ТЕХНОТРЕК

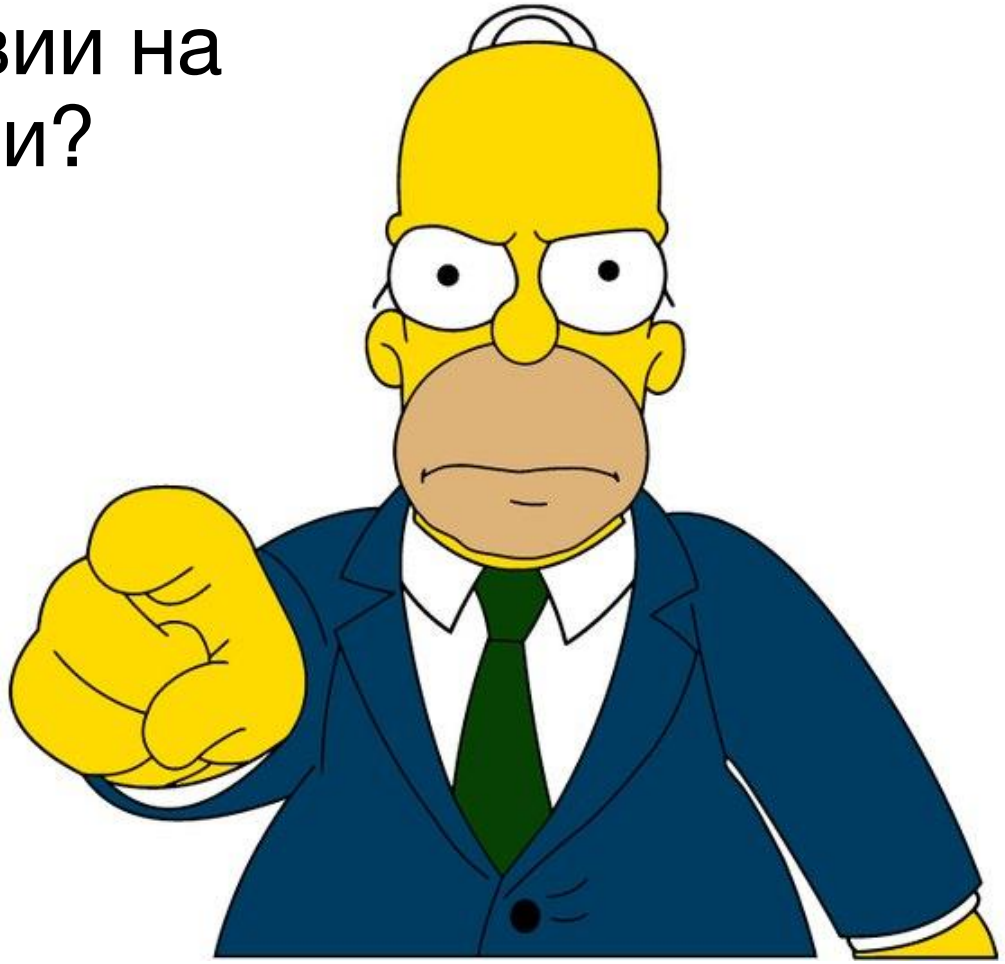
Занятие №4

Разработка приложений на Android

Юрий Береза
Кирилл Филимонов

Напоминание

А ты отметился о присутствии на занятии?



Agenda

0. Generics
1. Activity и его жизненный цикл
2. Task и Back stack
3. Implicit intent и explicit intent
4. Фрагменты и их жизненный цикл
5. Support Library
6. Android X
7. GUI элементы
8. Layouts
9. Gravity

Generics

- Параметризованные функции и свойства
- Параметризованные классы
- Ограничения типов
- Стирание типов (type erasure)
- Овеществляемые (reified) типы
- Вариантность

Generics: функции и классы

```
class Service<Req, Rep>(private val executor: RequestExecutor<Rep, Req>) {

    fun apply(request: Req): Future<Rep> {
        return executor.execute(request)
    }

    fun <T> convert(converter: (Req) -> T): List<T> {
        val list = mutableListOf<T>()
        for (request in executor.requests) {
            list.add(converter(request))
        }
        return list
    }
}
```

Generics

- Тип B - подтип типа A, если значение типа B можно использовать везде, где ожидается значение типа A
- Если B - подтип типа A, то A - супертип для B

Особенность типов в Kotlin

- Int - **является** подтипом Int?
- Int? - **не является** подтипом Int

Generics

- Обобщенный класс C называют **инвариантным** по типовому параметру, если для любых разных типов A и B, C<A> не является подтипом C<B>
- Обобщенный класс C называют **ковариантным** по типовому параметру, если для типов A и B, где A - подтип B, C<A> является подтипом C<B>
- Обобщенный класс C называют **контравариантным** по типовому параметру, если для типов A и B, где B - супертип A, C<B> является подтипом C<A>

Generics: ковариантность

- отношение тип-подтип сохраняется
- типовой параметр используется только на исходящей позиции (ключевое слово **out**)

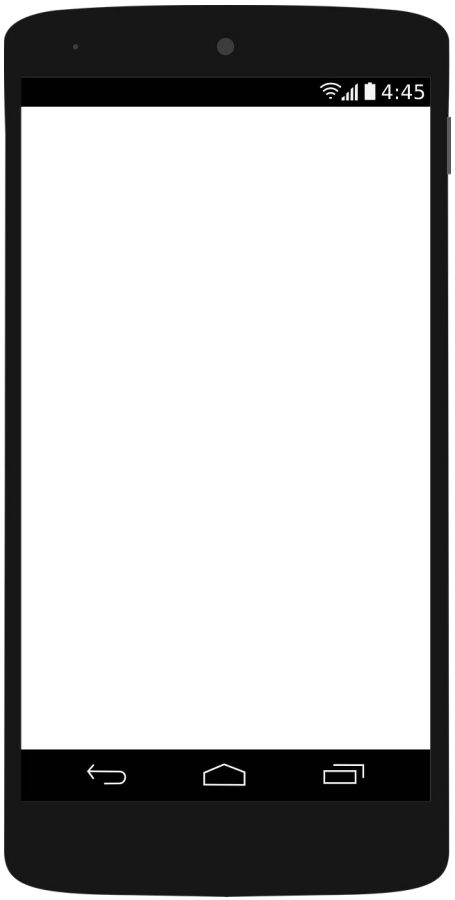
```
interface List<out T>: Collection<T>  {
    operator fun get(index: Int): T

    //...
}
```

Generics: контравариантность

- отношение тип-подтип изменяется на противоположное
- ключевой параметр используется на входящей позиции (ключевое слово **in**)

```
interface Comparator<in T> {
    fun compare(e1: T, e2: T): Int
}
```

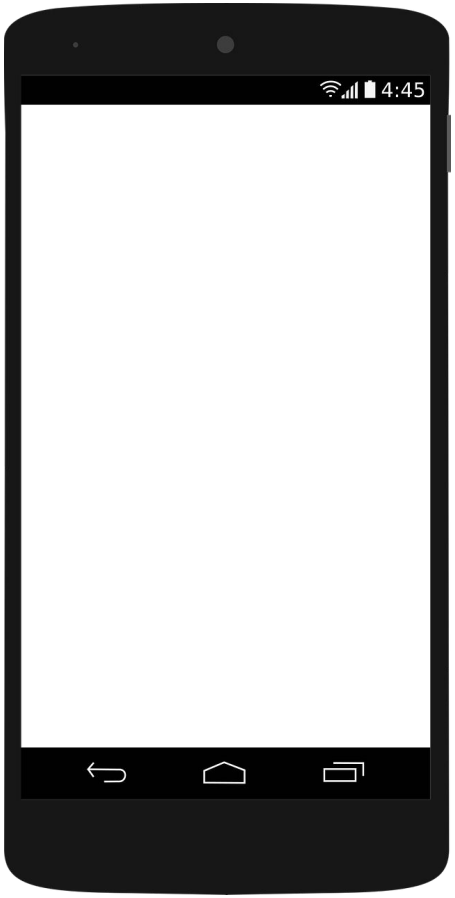
Activity



Объявление Activity в AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ru.mail.sample" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="ru.mail.sample.MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

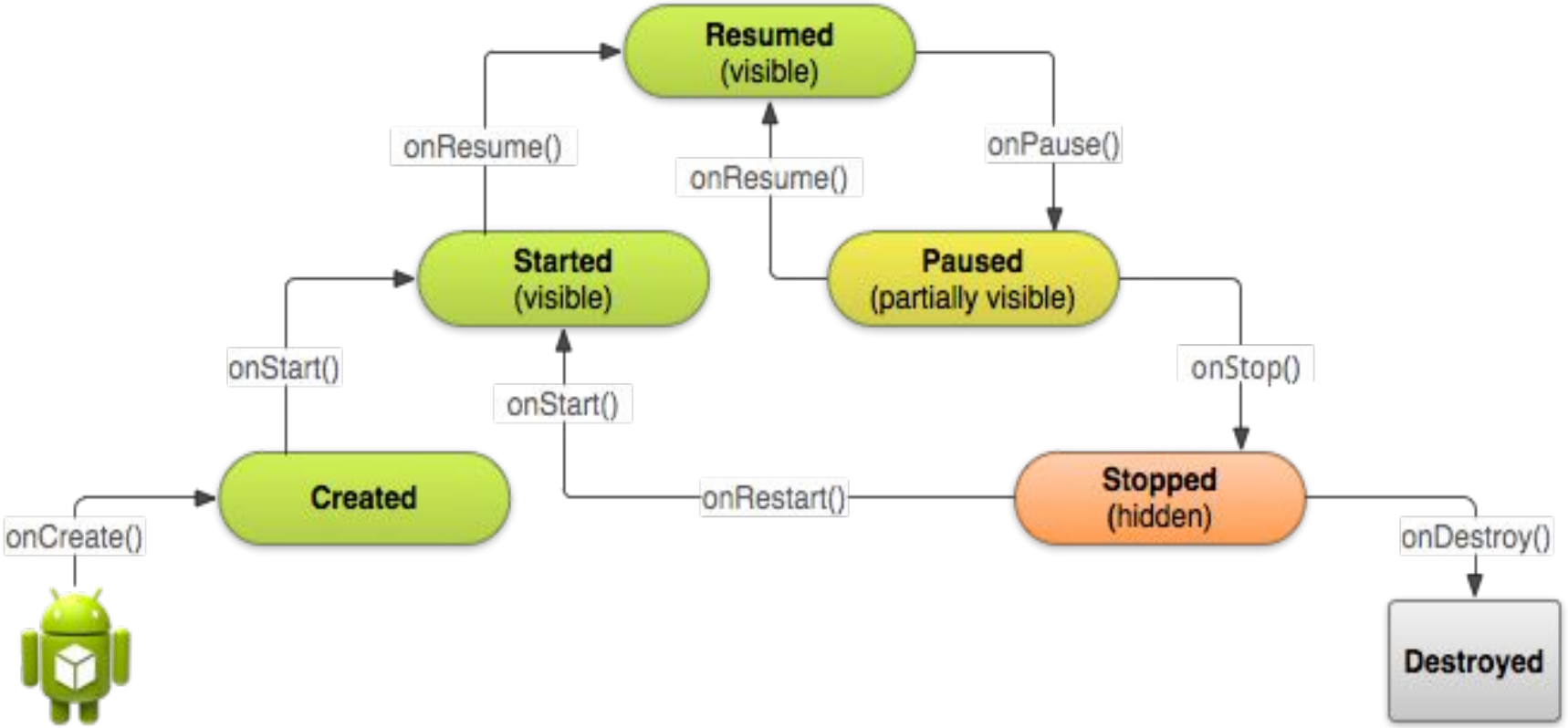
Activity



Объявление класса MainActivity

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

}
```

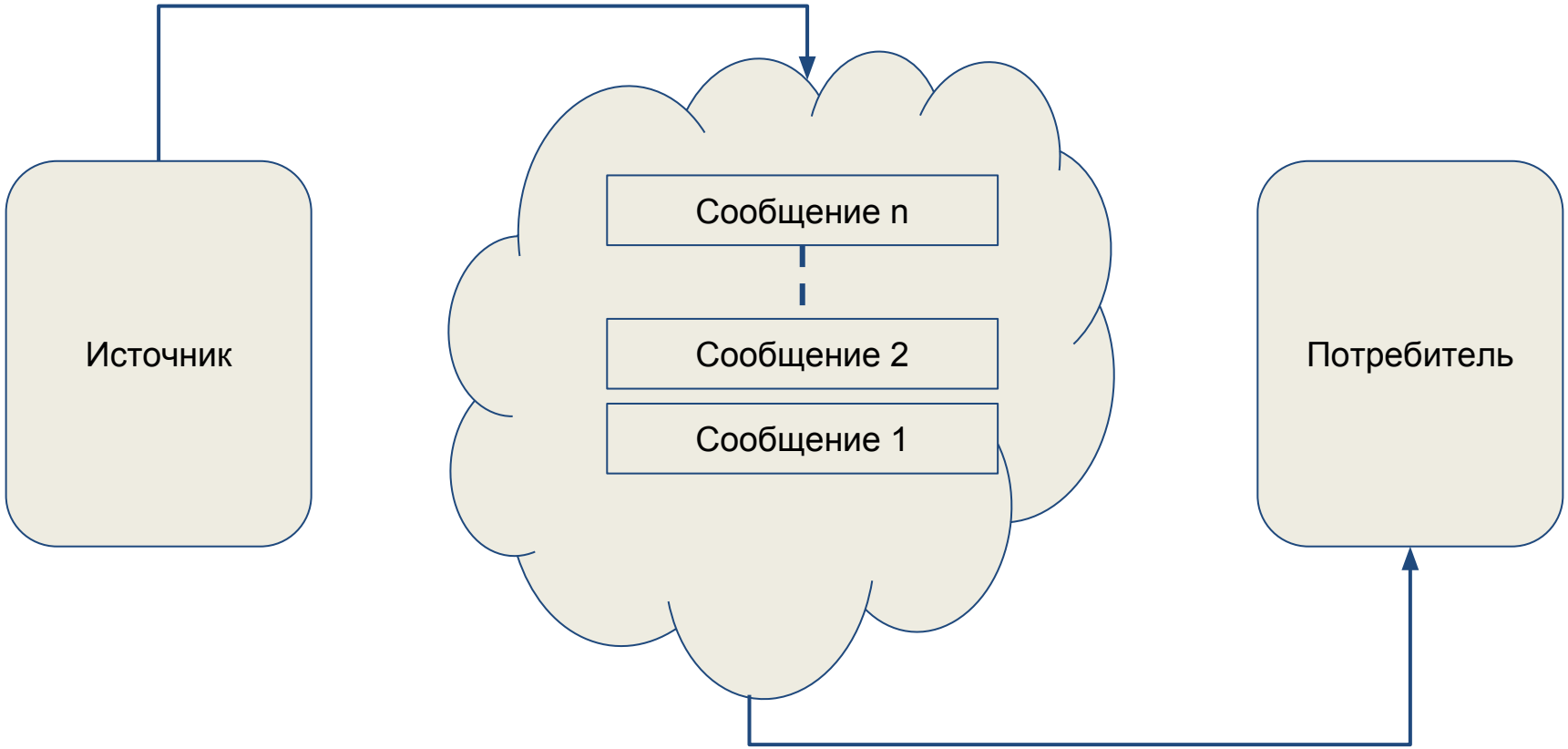
Activity и его жизненный цикл



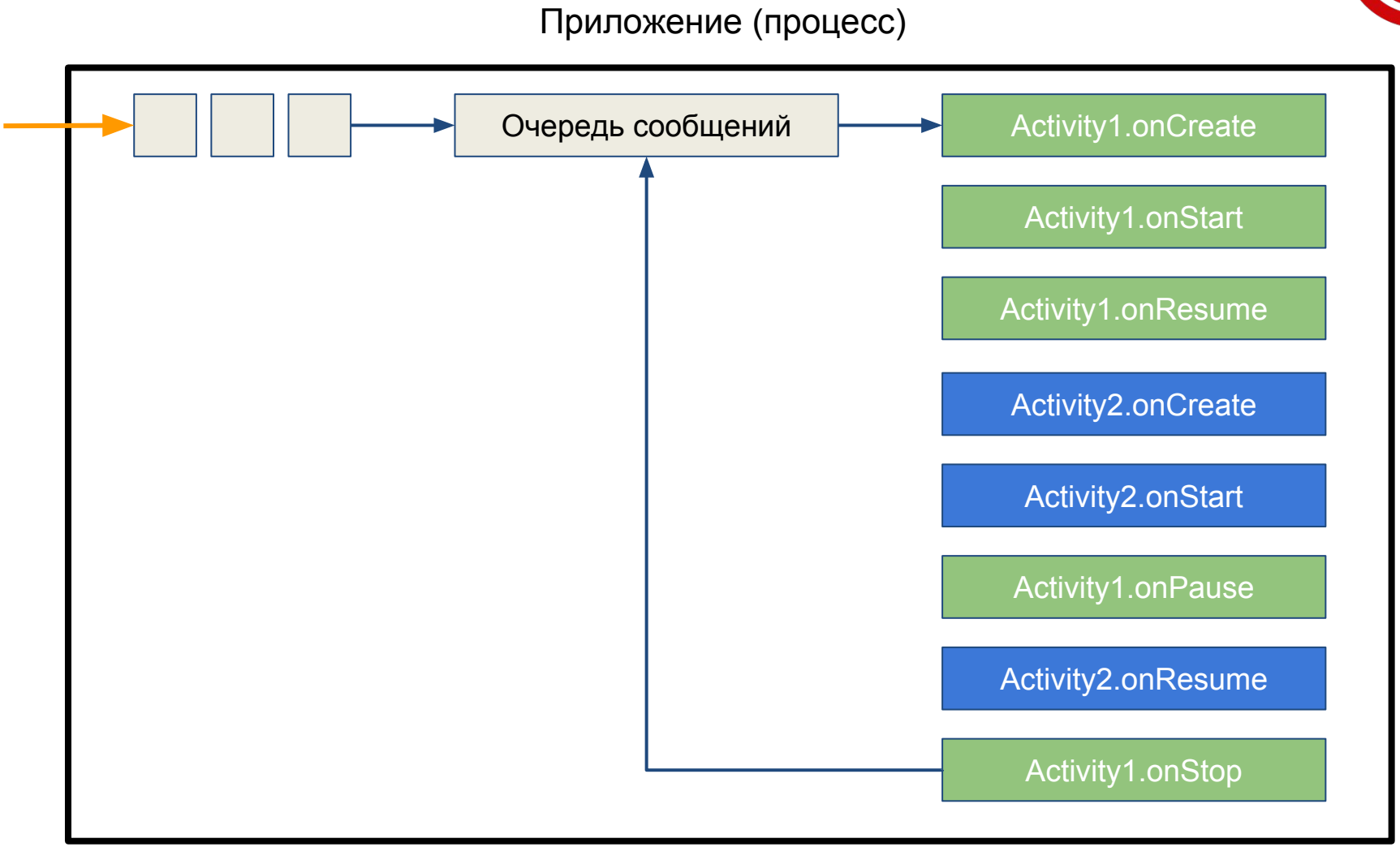
Activity и его жизненный цикл

- `onCreate(Bundle saveInstanceState)`
  - Вызывается, когда создается activity
  - Получает сохраненное состояние (если оно есть)
  - Как конструктор
- `onResume()`
  - Вызывается перед тем, как activity станет видимым пользователю
- `onPause()`
  - Вызывается перед тем, как у другой activity вызовется onResume()
  - Здесь все завершающие операции
  - Не делать долгих операций!
- `onStop()`
  - Вызывается, когда activity уже не видима пользователю
- `onDestroy()`
  - Вызывается перед уничтожением activity

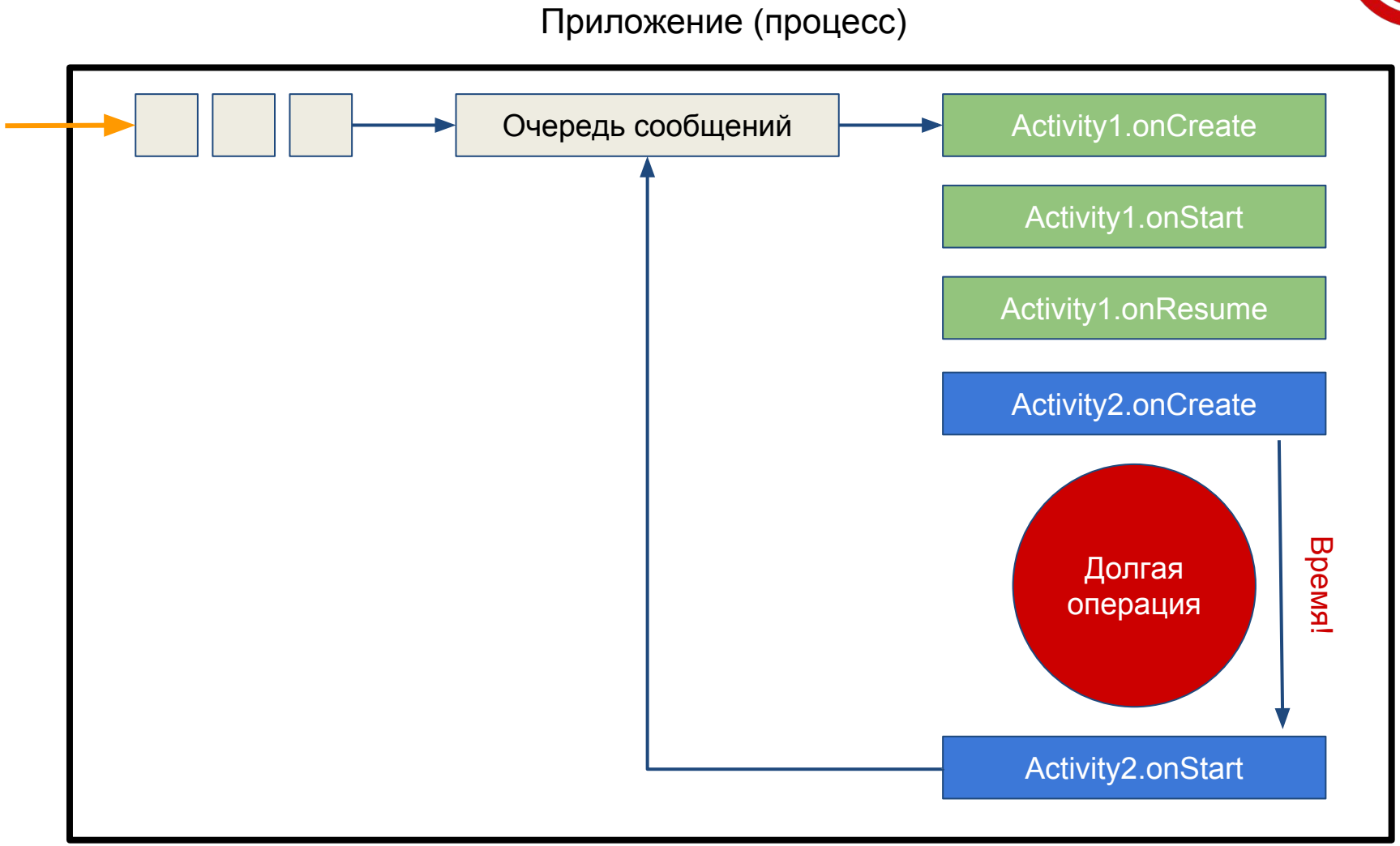
Очередь сообщений



Приложение и Activity



Приложение и Activity



Activity - сохранение состояния

Когда уничтожается Activity:

- был вызван метод finish()
- пользователь нажал кнопку “Назад” на телефоне
- нехватка памяти, система освобождает ресурсы
- поворот экрана

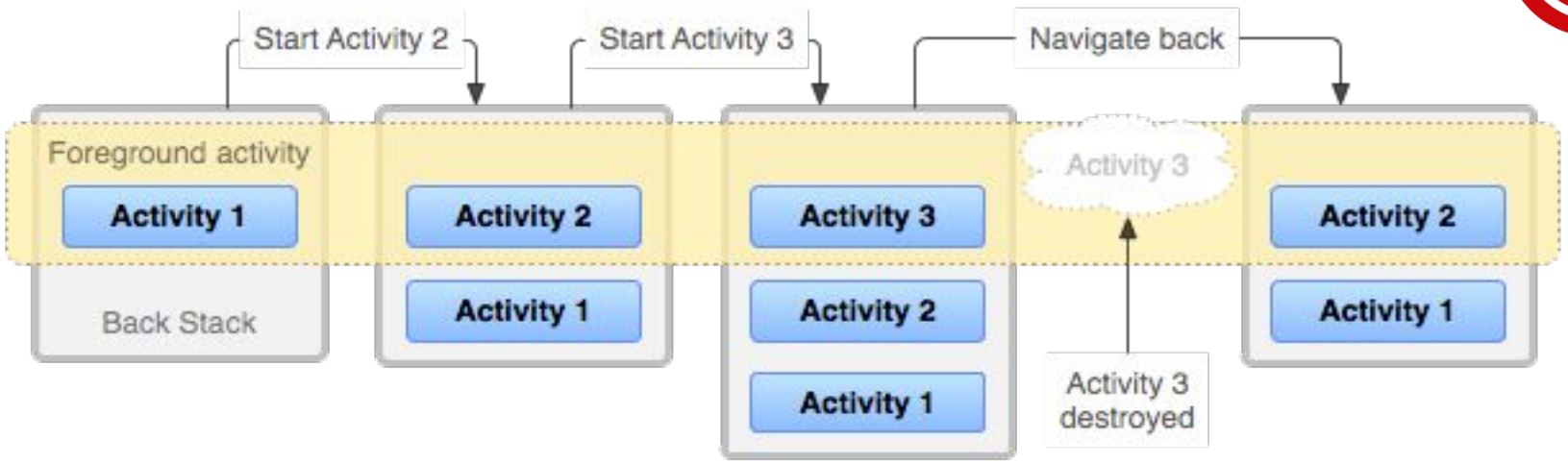
- Сохранение - `onSaveInstanceState(state : Bundle)`
  - Не забывайте вызвать метод `super.onSaveInstanceState(state)`
- Восстановление - `onRestoreInstanceState(state : Bundle)` или `onCreate(state : Bundle?)`
  - Не забывайте вызвать метод `super.onRestoreInstanceState(state)`

Важно! Если был вызван метод onSaveInstanceState, это еще не означает, что будет вызван метод onRestoreInstanceState

Сохранение состояния

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    savedInstanceState?.run {
        defaultText = getString("SAVED_TEXT_STATE") ?: ""
    }
}

override fun onSaveInstanceState(outState: Bundle) {
    outState.putString("SAVED_TEXT_STATE", defaultText)
    super.onSaveInstanceState(outState)
}
```

Task и Back Stack



- Task - набор Activity
- Каждый task имеет свой стек
- Одна и та же Activity может иметь сколько угодно экземпляров в стеке
- Новая activity пушится в стек, а у предыдущей вызывается onStop()
- По кнопке back верхняя activity достается из стека и уничтожается, а у activity под ней
- вызывается onResume()

Explicit Intent

- Адресуются конкретному компоненту (с помощью component name)
- Обычно используются для запуска внутренних компонентов

```
val intent = Intent(this, WordsWriter::java.class);
startActivity(intent);
```

Implicit Intent

- Не имеют конкретного адресата
- Обычно используются для запуска компонентов сторонних приложений
- Система находит наиболее подходящие компоненты (или несколько)

```
val intent = Intent(Intent.ACTION_VIEW)
intent.data = Uri.parse("http://mail.ru")
startActivity(intent)
```

Получение результата от Activity

Activity 1 (получатель)

```
fun showSecondActivity() {
    val intent = Intent(this, MainActivity::class.java)
    startActivityForResult(intent, 12345)
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    when {
        requestCode == 12345 && resultCode == Activity.RESULT_OK -> handleResult(data)
    }
}
```

Activity 2 (отправитель)

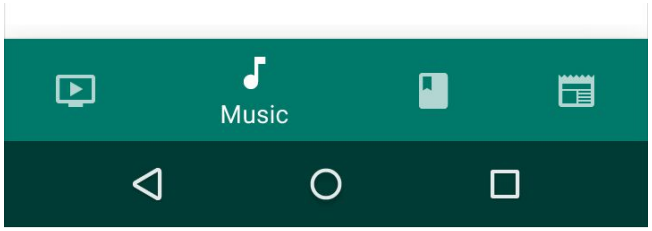
```
val data = Intent()
data.putExtra("result", "dataString")
setResult(Activity.RESULT_OK, data)
```

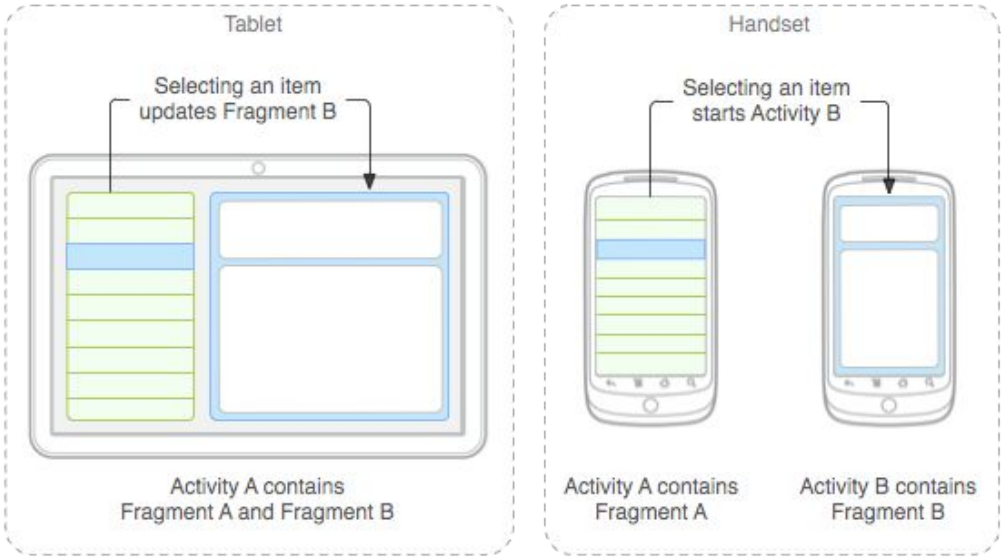
Фрагменты

- Что такое фрагмент
- Жизненный цикл
- Менеджер фрагментов
- Как использовать

Что такое фрагменты

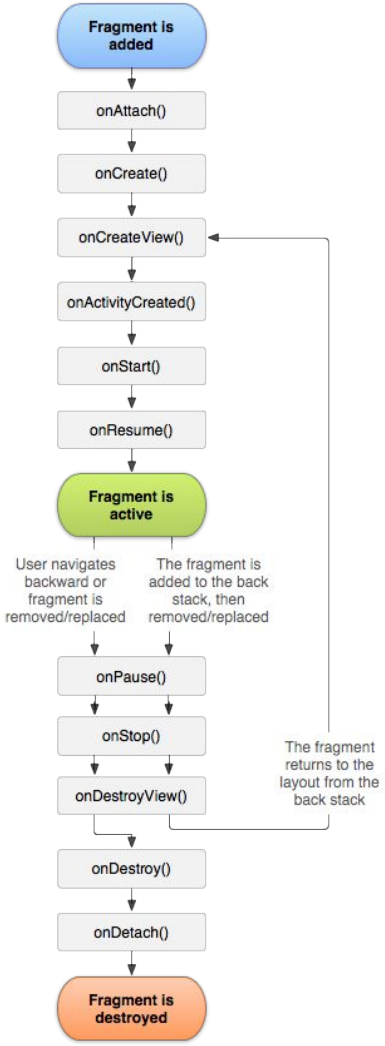
- Fragments – представляет собой часть пользовательского интерфейса в Activity
- Можно в одном Activity использовать несколько фрагментов эмулирую многопанельный интерфейс
- Жизненный цикл фрагмента привязан к циклу activity





Жизненный цикл фрагмента

- Код почти как в Activity
- onAttach – фрагмент прикрепили к activity
- onCreate - аналог аналогичной функци у activity, но пока нет доступа к элементам UI
- onCreateView – тут создается View фрагмента который надо отдать системе
- onActivityCreate – фрагмент создан есть доступ к UI
- onDetach – фрагмент откреплися от activity



Как работать с фрагментами

Статические фрагменты

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="horizontal"
              android:layout_width="match_parent"
              android:layout_height="match_parent">

    <fragment
              android:name="ru.mail.technotrack.mainui.fragments.LayoutFragment"
              android:id="@+id/list"
              android:layout_weight="1"
              android:layout_width="0dp"
              android:layout_height="match_parent" />
    <fragment
              android:name="ru.mail.technotrack.mainui.fragments.WidgetsFragment"
              android:id="@+id/viewer"
              android:layout_weight="2"
              android:layout_width="0dp"
              android:layout_height="match_parent" />
</LinearLayout>
```

Как работать с фрагментами

Динамические фрагменты

- Создать класс унаследованный от Fragment реализовать нужные методы
- Добавить его с помощью метода add или replace у SupportFragmentManager

```
val fragment = MyFragment()
val transaction = supportFragmentManager.beginTransaction()
transaction.replace(fragmentContainer, fragment)
transaction.addToBackStack("MyFragment")
transaction.commit()
```

FragmentManager

- Позволяет управлять фрагментами и историей фрагментов
- Позволяет добавить, удалить или заменить фрагмент в Activity
- Позволяет работать со стеком транзакций фрагментов
- Вся работа идет через FragmentTransaction
- Для того чтобы действие свершилось надо вызвать commit у транзакции.

Фрагменты и Android X

- Для того чтобы использовать фрагменты, необходимо подключить android x или support library. Activity наследовать от AppCompatActivity
- Чтобы получить FragmentManager надо у activity вызвать getSupportFragmentManager
- Не используйте android.app.Fragment и android.app.FragmentManager

Подробнее о Android X

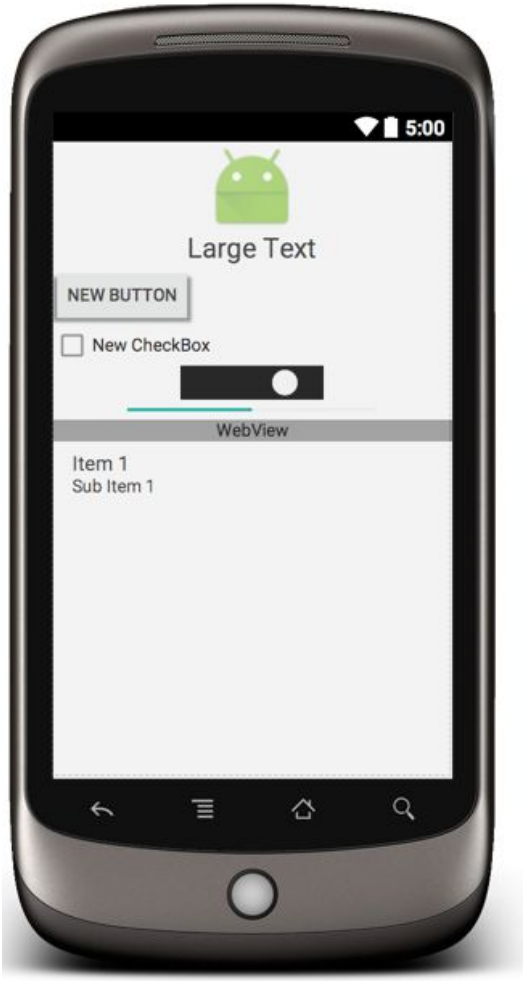
- Позволяет на старых версиях Android использовать новый API
- Делает код проще без ветвления по версиям устройств
- Делает приложение единым для всех версий Android
- Экономит много времени программиста
- Легко добавляется в приложение в виде готовых библиотек

Поддержка разных версий API

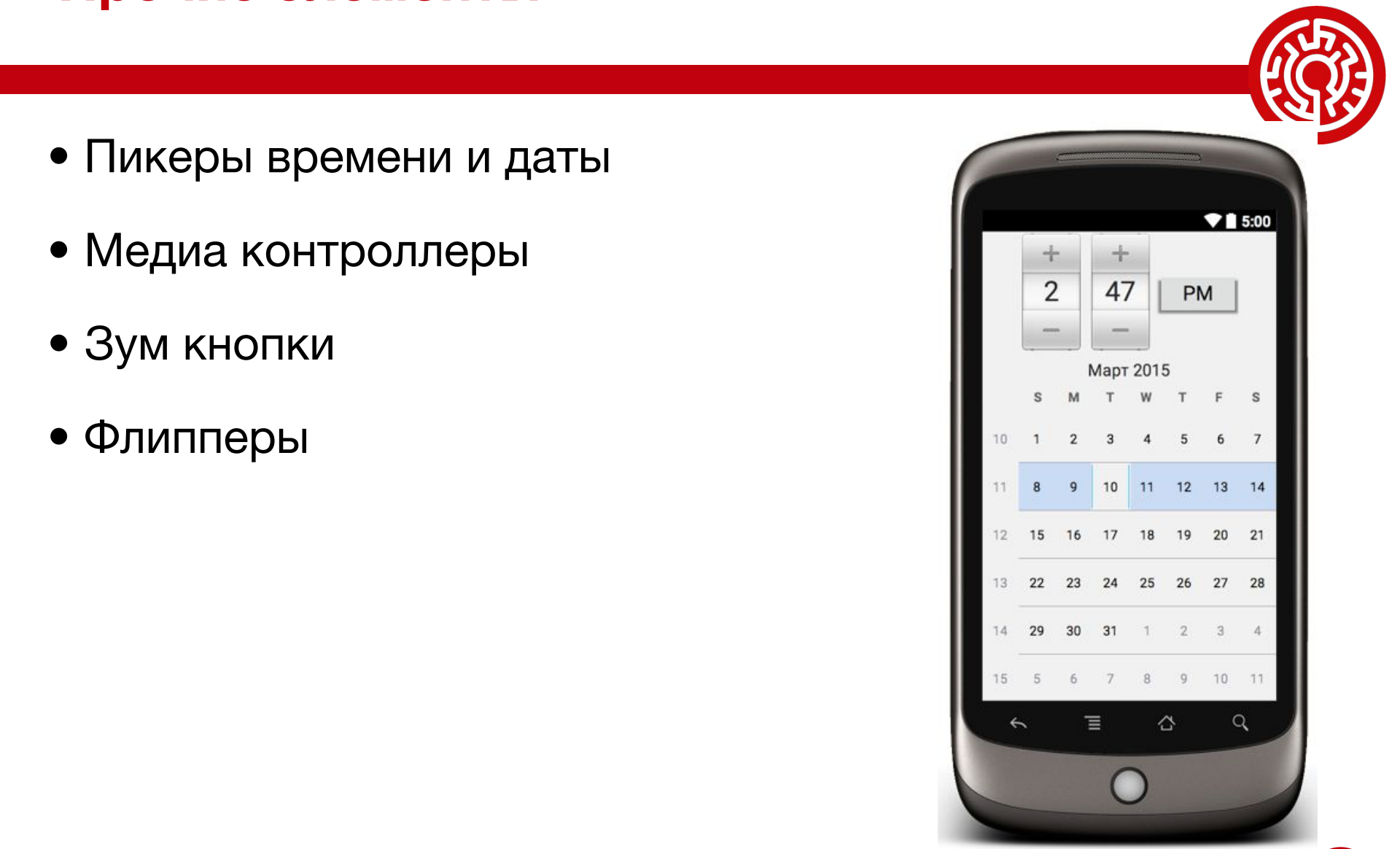
```
fun View.getClipToOutlineCompat() : Boolean {
    return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) clipToOutline else false
}

fun View.postOnAnimationCompat(runnable: () -> Unit) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN)
        postOnAnimation(runnable)
    else postDelayed(runnable, 16)
}
```

Простые UI элементы

- View
- Image
- Button
- ImageButton
- TextView
- RadioButton
- CheckBox
- Switch
- ProgressBar
- WebView
- Spinner



Прочие элементы



- Пикеры времени и даты
- Медиа контроллеры
- Зум кнопки
- Флипперы

Разбираем GUI layout

- FrameLayout
- LinearLayout
- RelatedLayout
- ConstraintLayout

FrameLayout

- тип верстки, внутри которого может отображаться только один элемент в строке.
- если внутри FrameLayout вы поместите несколько элементов, то следующий будет отображаться поверх предыдущего.

LinearLayout

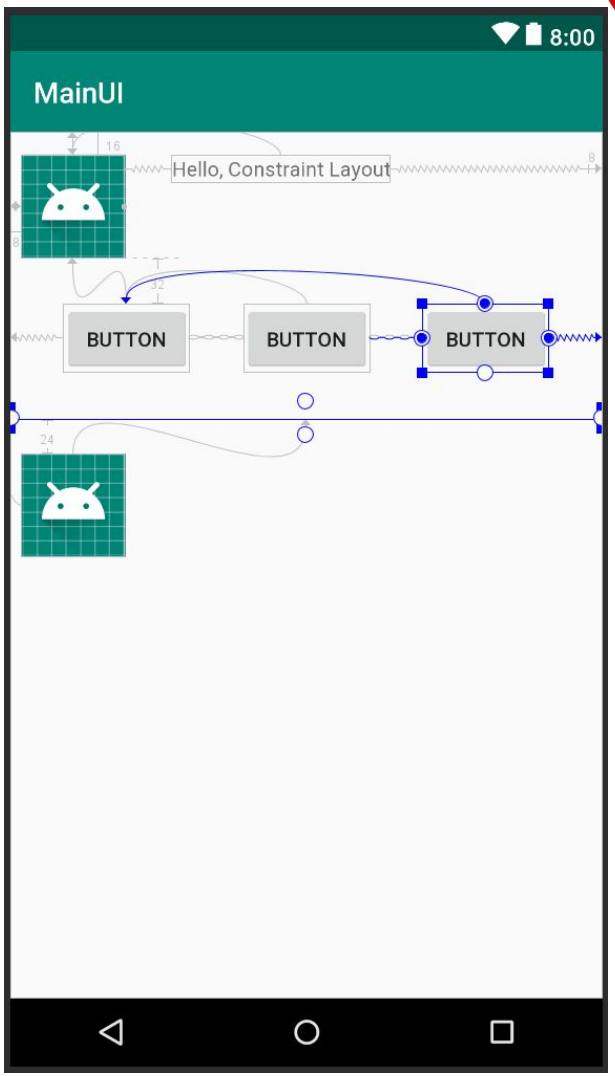
- тип верстки, при котором область верстки делится на строки, и в каждую строку помещается один элемент.
- разбиение может быть вертикальное или горизонтальное, тип разбиения указывается в атрибуте LinearLayout android:orientation.

RelativeLayout

- Тип верстки, при котором позиционирование элементов происходит относительно друг друга и относительно главного контейнера.
- За то, каким образом будут позиционироваться элементы, отвечают следующие группы атрибутов:
  - Атрибуты позиционирования относительно контейнера
  - Атрибуты позиционирования относительно других элементов.
  - Выравнивание относительно других элементов.
- Необходимо два прохода по элементам!

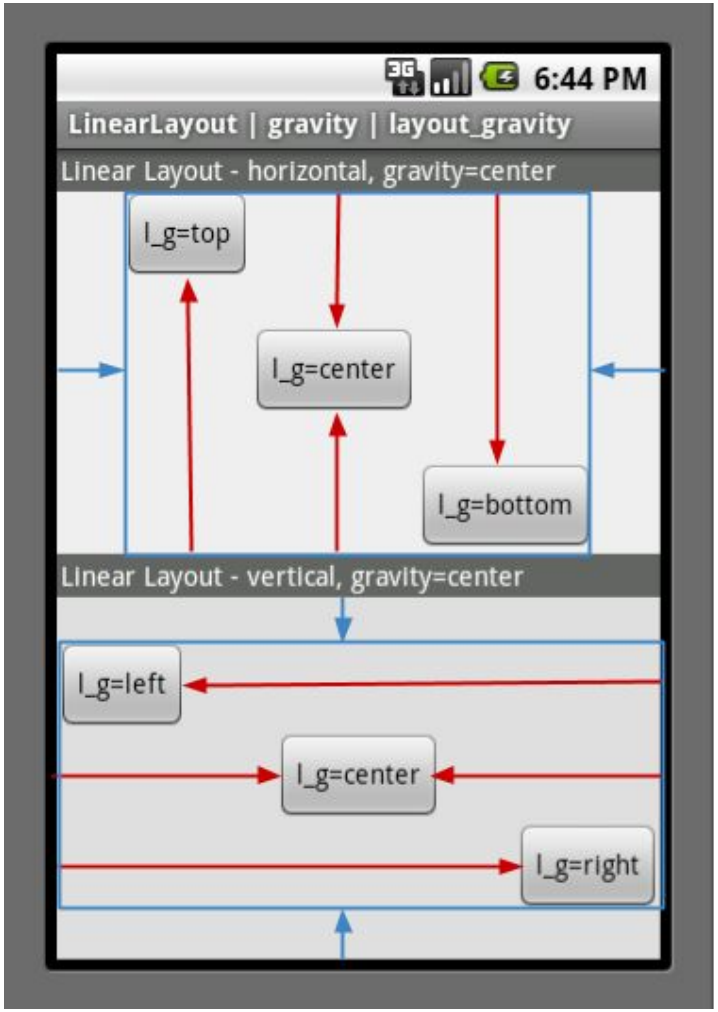
ConstraintLayout

- Пришел на замену RelativeLayout
- Также использует механизм зависимостей элементов
- Использует более эффективный алгоритм распределения элементов (Cossowary Algorithm)
- Плоская иерархия View
- Минимальный API 9




```
dependencies {
    implementation 'com.android.support.constraint:constraint-layout:2.0.0-alpha2'
}
```

Немного о гравитации

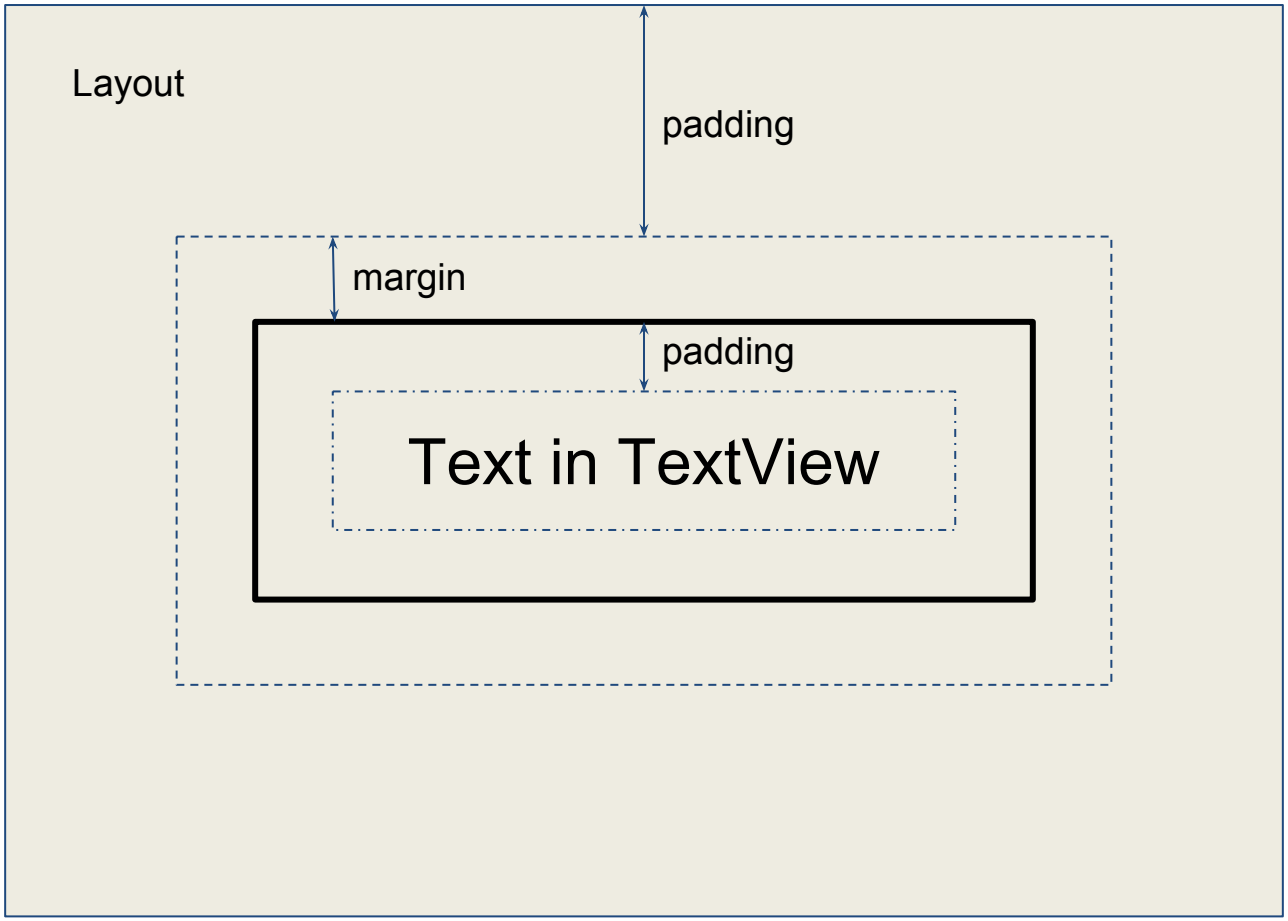


layout_gravity

gravity

- android:gravity – расположение контента внутри контейнера
- android:layout_gravity – расположение относительно родителя

И немного о отступах

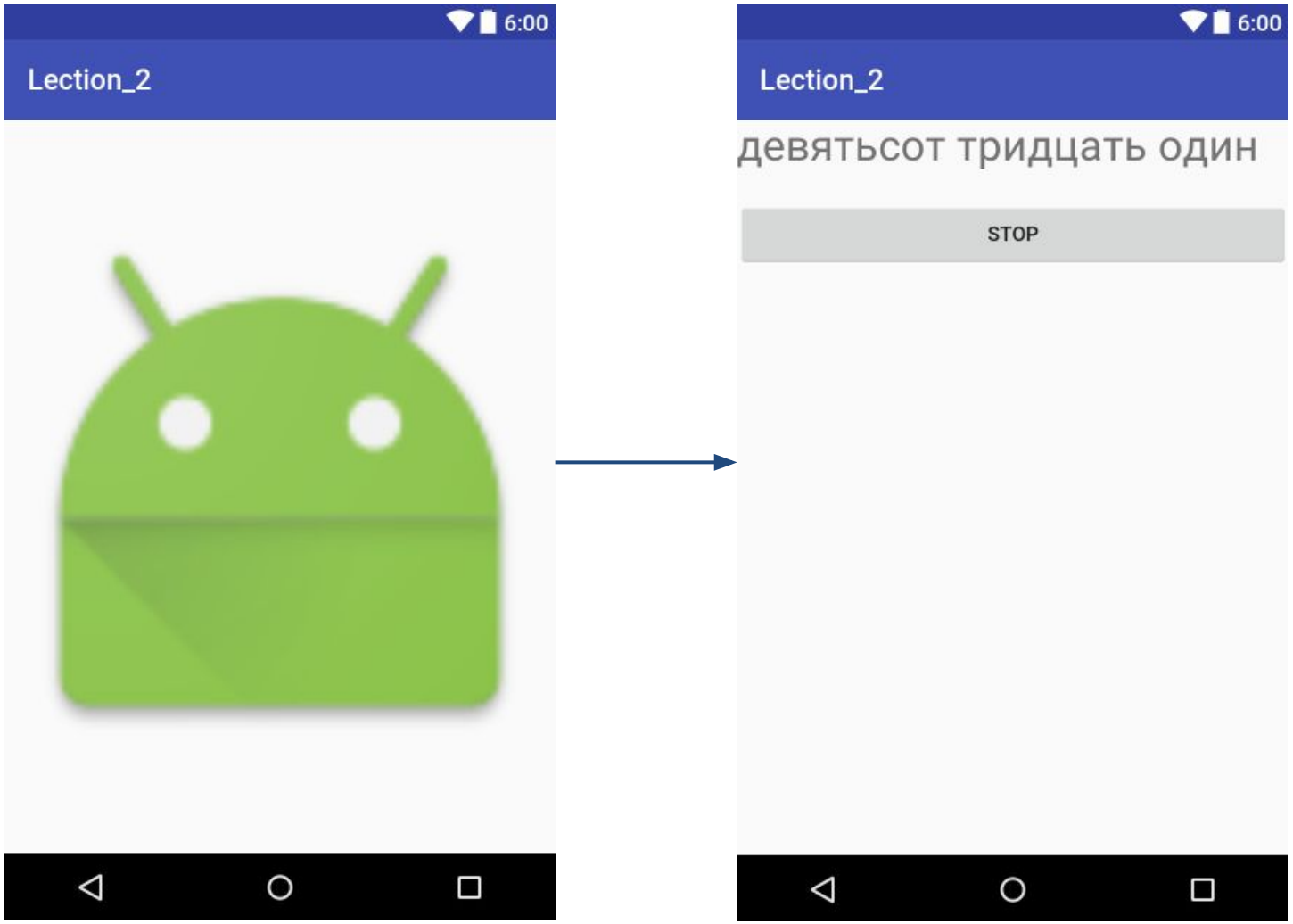


Попробуем создать тестовое приложение

- Сделаем тестовое приложение, в котором есть по каждому виду слоя и каждому элементу

- Можно взять готовое приложение
- https://github.com/ybereza/technotrack/2018-autumn/04/MainUI

Домашнее задание



Домашнее задание

- Запуск основной Activity из первой (Splash screen)
- Закрытие первой Activity (Splash screen)
- Корректная работа с таймером
- Корректная работа с поворотом экрана
- Управление состоянием
- Конвертация чисел в строковое представление

ТЕХНОТРЕК

Спасибо за внимание!

Юрий Береза – ybereza@gmail.com

Кирилл Филимонов - kirill.filimonov@gmail.com