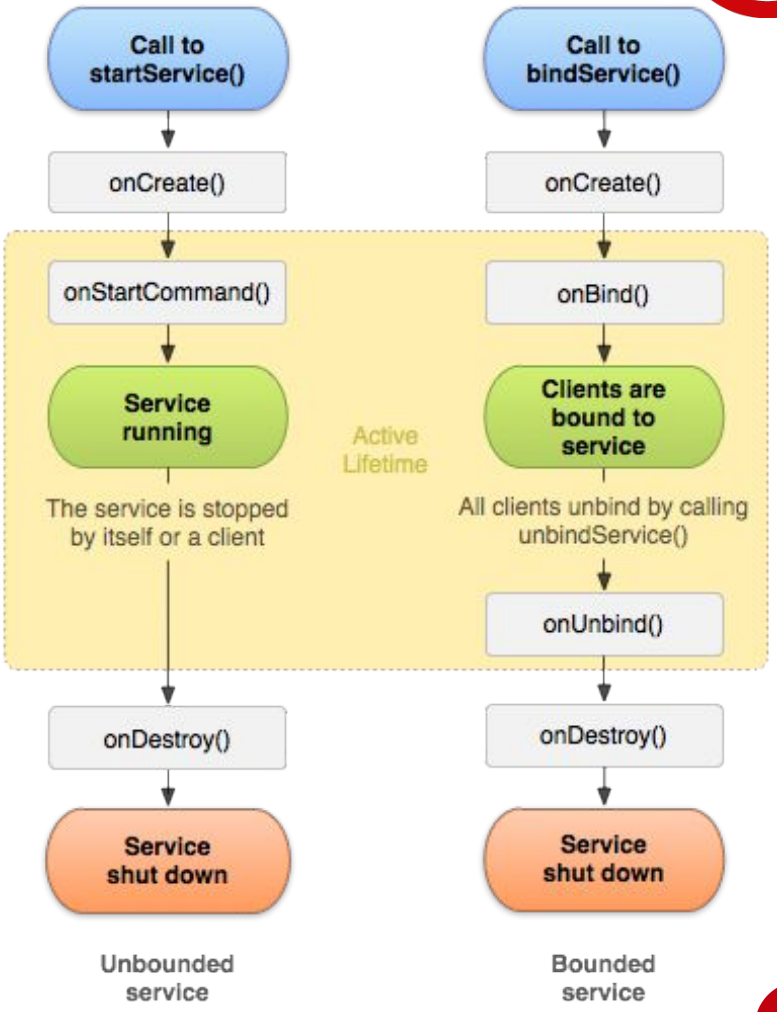
- Это компонент приложения
- Его нужно прописывать в манифесте
- Позволяет приложению осуществлять действия без взаимодействия с пользователем, в том числе в фоне
- Сервис по умолчанию запускается в основном потоке хост процесса (того кто его породил)
- **Сервис это не процесс** он запускается в рамках процесса хоста (есть исключения)
- **Сервис это не поток** он не создает отдельный поток

Сервис в манифесте

```
<service android:enabled=["true" | "false"]
         android:exported=["true" | "false"]
         android:icon="drawable resource"
         android:isolatedProcess=["true" | "false"]
         android:label="string resource"
         android:name="string"
         android:permission="string"
         android:process="string" >
    . . .
</service>
```

Типы сервисов

- Started (запущенные) — сервисы которые запускаются любым другим компонентом приложения и работают пока не остановят сами себя или кто-то не остановит их.
- Bound (связанные) — сервис который выступает в роли сервера в клиент-серверной архитектуре. Такой сервис создается при первом соединении(запросе) от другого компонента приложения. Сервис останавливается, когда отсоединится последний клиент.
- Сервис может быть одновременно и Started и Bound. Такой сервис способен «жить вечно» и обслуживать запросы клиентов



Запущенный сервис

```
override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    // handler start actions
    // ...
    // return startup type
}
```

- START_STICKY - сервис перезапустится, интент не будет доставлен повторно
- START_NOT_STICKY - сервис не перезапустится
- START_REDELIVER_INTENT - сервис перезапустится, интент будет повторно доставлен

Остановка сервиса:

```
// from service code
stopSelf()

// from outside
stopService(intent)
```

Запущенный сервис

Работа в фоне

```
private fun startWithNotification() {
    val notification = Notification.Builder(this, CHANNEL)
                .setContentText(getString(R.string.description))
                .setSmallIcon(R.id.smallIcon)
                .build()

    val notificationIntent = Intent(this, MainActivity.class)
    val pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0)
    notification.setLatestEventInfo(this, getText(R.string.notification_title),
                getText(R.string.notification_message), pendingIntent);
    startForeground(ONGOING_NOTIFICATION_ID, notification);
}
```

Остановка и сокрытие нотификации

```
stopForeground(true)
```

Привязанный сервис

```
class TestService : Service() {
    private val binder: IBinder
    private val allowRebind: Boolean

    override fun onBind(intent: Intent?): IBinder? {
        return binder
    }

    override fun onUnbind(intent: Intent?): Boolean {
        return allowRebind
    }

    override fun onRebind(intent: Intent?) {
        // client call bind after unbind, and unbind returns true
    }
}
```

Класс сервиса

```
class TestService : Service() {

    override fun onCreate() {
        super.onCreate()
        // initialize service components
    }

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        // start service action
        return super.onStartCommand(intent, flags, startId)
    }

    override fun onDestroy() {
        super.onDestroy()
        // finalize service components
    }

    override fun onBind(intent: Intent?): IBinder? {
        return null
    }
}
```

IntentService

- Подкласс обычного Service
- Создает новый поток для работы (HandlerTread)
- Все Intent обрабатывает в onHandleIntent
- Когда очередь пустая сам завершает работу

```
class TestService : IntentService("TestService") {

    override fun onCreate() {
        super.onCreate()
    }

    override fun onHandleIntent(intent: Intent?) {
        // handle intent data
        // perform action
    }

    override fun onDestroy() {
        super.onDestroy()
    }
}
```

AIDL

- В буквальном переводе – язык описания интерфейсов Android.
- Используется для описания композиции и декомпозиции Java объектов в примитивы ОС для непосредственно передачи между процесами.
- Импортировать нужно даже те aidl файлы, которые находятся в том же пакете.
- Ключевое слово oneway в декларации void метода означает что метод будет вызван асинхронно (клиент не дожидается его выполнения).
- Использовать можно только примитивы, String, List и Parcelable классы, объявленные в других aidl файлах.

Планирование задач

- Не требует выполнения немедленно

Планирование задач

- Не требует выполнения немедленно
- Периодическое выполнение

Планирование задач

- Не требует выполнения немедленно
- Периодическое выполнение
- Условное выполнение

Решение задачи планирования

- TimerTask

Решение задачи планирования

- TimerTask
- ScheduledExecutor

Решение задачи планирования

- TimerTask
- ScheduledExecutor
- Handler.postDelayed()

Решение задачи планирования

- TimerTask
- ScheduledExecutor
- Handler.postDelayed()
- AlarmManager

Решение задачи планирования

- TimerTask
- ScheduledExecutor
- Handler.postDelayed()
- AlarmManager
- SyncAdapter

Решение задачи планирования

- TimerTask
- ScheduledExecutor
- Handler.postDelayed()
- AlarmManager
- SyncAdapter
- JobScheduler

Решение задачи планирования

- TimerTask
- ScheduledExecutor
- Handler.postDelayed()
- AlarmManager
- SyncAdapter
- JobScheduler
- WorkManager (*alpha*)

Job Scheduler

Режим работы Android до API 21 (Android 1.6 - 4.4)

Job Scheduler

Режим работы Android начиная с API 21 (Android 5.0)
[Android Doze Mode]

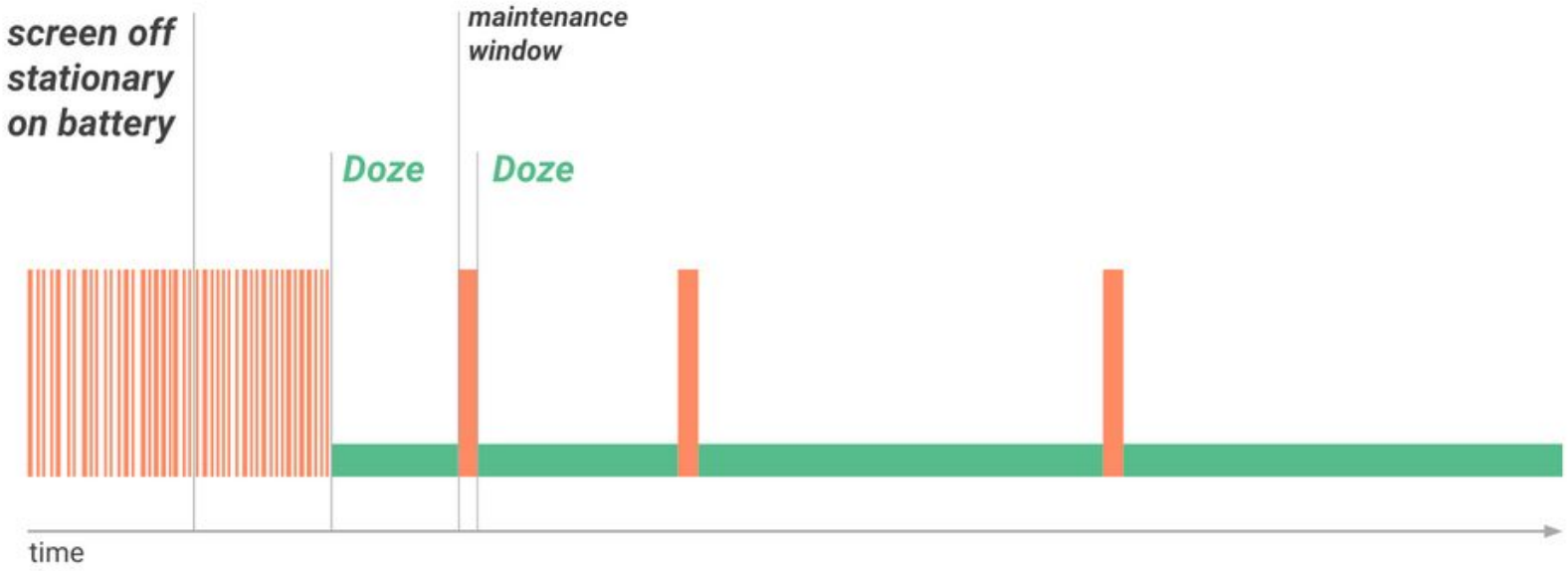


Figure 1. Doze provides a recurring maintenance window for apps to use the network and handle pending activities.

Job Scheduler

App StandBy

- распределение приложений по группам
- политики доступа к ресурсам в группах

- Active - используются в настоящий момент
- Working set - используются регулярно
- Rare - используется редко
- Never - были установлены, но ни разу не запускались

JobScheduler

Pros:

- Оптимизация расходования заряда батареи
- Оптимизация использования памяти
- Оптимизация сетевого взаимодействия

JobScheduler

Pros:

- Оптимизация расходования заряда батареи
- Оптимизация использования памяти
- Оптимизация сетевого взаимодействия

Cons: API >= 21

WorkManager:

- преимущества JobScheduler
- Обратная совместимость со старыми API

JobScheduler: условный запуск

- Устройство заряжается
- Сетевое подключение без ограничений
- Устройство не в роуминге
- Устройство в состоянии Idle
- Не ранее, чем через…
- Не позднее, чем…
- В течение...

Job

```
class ServiceJob : JobService() {

    override fun onStartJob(params: JobParameters): Boolean {
        // вызывается при начале выполнения задачи
        // !!! выполняется на Main Thread !!!
        // если выполнение задачи завершено в до вызова return
        // вернуть false, иначе true
    }

    override fun onStopJob(params: JobParameters): Boolean {
        // вызывается в случае остановки выполнения задачи
        // например, после вызова cancel или при изменении условий
        // если задачу необходимо перезапланировать, вернуть true
    }
}
```

Job

AndroidManifest.xml:

```
<service android:name=".ServiceJob"
         android:permission="android.permission.BIND_JOB_SERVICE"/>
```

JobInfo

```
val builder = JobInfo.Builder();
builder.setBackoffCriteria(initialBackoff, policy)
                .setExtras(bundle)
                .setMinimumLatency(latency)
                .setOverrideDeadline(maxDelay)
                .setPeriodic(period)
                .setPersisted(isPersisted)
                .setRequiredNetworkType(type)
                .setRequiresCharging(isRequiresCharging)
                .setRequiresDeviceIdle(isRequiresIdle)
                .setTriggerContentMaxDelay(maxDelay)
                .setTriggerContentUpdateDelay(delay);
val info = builder.build();
```

JobScheduler

```
val scheduler = context.getSystemService(Context.JOB_SCHEDULER_SERVICE) as JobScheduler
scheduler.schedule(info)
```

JobIntentService

- Подкласс обычного Service
- Работает в отдельном потоке
- На Android O выполняет Job на JobScheduler
- На Android pre-O с помощью startService
- Обработка интента происходит в onHandleWork
- Когда очередь пустая сам завершает работу

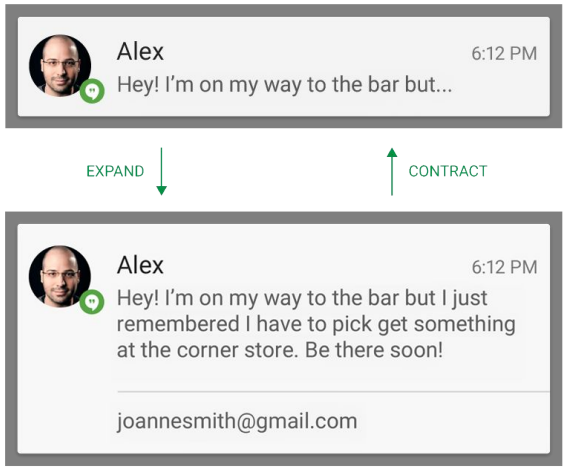
```
class TestService : JobIntentService() {
    override fun onHandleWork(intent: Intent) {
        // do service work
    }
}
```

Demo

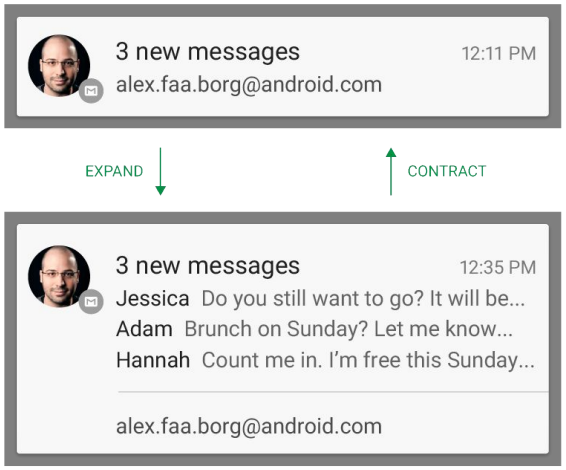
https://github.com/kirillF/JobSchedulerDemo

Нотификации

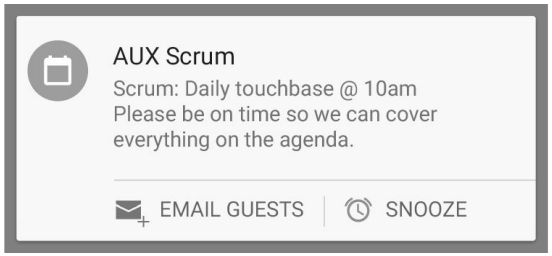
- Можно группировать сообщения
- Можно взаимодействовать с нотификацией
- Можно отображать расширенную информацию

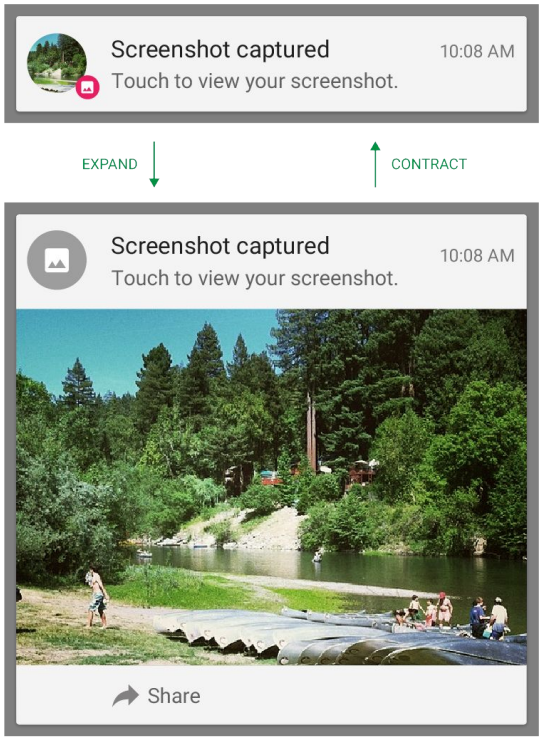


TEXT



INBOX





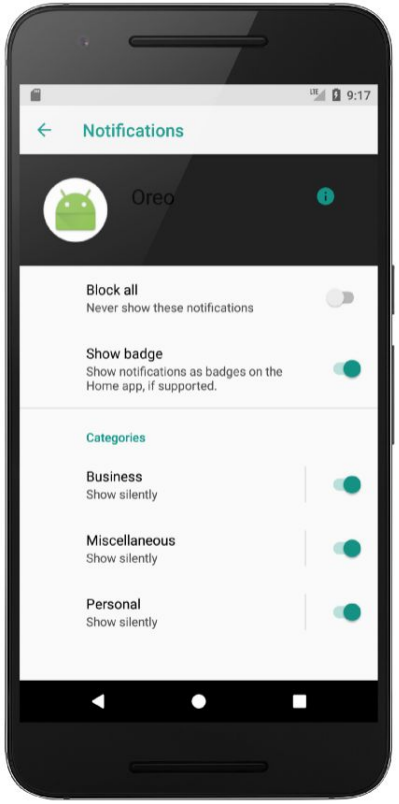
IMAGE

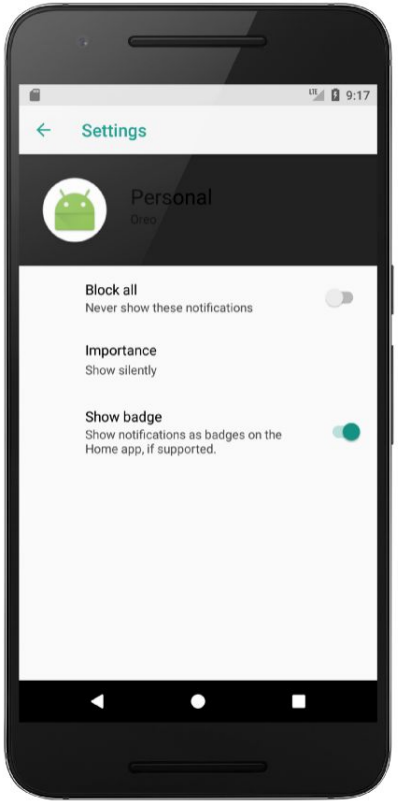
Нотификации

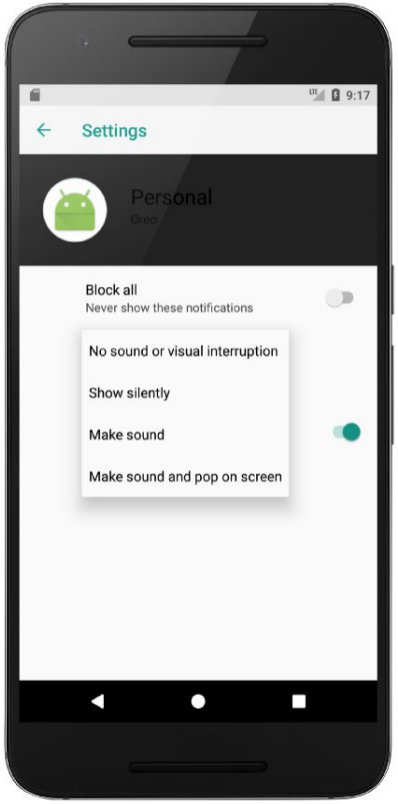
- Бываю перекрывающими (Head-up notification)
- Используются для критических вещей
- Могут отображаться на lock экране
- Могут иметь приватную и публичную часть

Каналы

Требуется targetSdk = 26 (Android 8)







Рекомендации и требования

- По клику на уведомление, пользователю должен открываться соответствующий экран приложения. В некоторых случаях достаточно, чтобы по клику уведомление просто убиралось.
- Указание времени события в уведомлении, также является хорошим подходом.
- Рекомендуется схожие события складывать в одно уведомление, а не отображать на каждое событие своё.
- Всегда убирать из статус-бара уведомления, с которыми пользователь уже ознакомился и произвел соответствующие действия.
- Показывать маленькое превью уведомления при его создании в свёрнутом статус-баре
- Позволять пользователю отключать уведомления в настройках приложения.
- Использовать иконки, обозначающие принадлежность уведомления определённому приложению. Иконки делать монохромными
- В случае, если событие требует непосредственной реакции пользователя — вместо уведомлений использовать диалоги.

Простая нотификация

- Создаем билдер
- Заполняем параметры нотификации
- Запускаем ее

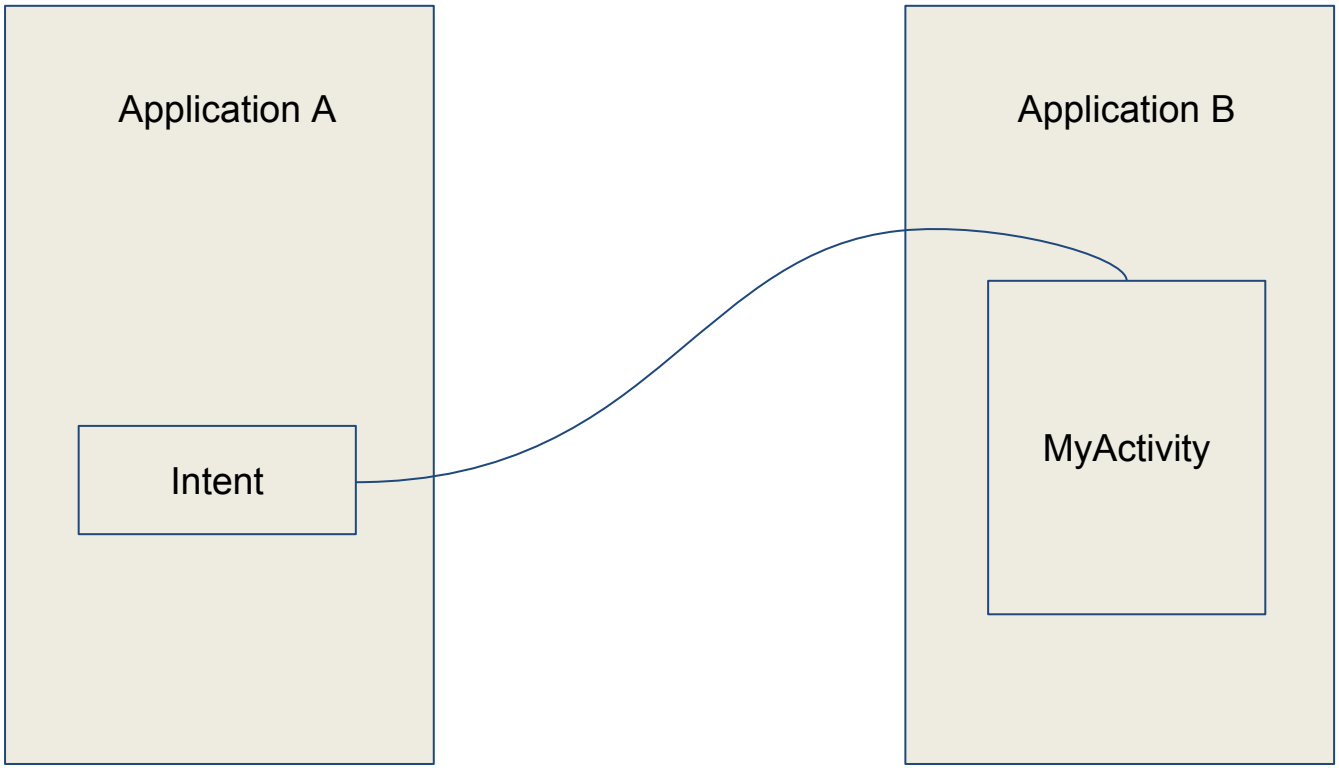
```
fun createSimpleNotification(context: Context) {
    val notificationIntent = Intent(context, MainActivity::class.java)
    val contentIntent = PendingIntent.getActivity(
        context,
        0, notificationIntent,
        PendingIntent.FLAG_CANCEL_CURRENT
    )

    val msg = "Это тестовая нотификация"

    val builder = NotificationCompat.Builder(context, "default")

    builder.setContentIntent(contentIntent)
        .setSmallIcon(R.drawable.notification)
        .setLargeIcon(R.mipmap.ic_launcher)
        .setWhen(System.currentTimeMillis())
        .setShowWhen(true)
        .setAutoCancel(true)
        .setContentTitle("Напоминание")
        .setContentText(msg)

    val nc = builder.build()

    val nm = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    nm.notify(1, nc)
}
```

Какие есть возможности

- Группировать
- Звук
- Вибрация
- Управлять светодиодом
- Можно обновляет состояние по ID
- Можно удалять
- Можно создавать свой layout для нотификации
- Можно задать иконку и картинку для отображения
- Можно задавать приоритет
- Можно задавать категорию
- Можно задать персону (контакт) по URI

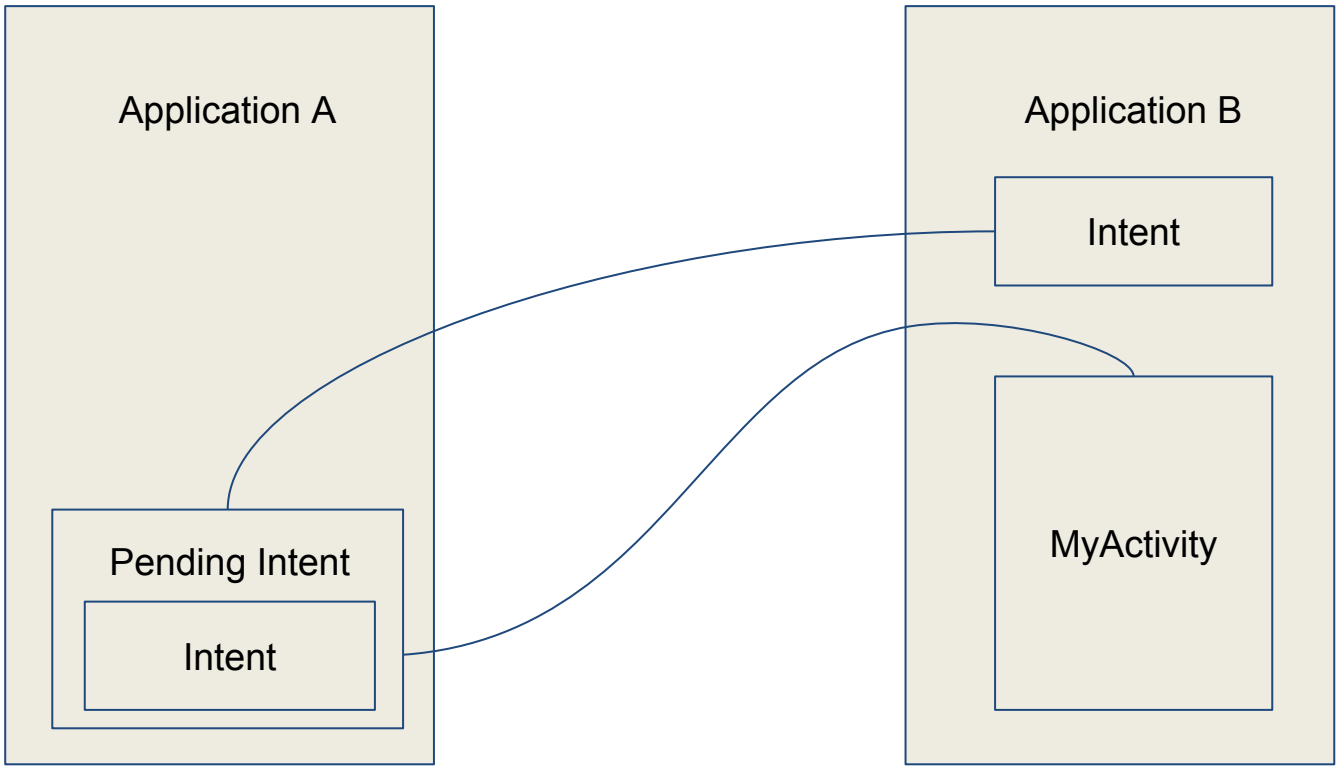
Intent

```
val intent = Intent("com.my", "com.my.MyActivity");
context.startActivity(intent);
```



Pending Intent

```
val intent = Intent("com.my", "com.my.MyActivity");
PendingIntent.getActivity(context, 123, intent,
                          PendingIntent.FLAG_CANCEL_CURRENT);
```



Pending Intent

Помните! Если вы хотите совершить два действия, вам необходимо создать два Intent и два PendingIntent с ДВУМЯ РАЗНЫМИ ID

```
val intent1 = Intent("com.my", "com.my.MyActivity");
val intent2 = Intent("com.my", "com.my.MyActivity2");

PendingIntent.getActivity(context, 123, intent1,
                          PendingIntent.FLAG_CANCEL_CURRENT);

PendingIntent.getActivity(context, 321, intent2,
                          PendingIntent.FLAG_CANCEL_CURRENT);
```

- FLAG_NO_CREATE
- FLAG_CANCEL_CURRENT
- FLAG_UPDATE_CURRENT
- FLAG_IMMUTABLE

ТЕХНОТРЕК

Спасибо за внимание!

Кирилл Филимонов - Kirill.Filimonov@gmail.com

Юрий Береза - ybereza@gmail.com