

Активности (Activity)

Каждая активность — это экран (по аналогии с web-формой), который приложение может показывать пользователям. Чем сложнее создаваемое приложение, тем больше экранов (активностей) потребуется. При создании приложения потребуется, как минимум, начальный (главный) экран, который обеспечивает основу пользовательского интерфейса приложения. При необходимости этот интерфейс дополняется второстепенными активностями, предназначенными для ввода информации, ее вывода и предоставления дополнительных возможностей. Запуск (или возврат из) новой активности приводит к «перемещению» между экранами UI.

Большинство активностей проектируются таким образом, чтобы использовать все экранное пространство, но можно также создавать полупрозрачные или плавающие диалоговые окна.

Создание активности

Для создания новой активности наследуется класс AppCompatActivity или его подкласс (ListActivity, FragmentActivity и т. п.). Внутри реализации класса необходимо определить пользовательский интерфейс и реализовать требуемый функционал.

Базовый класс AppCompatActivity представляет собой пустой экран, который не особенно полезен, поэтому первое, что вам нужно сделать, это создать пользовательский интерфейс с помощью Представлений (View) и разметки (Layout).

Представления (View) — это элементы UI, которые отображают информацию и обеспечивают взаимодействие с пользователем. Android предоставляет несколько классов разметки (Layout), называемых также View Groups, которые могут содержать внутри себя несколько Представлений, для создания пользовательского интерфейса приложения.

Жизненный цикл активности

Приложения Android не могут контролировать свой жизненный цикл, ОС сама управляет всеми процессами и, как следствие, активностями внутри них. При этом, состояние активности помогает ОС определить приоритет родительского для этой активности Приложения (Application). А приоритет приложения влияет на то, с какой вероятности его работа (и работа дочерних активностей) будет прервана системой.

Стеки активностей

Состояние каждой активности определяется ее позицией в стеке (LIFO) активностей, запущенных в данный момент. При запуске новой активности представляемый ею экран помещается на вершину стека. Если пользователь нажимает кнопку «назад» или эта активность закрывается каким-то другим образом, на вершину стека перемещается (и становится активной) нижележащая активность.

На приоритет приложения влияет его самая приоритетная активность. Когда диспетчер памяти ОС решает, какую программу закрыть для освобождения ресурсов, он учитывает информацию о положении активности в стеке для определения приоритета приложения.

Состояния активностей

Активности могут находиться в одном из четырех возможных состояний.

Активное (Active). Активность находится на переднем плане (на вершине стека) и имеет возможность взаимодействовать с пользователем. Android будет пытаться сохранить ее работоспособность любой ценой, при необходимости прерывая работу других активностей, находящихся на более низких позициях в стеке для предоставления необходимых ресурсов. При выходе на передний план другой активности работа данной активности будет приостановлена или остановлена.

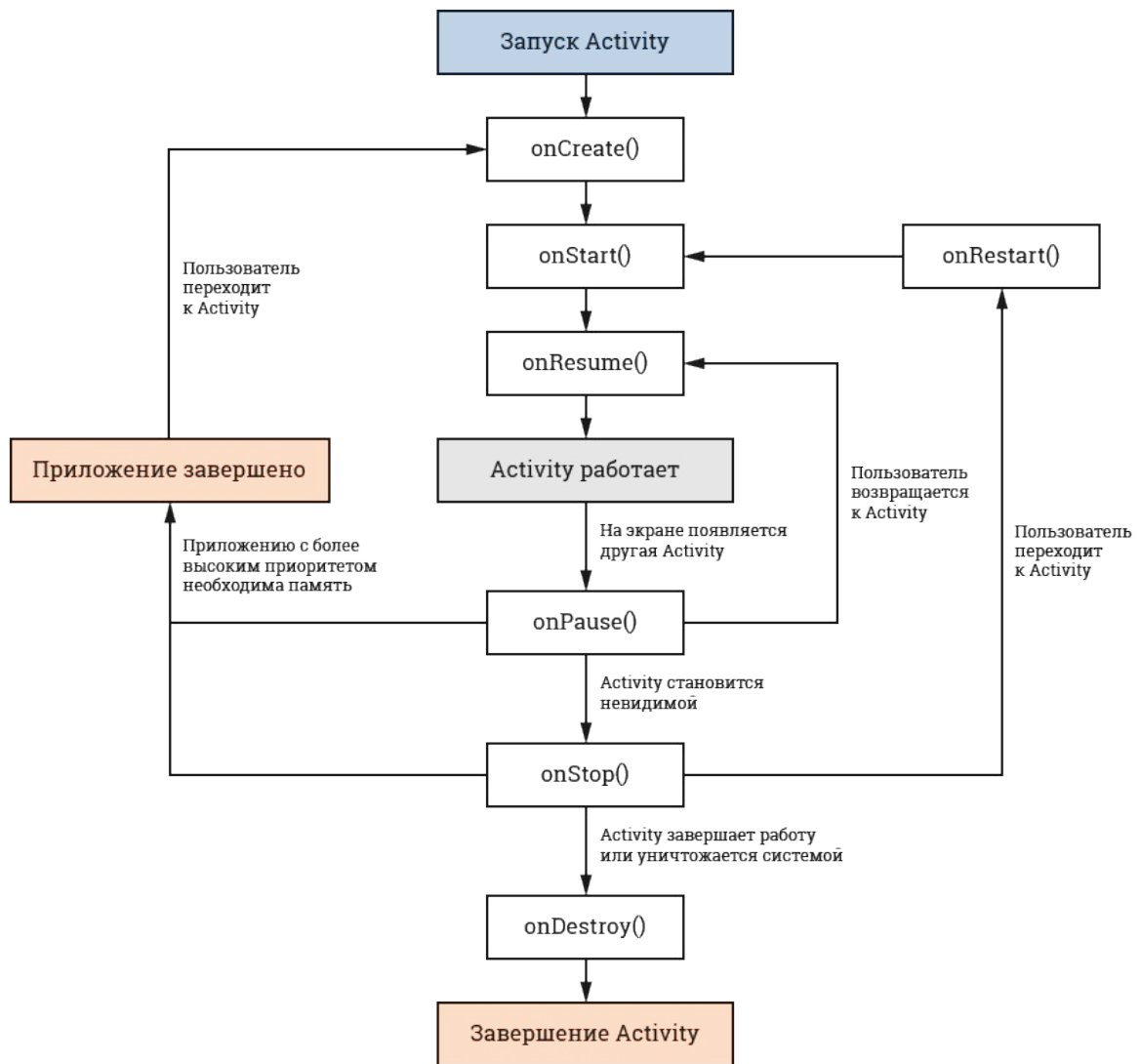
Приостановленное (Paused). Активность может быть видна на экране, но не может взаимодействовать с пользователем: в этот момент она приостановлена. Это случается, когда на переднем плане находятся полупрозрачные или плавающие (например, диалоговые) окна. Работа приостановленной активности может быть прекращена, если ОС необходимо выделить ресурсы активности переднего плана. Если активность полностью исчезает с экрана, она останавливается.

Остановленное (Stopped). Активность невидима, она находится в памяти, сохраняя информацию о своем состоянии. Такая активность становится кандидатом на преждевременное закрытие, если системе потребуется память для чего-то другого. При остановке активности разработчику важно сохранить данные и текущее состояние пользовательского интерфейса (состояние полей ввода, позицию курсора и т. д.). Если активность завершает свою работу или закрывается, он становится неактивным.

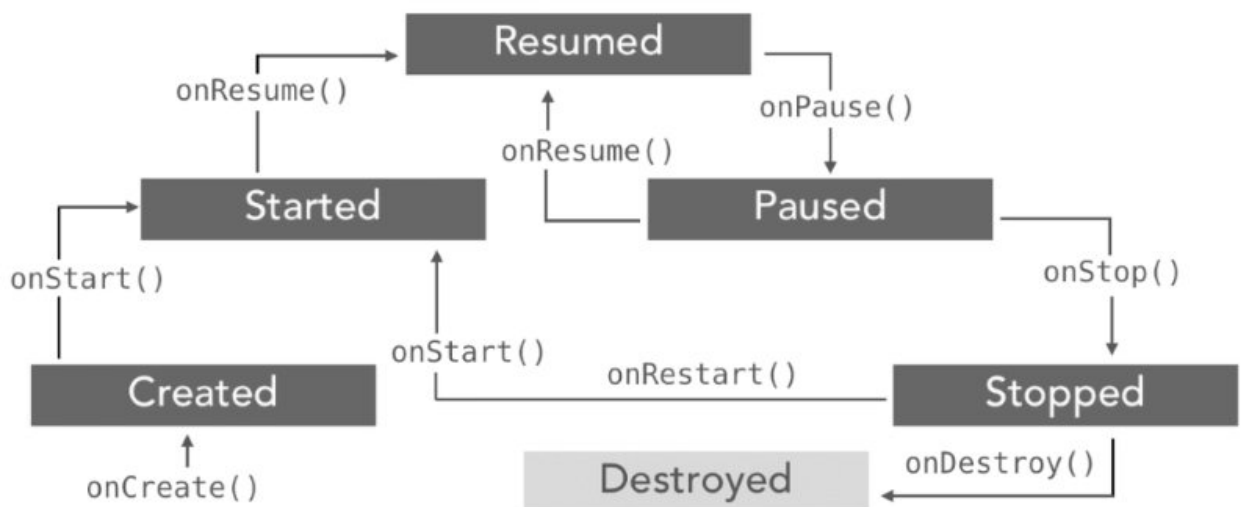
Неактивное (Inactive). Когда работа активности завершена, и перед тем, как она будет запущена, данная активности находится в неактивном состоянии. Такие активности удаляются из стека и должны быть (пере)запущены, чтобы их можно было использовать.

Изменение состояния приложения — недетерминированный процесс и управляется исключительно менеджером памяти Android. При необходимости Android вначале закрывает приложения, содержащие неактивные активности, затем остановленные и, в крайнем случае, приостановленные.

Для обеспечения полноценного интерфейса приложения, изменения его состояния должны быть незаметными для пользователя. Меняя свое состояние с приостановленного на остановленное или с неактивного на активное, активность не должна внешне меняться. При остановке или приостановке работы активности разработчик приложения должен обеспечить сохранение состояния активности, чтобы его можно было восстановить при выходе активности на передний план. Для этого в классе `AppCompatActivity` имеются обработчики событий, переопределение которых позволяет разработчику отслеживать изменение состояний активности.



В целом переход между состояниями activity можно выразить следующей схемой:



Введение в логирование

Основные уровни логирования

1. VERBOSE - наиболее подробная информация (для разработки)
2. DEBUG - отладочная информация
3. INFO - общая информация о работе приложения
4. WARN - предупреждения о потенциальных проблемах
5. ERROR - ошибки, которые не crash'ат приложение
6. ASSERT - критические ошибки (Что-то пошло очень не так)

Android предоставляет класс `android.util.Log` для базового логирования:

```
package com.walerider.myapplication;

import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

public class MainActivity extends AppCompatActivity {
    TextView textView;
    Button button;
    EditText editText;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.v("TAG", "Verbose message"); // VERBOSE
        Log.d("TAG", "Debug message");   // DEBUG
        Log.i("TAG", "Info message");     // INFO
        Log.w("TAG", "Warning message");  // WARN
        Log.e("TAG", "Error message");    // ERROR
    }
}
```

Разберём.

Log – статичный класс, который можно вызвать из любой точки приложения.

.v; .e; .d; .i; .w; его методы, которые требуют в поставить в себя:
(«Тег лога», «Сообщение лога»)

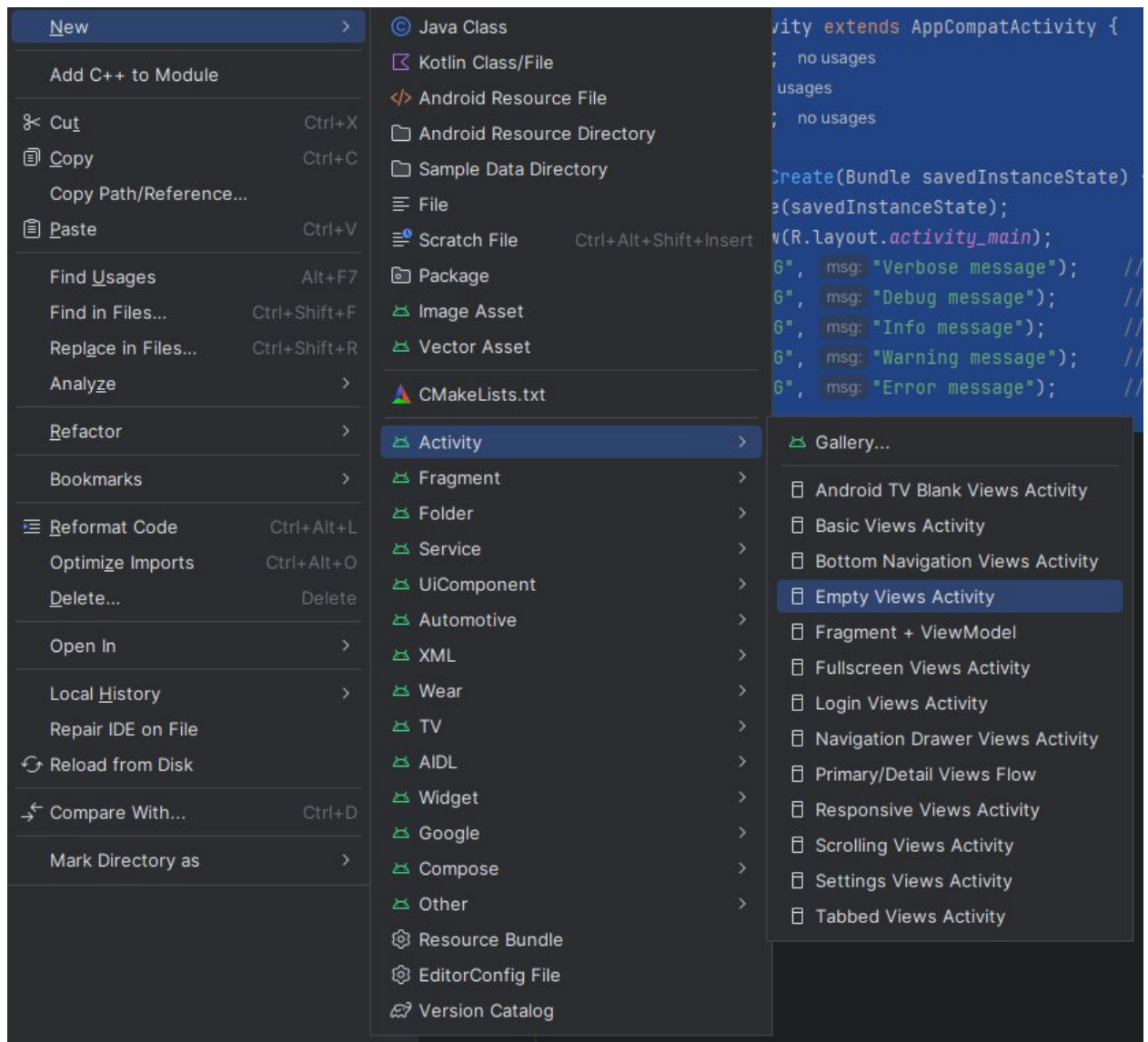
Задание 1

Сделайте так, чтобы ваше приложение выводило логи, информирующие о каждой стадии его работы.

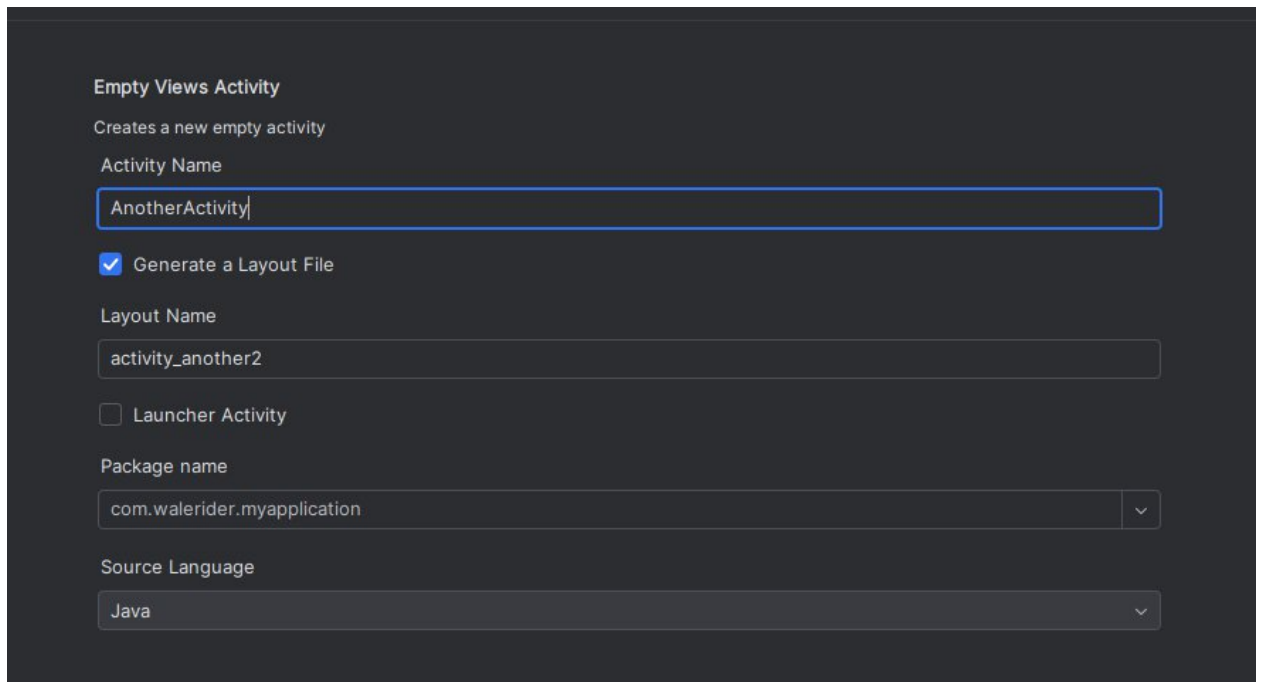
Введение в Intent. Запуск Activity

Для взаимодействия между различными объектами activity ключевым классом является **android.content.Intent**. Он представляет собой задачу, которую надо выполнить приложению.

Для работы с Intent добавим новый класс Activity. Для этого нажмем правой кнопкой мыши в папку, в которой находится класс MainActivity, и затем в контекстном меню выберем **New->Activity->Empty Views Activity**.



При создании можете назвать вашу новую Activity хоть как. У меня к примеру AnotherActivity:



Empty Views Activity

Creates a new empty activity

Activity Name

AnotherActivity

☒ Generate a Layout File

Layout Name

activity_another2

☐ Launcher Activity

Package name

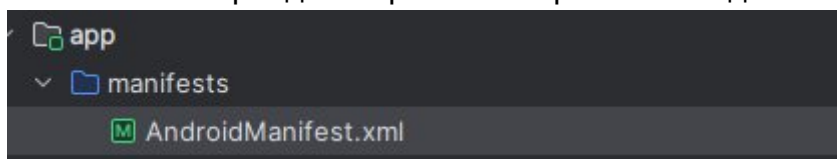
com.walerider.myapplication

Source Language

Java

И после этого в проект будет добавлена новая Activity.

После этого перейдём в файл манифеста. Находится он здесь:



И увидим там следующие строчки:

```
<activity
    android:name=".AnotherActivity"
    android:exported="false" />
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Все используемые классы activity должны быть описаны в файле

AndroidManifest.xml с помощью элемента `<activity>`. Каждый подобный элемент содержит как минимум один атрибут `android:name`, который устанавливает имя класса activity.

Однако по сути activity - это стандартные классы java, которые наследуются от класса **Activity** или его наследников. Поэтому вместо встроенных шаблонов в Android Studio можем добавлять обычные классы, и затем их наследовать от класса Activity. Однако в этом случае нужно будет вручную добавлять в файл манифеста данные об activity.

Причем для MainActivity в элементе **intent-filter** определяется интент-фильтр. В нем элемент action значение "android.intent.action.MAIN" представляет главную точку входа в приложение. То есть MainActivity остается основной и запускается приложением по умолчанию.

Для AnotherActivity просто указано, что она в проекте, и никаких intent-фильтров для нее не задано.

Чтобы из MainActivity запустить AnotherActivity, надо вызвать метод `startActivity()`:

```
Intent intent = new Intent(this, AnotherActivity.class);
startActivity(intent);
```

В качестве параметра в метод `startActivity` передается объект `Intent`. Для своего создания `Intent` в конструкторе принимает два параметра: контекст выполнения (в данном случае это текущий объект `MainActivity`) и класс, который используется объектом `Intent` и представляет передаваемые в задачу данные (фактически класс `activity`, которую мы будем запускать).

Передача данных между Activity.

Для передачи данных между двумя Activity используется объект **Intent**. Через его метод **putExtra()** можно добавить ключ и связанное с ним значение.

Например, передача из текущей activity в AnotherActivity строки "Do u need a name?" с ключом "name":

```
Intent intent = new Intent(this, AnotherActivity.class);
intent.putExtra("name", "Do u need a name?");
startActivity(intent);
```

Для передачи данных применяется метод **putExtra()**, который в качестве значения позволяет передать данные простейших типов - `String`, `int`, `float`, `double`, `long`, `short`, `byte`, `char`, массивы этих типов, либо объект интерфейса `Serializable`.

Чтобы получить отправленные данные при загрузке AnotherActivity, можно воспользоваться методом **get()**, в который передается ключ объекта:

```
Bundle arguments = getIntent().getExtras();  
String name = arguments.get("name").toString();
```

В зависимости от типа отправляемых данных при их получении мы можем использовать ряд методов объекта Bundle. Все они в качестве параметра принимают ключ объекта. Основные из них:

- **get()**: универсальный метод, который возвращает значение типа Object. Соответственно поле получения данное значение необходимо преобразовать к нужному типу
- **getString()**: возвращает объект типа String
- **getInt()**: возвращает значение типа int
- **getByte()**: возвращает значение типа byte
- **getChar()**: возвращает значение типа char
- **getShort()**: возвращает значение типа short
- **getLong()**: возвращает значение типа long
- **getFloat()**: возвращает значение типа float
- **getDouble()**: возвращает значение типа double
- **getBoolean()**: возвращает значение типа boolean
- **getCharArray()**: возвращает массив объектов char
- **getIntArray()**: возвращает массив объектов int
- **getFloatArray()**: возвращает массив объектов float
- **getSerializable()**: возвращает объект интерфейса Serializable

Задание 2.

Сделайте, чтобы при сворачивании приложения запускалась вторая Activity и считала прошедшее в свёрнутом состоянии время. Используйте метод `System.currentTimeMillis()`. Переведите это в секунды.

