

АНГЕЛИН ЛАЛЕВ

**МОБИЛНО ПРОГРАМИРАНЕ С
JAVA**

/ЛЕКЦИОНЕН КУРС/

2019

Мобилно програмиране с JAVA /лекционен курс/

Онлайн издание.

Copyright© Ангелин Лалев; Автор: Ангелин Лалев

ISBN:

Настоящата книга е лицензирана под Creative Commons CC BY-NC-ND 4.0

лиценз. Пълният текст на лиценза се намира на следния адрес:

<https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode>

СЪДЪРЖАНИЕ НА ГЛАВА 3

Глава 3. ДЕЙНОСТИ (ACTIVITIES)	5
1. Основни концепции при работа с дейности.....	5
1.1. Жизнен цикъл на дейностите	6
1.2. Задачи и back-стек.....	11
2. Създаване на дейност	13
3. Примерен проект.....	21
3.1. Създаване на скелет на приложението	21
3.2. Модифициране на layout файла.....	24
3.3. Модификация на програмния код.....	26

ГЛАВА 3. ДЕЙНОСТИ (ACTIVITIES)

1. Основни концепции при работа с дейности

Дейностите (Activities) в Андроид са основният начин, по който потребителската програма взаимодейства с потребителя. Те съдържат визуални елементи за комуникация с потребителя и могат да бъдат оприличени на диалоговите прозорци при нормалните десктоп системи.

За разлика от повечето десктоп системи, при които обикновено работата следва определена последователност (например стартиране на приложението, главно меню, документ, диалогов прозорец, диалогов подпрозорец), ограниченията на потребителския интерфейс на мобилните устройства налагат оптимизация на процеса по стартиране на приложения. Андроид, повече от традиционните десктоп системи, разглежда приложенията като набор от модули (наричани точно „дейности“), които могат да се стартират независимо една от друга.

Така например, при стартирането на клиент за електронна поща, той може да бъде стартиран с дейността, която реализира главното меню на програмата или пък с дейността, която показва получените нови съобщения или с дейността, която реализира писане на ново съобщение.

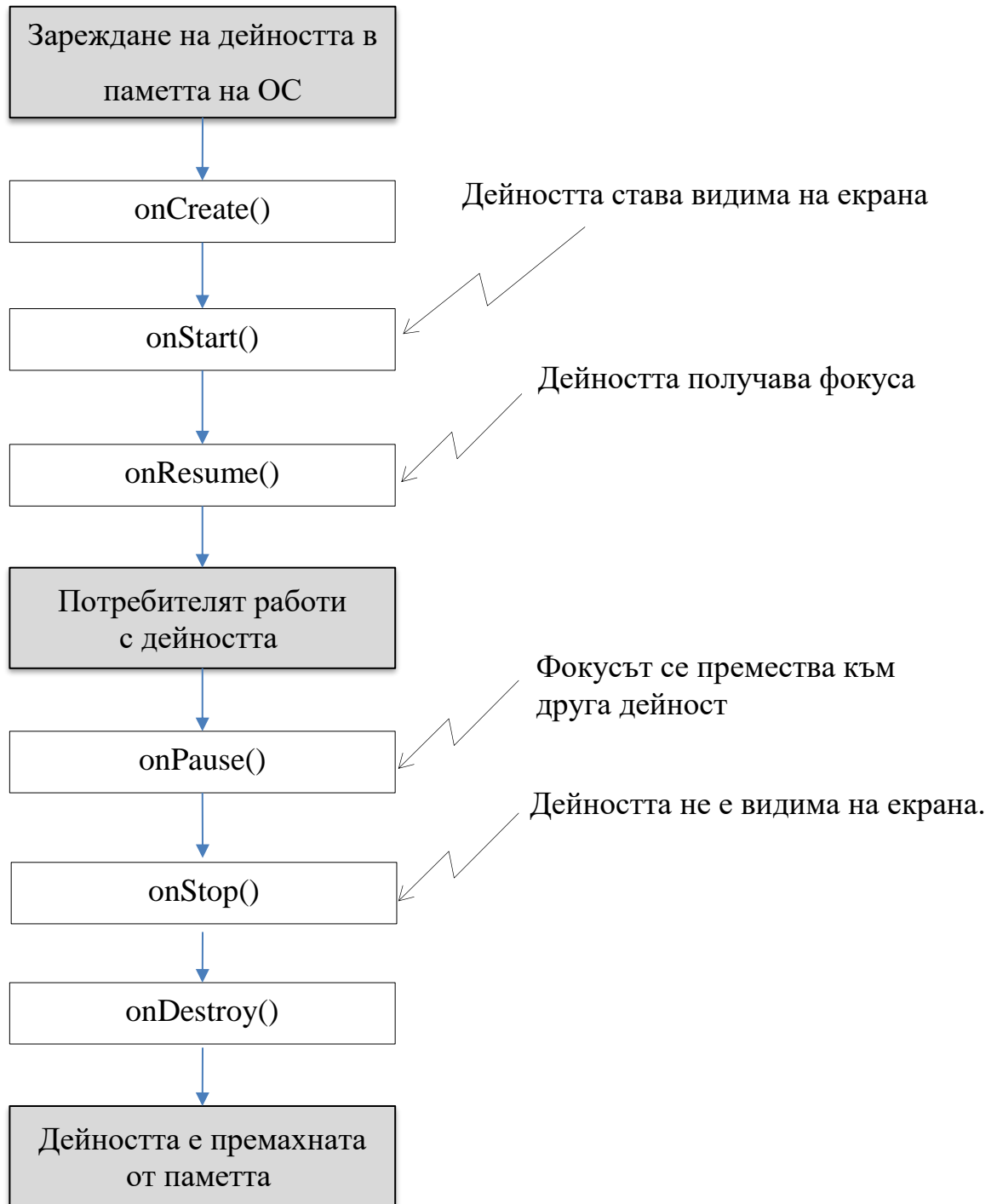
Дейностите в едно приложение могат да бъдат стартирани директно от потребителя или пък могат да бъдат извиквани от други приложения. Така например, ако потребител на приложението Photos избере да сподели снимка чрез електронната поща, Photos ще стартира директно дейността по създаване на ново съобщение от клиента за електронна поща на системата.

Една от дейностите в дадено мобилно приложение за Андроид обикновено се обозначава като *главна*. Главната дейност се стартира когато потребителят ръчно активира приложението.

В мобилното приложение, дейностите се представляват от класове, наследници на класа *Activity*. Програмистите могат да изберат да наследят директно от *Activity*, или както е много по-често срещаният начин, да наследят някой от наследниците на *Activity*. Такива могат да бъдат например *FragmentActivity* или *AppCompatActivity*. *AppCompatActivity* например гарантира, че дейността ще работи и на по-стари устройства, показвайки лентата на приложението (*AppBar*), която е приетата за стандарт в модерните приложения на Android.

1.1. Жизнен цикъл на дейностите

Жизненият цикъл на дейностите преминава през различни състояния. Операционната система уведомява дейността за промени в състоянието като извиква специални колбак (callback) методи. В приложенията за Андроид, колбак методите се реализират като наследници на методи, предлагани от класа *Activity*.



а/ Методи *OnCreate* и *OnDestroy*

Ако програмистът дефинира *OnCreate* метод в своята програма (това е на практика задължително) този метод ще бъде извикан от операционната система веднага щом тя зареди в оперативната памет дейността. Отговорност на този метод е да извърши инициализацията на приложението. Например тук обикновено се създават изгледите, които ще бъдат поставени в прозореца на дейността.

Извикването на *OnCreate* още не означава, че дейността започва изпълнението си или че нейният прозорец е видим на екрана на потребителя. След извикването на *OnCreate* и преди дейността да стане видима и да получи фокус, Android ще извика още два колбач метода – *OnStart* и *OnResume*.

OnDestroy се извиква от операционната система, когато дейността се премахва от паметта на компютъра. Програмистите наследяват този метод, когато трябва да поставят в него код за окончателно освобождаване на заетите ресурси. Преди извикването на *OnDestroy*, дейност, която се вижда на екрана потребителя ще получи още две уведомления чрез колбач методи – *OnPause* и *OnStop*. Според това, какви ресурси са нужни, за да работи дейността, програмистът може да реши да освободи тези ресурси в *OnStop* или *OnPause*. Така например, камерите на телефона са ресурс, който не трябва да бъде задържан от една дейност, когато тя не е видима на екрана. Те би следвало да се освобождават далеч преди операционната система да извика *OnDestroy*.

Важно е да се отбележи, че дейност, която не е видима на екрана, не винаги се унищожава веднага след като се спре и стане невидима за потребителя, тъй като потребителят може да се върне към нея. Тя просто се запазва в паметта за определено време, което зависи от нуждите на системата от свободна памет.

Ако възникне нужда от повече памет, операционната система ще премахне някои спрени дейности от паметта. При това Андроид ще опита да

извика *OnDestroy* метода на засегнатите дейности. Ако обаче нуждата от повече памет възникне вследствие на заявка от страна на процес с висок приоритет (например системен процес), за да се избегне „забиването“ и забавянето на цялото мобилно устройство, системата ще убие¹ процеса на дейността, като при това *OnDestroy* няма да бъде извикан.

Ето защо записването на текущото състояние на дейността (например актуализиране на записи в БД) трябва да се прави в по-предно извикваните колбач методи като *OnPause*. *OnDestroy* от своя страна може да се използва за чисто затваряне на сокети, файлове и пр. ресурси, които ще бъдат затворени (макар и не толкова чисто) и при убиването на процеса.

б/ методи *OnStart*, *OnStop* и *OnRestart*

Методът *OnStart* се извиква, когато дадено приложение стане видимо на екрана. В повечето случаи, това означава, че то автоматично ще получи фокус и веднага след *OnStart*, Android ще извика и *OnResume*. Версиите на Android от 7.0 нататък обаче, поддържат режим на работа в множество прозорци едновременно. Само една от дейностите, които са показани в такива прозорци, ще има фокуса, което означава, че потребителят ще взаимодейства директно с нея (така например входът от клавиатурата ще отива в полетата на тази дейност).

Методът *OnStop* на една дейност се извиква тогава, когато дейността не е видима повече на екрана.

Методите *OnStart* и *OnStop* могат да се извикват многократно, тъй като, както споменахме, Android не винаги премахва дейността от паметта. Това означава, че програмистите внимателно трябва да преценят къде да поставят

¹ Убиването“ всъщност е професионален термин за насилствено прекратяване на процеса на дадена програма и носи това име по традиция от операционните системи Линукс и Unix.

кода за инициализация на кода на приложението. Еднократни дейности, като например създаването на изгледи на дейността, вероятно трябва да се поставят в *OnCreate*.

Методът *OnRestart* се извиква, когато потребителят отново активира дадена дейност. Програмистите поставят в *OnRestart* код, който ще се активира само при *повторно* активиране на дейност, която вече е била активна, но след това е спряна. Такъв сценарий например е натискането на бутона „Назад”/”Back” на телефона, което активира предходната дейност.

Важно е да се отбележи, че при подобна ситуация, системата ще извика първо *OnRestart*, след което *OnStart* и *OnResume*.

в/ методи *OnResume* и *OnPause*

Методите *OnResume* и *OnPause* се извикват тогава, когато операционната система съответно предаде или отнеме фокуса на текущата дейност. В повечето случаи, операционната система ще показва само един прозорец в даден момент, но когато се работи в режим на множество прозорци, както бе споменато по-горе, *OnPause* ще означава, че потребителят е преминал към друг прозорец.

Жизненият цикъл на дейностите има редица интересни особености. Така например когато потребителят промени конфигурацията, като преоразмери прозореца на дейността (многопрозоречен режим) или обърне устройството от портретна (Portrait) към пейзажна (Landscape) ориентация, Android първо ще премахне изцяло дейността от паметта, извиквайки *OnPause*, *OnStop* и *OnDestroy*, след което ще стартира наново дейността, извиквайки *OnCreate*, *OnStart* и *OnResume*. Също така, когато потребителят натисне бутона *Back* на дадена дейност, тя ще бъде задължително премахната от паметта с последователното извикване на *OnPause*, *OnStop* и *OnDestroy*, тъй като се допуска, че потребителят не се интересува от успешното ѝ завършване.

1.2. Задачи и back-стек

Android групира дейностите в задачи (tasks). Стартирането на нова задача започва с щракване върху икона на дадено приложение в домашния екран на мобилното устройство. Това стартира главната дейност на това приложение, която на свой ред стартира следваща дейност от същото или друго приложение. Именно съвкупността от тези дейности се нарича „задача“.

Потребителят може да спре изпълнението на една задача и да стартира друга като натисне бутона Home на мобилното си устройство. Еднократното натискане ще покаже домашния екран, от който потребителят евентуално ще стартира нова дейност от ново приложение, която ще формира нова задача. Продължителното натискане ще позволи потребителя да активира една от задачите, работили по-рано на устройството.

Важно е да се отбележи, че всяка задача помни последователността на дейностите, които са се изпълнявали в рамките на задачата. Този списък е известен като *back-стек*. Името е свързано с бутона “Back” на мобилното устройство. Натискането на този бутон води до унищожаване на текущата активна дейност и активирането на предната дейност, запомнена в този списък.

Редът на дейностите в back-стека не може да се променя. Това значи, че ако дадена дейност е стартирана няколко пъти в последователността на задачата, то в back-стекът ще бъдат направени няколко копия на дейността.

Полезно е да си представим как работят задачите и back-стека с помощта на малък казус, който ще помогне да изясним и отношенията между дискутираните до момента понятия.

Така например, потребителят може да стартира приложение за електронна поща от домашния екран и по този начин да създаде нова задача.

Да приемем, че главната дейност на това приложение показва адресна книга. Евентуално потребителят избира потребител `lalev@uni-svishtov.bg` и натиска бутон за изпращане на ново съобщение, който стартира нова дейност - съставяне на ново съобщение за електронна поща.

Към момента в бек-стека има две дейности. Първата е адресната книга, а на върха на стека е активната в момента дейност – съставяне на съобщение. След като е написал текста на съобщението, потребителят може да си припомни, че трябва да изпрати съобщението до още хора. За целта той ще натисне бутона за адресна книга, разположен до СС полето на съобщението за ел. поща. Това ще стартира ново копие на дейността адресна книга и в back-стека ще има три дейности – „адресна книга“, следвана от „ново съобщение“, следвана от „адресна книга“, която е на върха на стека и е видима за потребителя.

До момента, всички дейности в задачата са от едно приложение – клиентът за ел. поща. Възможно е обаче, на даден етап потребителят да реши да извика „Галерия“ и да прикачи снимка към новото съобщение. Дейността „Галерия“ е от друго приложение, но ще бъде стартирана и ще бъде поставена на върха на back-стека на текущата задача. Сега в back-стека ще има дейности от две приложения.

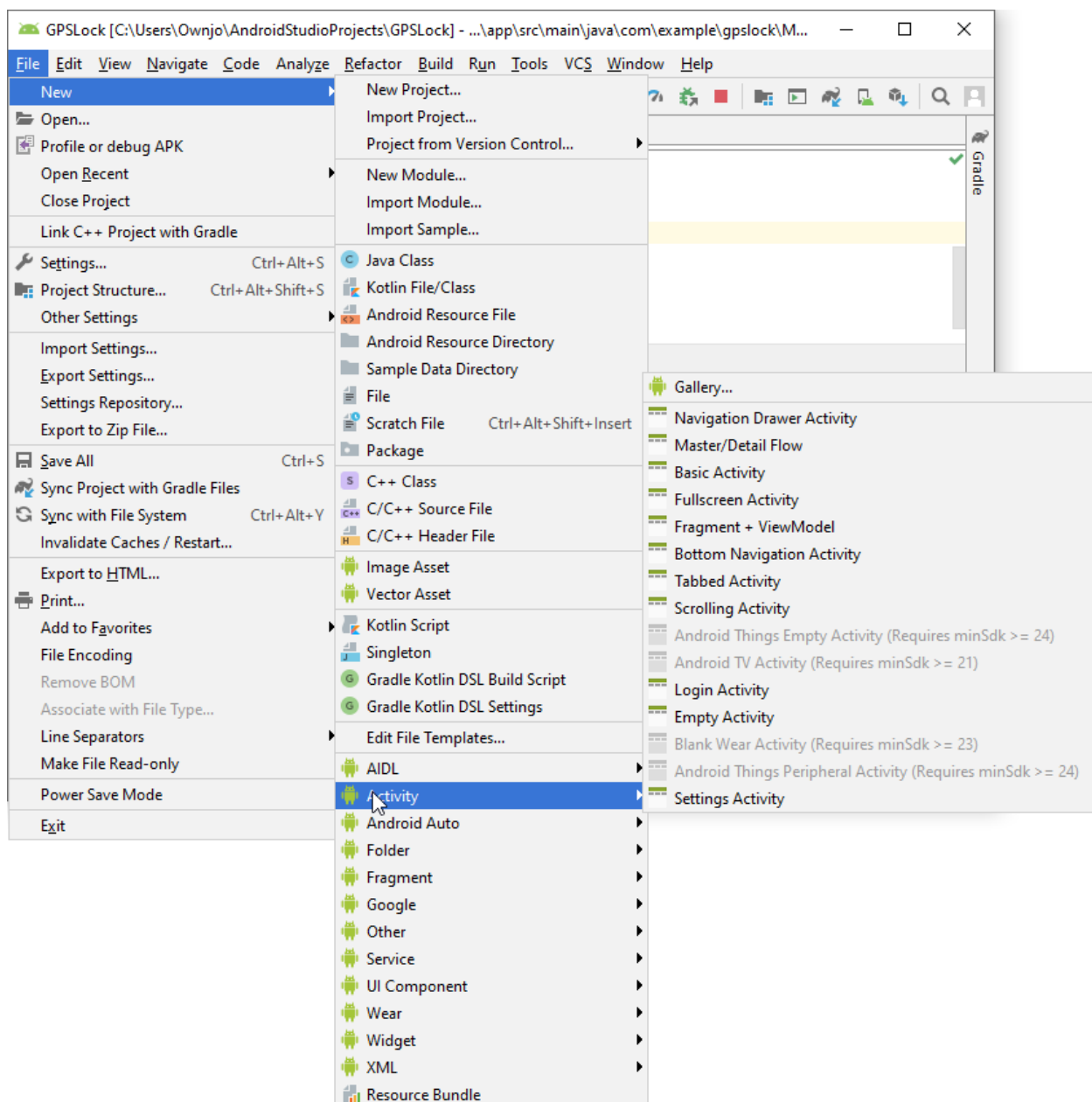
Докато извършва този процес, потребителят може да реши да си почине, като изгледа любимо видео на Youtube. Той може да натисне Home бутона на телефона и да стартира приложението Youtube от домашния екран. Това ще стартира нова задача със собствен back-стек. Ако сега потребителят задържи Home бутона, той ще може да избере от списъка на предходните задачи клиента за ел. поща и ще се върне към писането на съобщението.

Важен факт, който трябва да се спомене за задачите, е че ако потребителят дълго време не изпълнява дадена задача, операционната система евентуално ще изчисти всички дейности от back-стека, освен тази, която е била първа за задачата. Това се прави, защото след такъв период от време, потребителят най-вероятно е изоставил операциите, поддържани от стартираните дейности.

Ако това поведение е неприемливо за програмата, тя може да сигнализира това чрез задаване на специални атрибути на *Activity* елемента в манифеста на приложението. Ролята на манифеста и *Activity* елемента е дискутирана в по-голям детайл в следващата точка.

2. Създаване на дейност

Създаването на дейности в проект на Android Studio може да се извърши по няколко начина. Така например, от главното меню програмистът може да избере *File/New/Activity* (Вж. фиг. 3.1).



Фиг. 3.1. Създаване на нова дейност в Android Studio

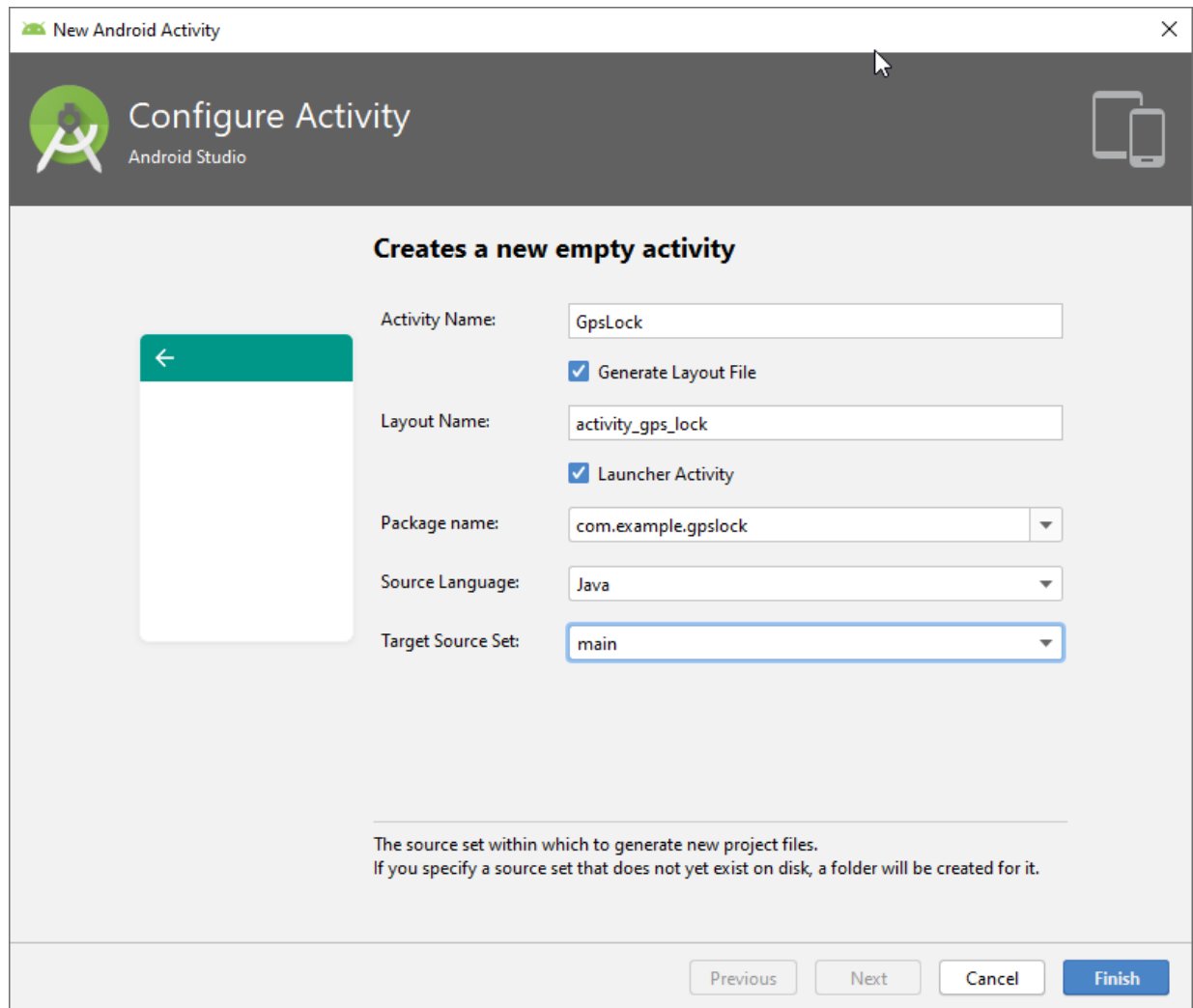
Подменюто, което следва *Activity*, съдържа елементи за различни готови шаблони, които могат да бъдат доработени и разширени от програмиста. По-важните от тях са:

Empty Activity	 <p>Празна дейност, която няма предварително добавени допълнителни визуални елементи.</p>
Basic Activity	 <p>Дейност, която има добавена лента на приложението (AppBar) и плуващ бутон за действие (Floating Action Button).</p>
Master/Detail Flow	 <p>Дейност, която показва списък с елементи и списък с детайли за елемент. При избор на елемент от единия списък, детайлите за избрания елемент се показват в</p>

	<p>другия. Според вида на екрана на устройството, разположението на визуалните елементи може да варира.</p>
<p>Bottom Navigation Activity</p>	 <p>Дейност с долна лента за навигация, която позволява превключване между няколко изгледа.</p>
<p>Login Activity</p>	 <p>Дейност, която съдържа визуалните елементи, необходими за удостоверяване и оторизация с име и парола.</p>

След като програмистът избере вид на създаваната дейност, *Android Studio* ще изведе диалогов прозорец за конфигуриране на дейността. Конфигурирането касае задаване на параметри като име на клас за дейността, име на лейаут файл/файлове и пр. Параметрите варират според избраната дейност.

Диалоговият прозорец за конфигуриране на празна дейност (*Empty Activity*) е показан по-долу:



Фиг. 3.2. Конфигуриране на празна дейност в Android Studio

В примера по-горе, програмистът създава дейност с име *GpsLock*. Името на дейността става и име на класа, който ще представлява дейността в програмния код. Ето защо в това име не може да се оставят интервали и е добра идея то да спазва стандартните конвенции за имена на класове.

Отметката *Generate Layout File* означава, че Android Studio ще генерира лейаут (layout) файл едновременно със създаване на дейността и ще включи код в метода *onCreate* на дейността, който ще зареди този файл при създаването на дейността от операционната система. Лейаут файлът съдържа XML инструкции за изчертаване на графичния потребителски интерфейс на

дейността и може да се редактира директно от визуалния редактор на Android Studio.

При повечето реални приложения, създаваната дейност ще бъде създадена с поне един лейаут файл. Рядко могат да бъдат срещнати и такива приложения, които могат да изберат да създадат лейаут на дейността си динамично, т.е. директно с програмен код без да зареждат XML файл.

Полето *Layout Name* указва името на лейаут файла. Когато *Generate Layout File* е отметната, Android Studio ще генерира предложение за името на този файл на база името на дейността, въведена от програмиста.

Ако отметката *Launcher Activity* е отбелязана, дейността ще бъде отбелязана в манифеста на приложението по специален начин. Като резултат от това отбелязване, стартиращата програма на Андроид ще покаже икона за дейността в домашния екран на мобилното устройство, където се намират иконите за стартиране на приложенията.

Package Name съдържа името на Java пакета, който ще съдържа класа за създаваната дейност. Ако при инсталирането на Android студио, то е инсталирано с поддръжка за Kotlin, в полето *Language*, потребителят ще може да избира между Java и Kotlin за програмен език, на който да бъде програмиран класът на дейността.

Когато потребителят е задал всички нужни параметри, той следва да натисне бутона *Finish*. При създаването на дейността, Android Studio ще нанесе три важни промени в дървото с кода и обслужващите файлове.

1. Ще бъде създаден нов клас за дейността
2. Евентуално ще бъдат създадени един или повече лейаут файлове, които позволяват на потребителя да редактира предварително потребителският интерфейс на дейността в графичния редактор на Android Studio.

3. Дейността ще бъде описана в манифеста на приложението, *AndroidManifest.xml*, като за целта ще бъде добавен XML елемент *Activity*. Информацията в този елемент информира операционната система за различни изисквания и ограничения на дейността.

По-долният пример показва съдържанието на създадените файлове при създаване на Empty Activity.

GpsLock.java

```
1 package com.example.gpslock;
2
3 import androidx.appcompat.app.AppCompatActivity;
4 import android.os.Bundle;
5
6 public class GpsLock extends AppCompatActivity {
7     @Override
8     protected void onCreate(Bundle savedInstanceState) {
9         super.onCreate(savedInstanceState);
10        setContentView(R.layout.activity_gps_lock);
11    }
12 }
```

AndroidManifest.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.example.gpslock">
4
5     <application
6         android:allowBackup="true"
7         android:icon="@mipmap/ic_launcher"
8         android:label="@string/app_name"
9         android:roundIcon="@mipmap/ic_launcher_round"
10        android:supportsRtl="true"
11        android:theme="@style/AppTheme">
12        <activity android:name=".GpsLock">
13            <intent-filter>
14                <action android:name="android.intent.action.MAIN" />
15
16                <category android:name="android.intent.category.LAUNCHER" />
17            </intent-filter>
18        </activity>
19    </application>
20
21 </manifest>
```

activity_gps_lock.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools">
```

```
6    android:layout_width="match_parent"  
7    android:layout_height="match_parent"  
8    tools:context=".GpsLock">  
9  
10 </androidx.constraintlayout.widget.ConstraintLayout>
```

GpsLock.java съдържа изцяло нов клас, генериран от Android Studio при създаването на дейността. Както е видно от ред 6, *GpsLock* наследява *AppCompatActivity*. Както бе споменато по-рано, *AppCompatActivity* показва лента на приложението и по-модерни елементи в изгледа на потребителския интерфейс на стари версии на Андройд, които нямат подобна функционалност, предоставена от операционната система.

Редове 12-18 в *AndroidManifest.xml* представляват Activity елемент, който е генериран от Android Studio, за да опише дейността. Потребител на тази информация е операционната система, която ще стартира готовото приложение.

Последният файл – *activity_gps_lock.xml* – е лейаут файл, съдържащ описание на визуалните елементи, които ще бъдат заредени от дейността при нейното създаване. Нормално Android Studio редактира този файл с помощта на визуален редактор и не е необходимо той да се редактира ръчно.

Класът *GpsLock* реализира един от дискутираните по-рано в главата колбак методи. Това е методът *OnCreate*. Читателите следва да забележат, че този метод се наследява от родителския клас и следователно е дефиниран с помощта на *@Override* директива (ред 7 на *GpsLock.java*). Също така е важно да се забележи, че *OnCreate* извиква метода на класа-родител (ред 9 на *GpsLock.java*). *OnCreate*, както всички дискутирани до момента колбак методи, получава параметър от тип *Bundle*. Той представлява начин за съхраняване на състоянието на инстанцията, който ще бъде разгледан по-нататък.

Най-важният ред в метода *OnCreate* обаче е ред 10. Методът *setContentView* зарежда лейаут файла *activity_gps_lock.xml* в дейността. Класът

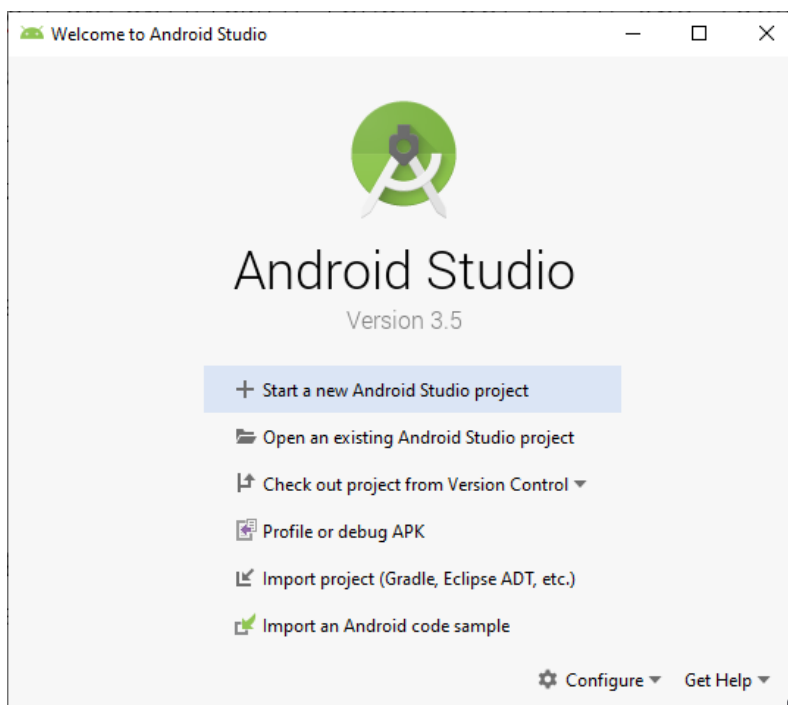
R е динамично генериран при компилацията на проекта клас, който съдържа цифровите идентификатори на всеки ресурс, използван от програмата. Лейаут файлът се третира като ресурс и при компилация получава уникален номер. Именно този номер е стойността на константата *R.layout.activity_gps_lock*.

3. Примерен проект

Примерният проект към тази тема е елементарно приложение с една дейност, която регистрира кога системата извиква разгледаните в главата колбак методи.

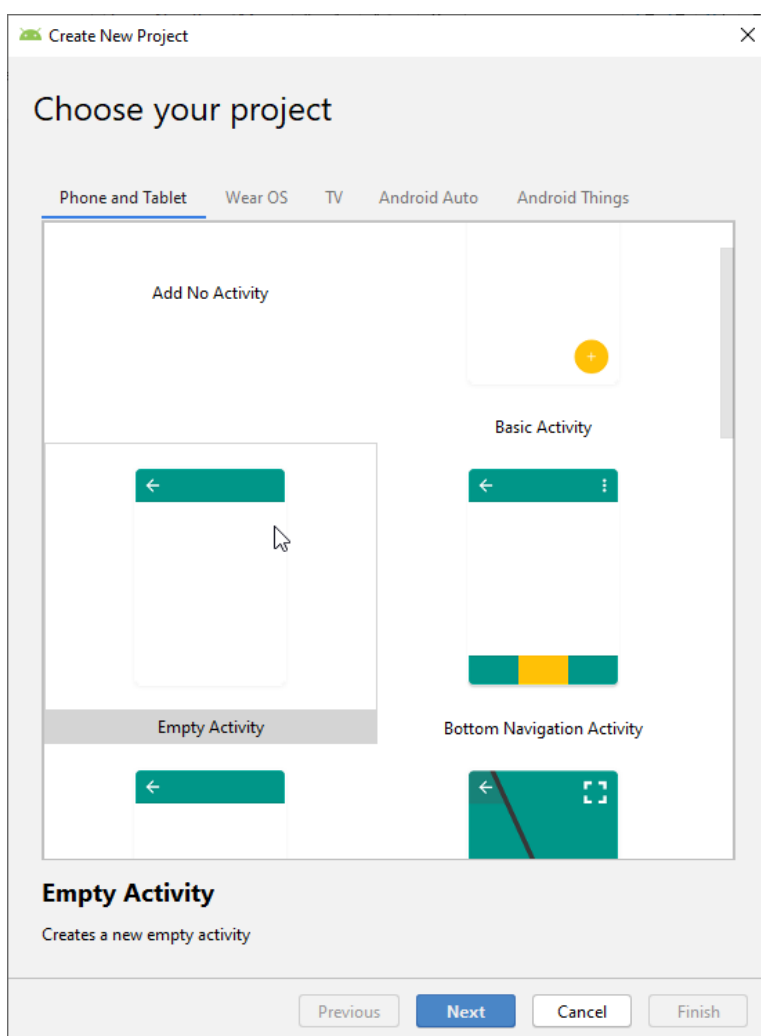
3.1. Създаване на скелет на приложението

За да създадем ново приложение, ние ще стартираме Android Studio и от началния екран (фиг. 3.3) ще изберем *Start a new Android Studio Project*.

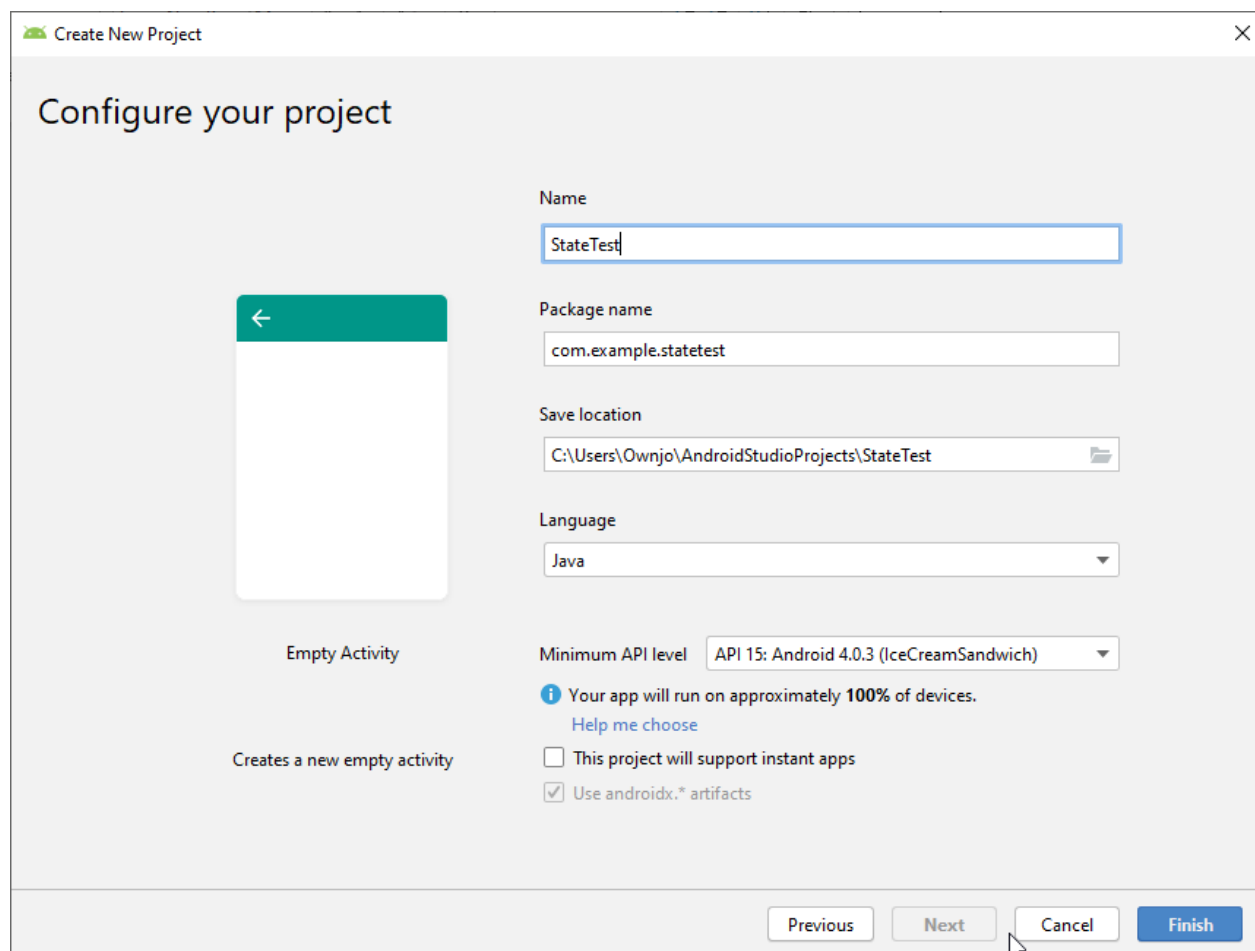


Фиг.3.3. Начален екран на Android Studio

На следваща стъпка, Android Studio ще изведе запитване дали искаме да започнем приложението със създаване на една от дискутираните дейности по-горе. Ние ще изберем *Empty Activity* от секцията *Phone & Tablet*, която е активна по подразбиране (вж. фиг. 3.4). След като изберем дейността, следва да натиснем бутона *Next*, който ще ни придвижи към следваща стъпка, която се състои в попълване на диалогов прозорец с различни параметри на проекта (фиг. 3.5).



Фиг. 3.4. Избор на начална дейност за проекта



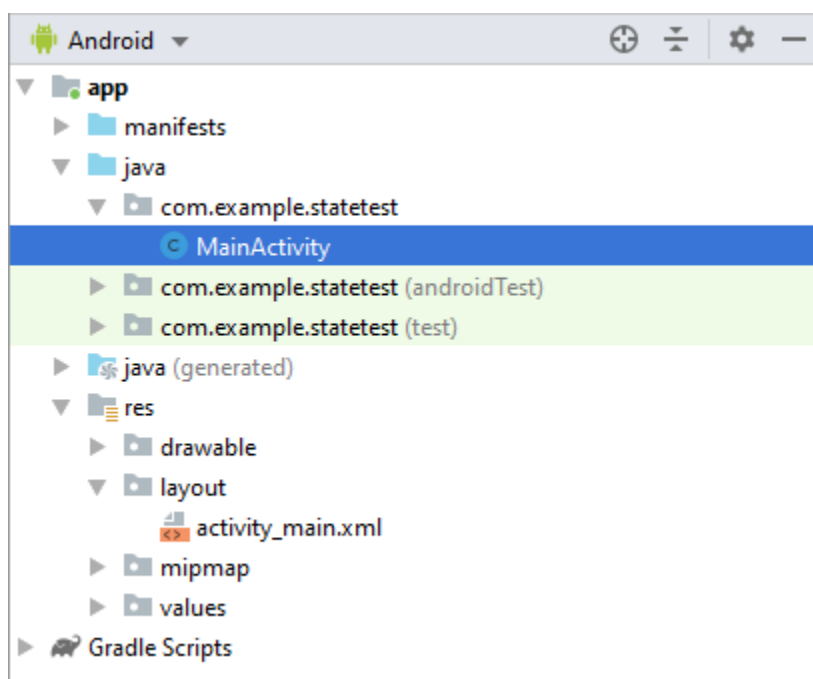
Фиг. 3.5. Конфигурация на проекта

В полето за име на проекта, ние ще въведем *StateTest*. На база това име, Android ще генерира име на пакета и име на папка, в която ще се съхраняват файловете на проекта. Ние ще оставим тези параметри непроменени, и ще изберем *Java* от полето *Language*.

Много важна настройка, която трябва да бъде определена на този етап, е изборът на минимално ниво на програмния интерфейс (*Minimum API Level*) това ниво определя най-ранната версия на операционната система Android, на която ще може да работи нашето приложение. В случая ще изберем API 15, което отговаря на Android 4.0.3 – Ice Cream Sandwich.

С това приключва въвеждането на параметри и ние ще натиснем бутона *Finish*.

След извършването на това действие, Android Studio ще генерира скелет на приложение с една дейност. От панела *Project* в лявата част на работния прозорец (фиг. 3.6) можем да видим, че този скелет съдържа една дейност с име *MainActivity*, която е генерирана според шаблона Empty Activity и следователно има един празен Layout файл, *activity_main.xml*.

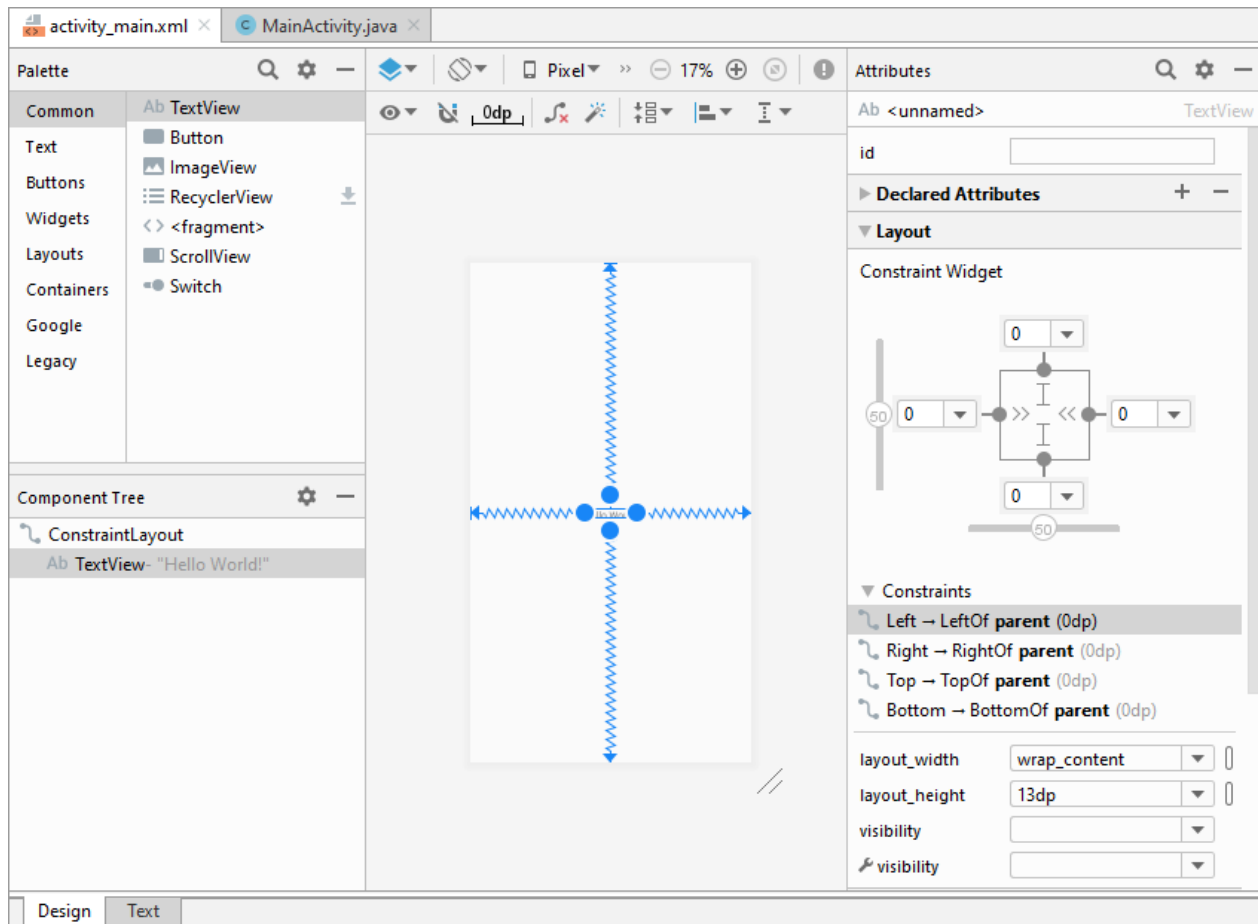


Фиг. 3.6. Панел Project

3.2. Модифициране на layout файла

На тази стъпка ще модифицираме лейаут файла *activity_main.xml* с графичния редактор. За целта в панела *Project* (вж. фиг. 3.6) ще щракнем на папка *res*, която съдържа ресурсите на приложението. Ресурсите са групирани в различни категории, всяка от които има собствена подпапка. Лейаут файловете се намират в папката *layout*.


При щракването върху името на файла, в главния прозорец на Android Studio ще се отвори графичният дизайнер.



Фиг. 3.7. Графичен дизайнер

Лейаут файла дефинира разположението на графичните елементи. Всеки такъв файл дефинира йерархия от визуални и невизуални компоненти, вложени един в друг. Йерархията на вложените компоненти е изложена в панела *Component tree*, разположен в лявата част на графичния дизайнер. От този панел е видно, че разположението, дефинирано от лейаут файла е от тип *ConstraintLayout*, който съдържа един визуален компонент от тип *TextView*. *TextView* компонентите се използват за показване на текст.

Ние ще щракнем именно върху този компонент, което ще го маркира и ще произведе екрана, показан на фиг. 3.7. Целта ни е да направим така, че *TextView* компонента да заеме целия екран на приложението. За целта в десния панел ще щракнем върху знаците **>>** и **<<** в секция *Layout* на десния панел

(панел *Attributes*) на редактора. Щракването ще продължи, докато тези знаци се заменят от знаците .

В хода на това действие, *TextView* компонентът ще заеме цялата ширина на екрана. По подобен начин следва да се модифицират и вертикалните знаци, така че *TextView* компонентът да заеме цялото налично пространство.

След това, в същия панел, ние ще скролираме до секция *Common Attributes*, която съдържа полето *text* и ще изчистим от него текста *Hello, World*.

Ще приключим подготовката на layout файла с това, че ще дадем уникално име на нашия *TextView* компонент. За целта ще се придвижим до най-горната част на панела и ще въведем *text* в полето *id*. Окончателният текстов вариант на нашия layout файл е поместен по-долу:

activity_main.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3  xmlns:android="http://schemas.android.com/apk/res/android"
4  xmlns:app="http://schemas.android.com/apk/res-auto"
5  xmlns:tools="http://schemas.android.com/tools"
6  android:layout_width="match_parent"
7  android:layout_height="match_parent"
8  tools:context=".MainActivity">
9
10  <TextView
11      android:id="@+id/text"
12      android:layout_width="0dp"
13      android:layout_height="0dp"
14      app:layout_constraintBottom_toBottomOf="parent"
15      app:layout_constraintLeft_toLeftOf="parent"
16      app:layout_constraintRight_toRightOf="parent"
17      app:layout_constraintTop_toTopOf="parent" />
18
19  </androidx.constraintlayout.widget.ConstraintLayout>
```

3.3. Модификация на програмния код

На този етап ние ще модифицираме метода *OnCreate* на нашия клас *MainActivity*, след което ще създадем методите *OnStart*, *OnResume*, *OnPause*,

OnStop и *OnRestart*. След изпълнение на тези действия, програмният код на MainActivity ще изглежда така:

GpsLock.java

```

1  package com.example.statetest;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6  import android.widget.TextView;
7
8  import java.util.Calendar;
9
10 public class MainActivity extends AppCompatActivity {
11
12     Calendar cal;
13
14     private void registerEvent(String event) {
15         TextView text = findViewById(R.id.text);
16         text.setText(text.getText()+"\n"+ cal.getTime()+" " +event + " called");
17     }
18
19     @Override
20     protected void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         setContentView(R.layout.activity_main);
23         cal = Calendar.getInstance();
24
25         registerEvent("onCreate");
26     }
27
28     @Override
29     protected void onStart() {
30         super.onStart();
31         registerEvent("onStart");
32     }
33
34     @Override
35     protected void onResume() {
36         super.onResume();
37         registerEvent("onResume");
38     }
39
40     @Override
41     protected void onPause() {
42         super.onPause();
43         registerEvent("onPause");
44     }
45
46     @Override
47     protected void onStop() {
48         super.onStop();
49         registerEvent("onStop");
50     }
51
52     @Override
53     protected void onRestart() {

```

```
54     super.onRestart();  
55     registerEvent("onRestart");  
56 }  
57 }
```

Основната функционалност на приложението е поместена в метода *registerEvent* (редове 14-17). Този метод взема един аргумент – текстов низ, който съдържа типа на събитието. *registerEvent* записва този тип, заедно с точната дата и час, в съдържанието на *textView* елемента с име *text*.

Читателите следва да обърнат внимание на ред 15, където методът *findViewById* (който е наследен от родителския клас) се използва за да се достъпи *TextView* елемента. Класът *R*, както вече бе споменато, се генерира автоматично при компилация и съдържа идентификаторите на всички ресурси.

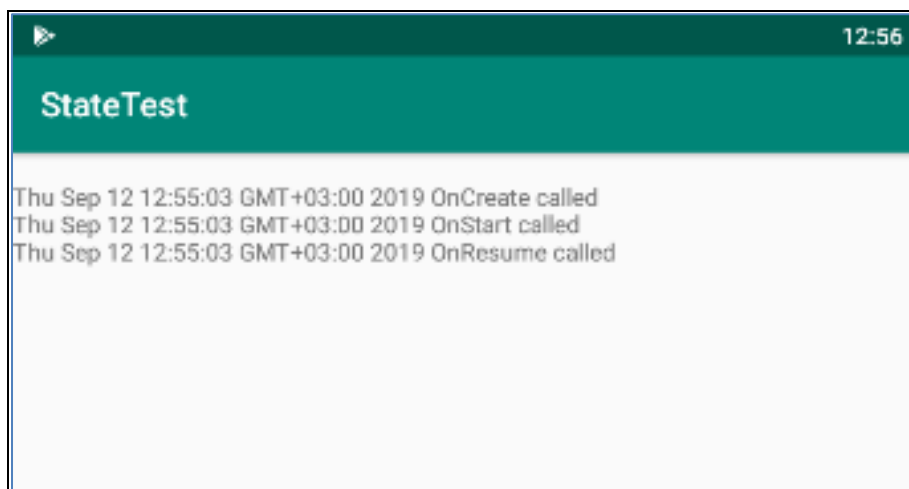
Ние нарекохме нашия *TextView* елемент “*text*”, поради което Android Studio създаде числовата константа *R.id.text*, която уникално идентифицира нашия елемент. *findViewById* използва стойността на тази константа, за да извлече обект, който представлява *TextView* елемента в програмата.

Този обект има методи *getText* и *setText*, които се използват от *registerEvent* съответно за получаване и задаване на стойности за текста, поместен в *TextView* елемента.

Наблюдателните читатели може би са забелязали, че нашата програма не прихваща *OnDestroy* метода на родителския клас, тъй като когато той се извика, целият графичен потребителски интерфейс на програмата вече няма да бъде видим, а веднага след изпълнението на метода, екранът на дейността ще бъде унищожен заедно с цялото копие на дейността в оперативната памет на устройството. Поради тази причина *registerEvent* никога не би могъл да покаже *onDestroy* събитие на потребителя.

Фиг. 3.8. показва снимка на дейността на мобилното приложение. Читателите следва да построят приложението и да го инсталират на

устройството или емулятора си и да преминат през няколко цикъла на обръщане на ориентацията на устройството.



Фиг. 3.8. Екран на мобилното приложение

При всяко такова обръщане на пръв поглед съобщенията няма да се променят, но читателите трябва да забележат, че датите на съобщенията са различни всеки път, което потвърждава, при промяна на ориентацията, системата извиква *onDestroy* и унищожава дейността, преди да я стартира наново.

Читателите също така следва да експериментират с бутона Home и стартирането на друга задача, последвано от връщане към тази на настоящото приложение. Всичко това би им помогнало да се ориентират още по-добре в последователностите, в които операционната система извиква колбач методите за жизнения цикъл на дейността.