Лабораторная работа № 1 по теме:
«Компоновка элементов управления. Обработка событий»

Теоретическая часть

Платформа Android представляет собой программный стек для мобильных устройств (смартфонов, планшетов) и не только (часов, фитнес-браслетов, телевизоров и т.д.), который включает операционную систему, программное обеспечение промежуточного слоя (связующее ПО), а также основные пользовательские приложения.

Android-приложения состоят из отдельных компонентов, которые могут быть вызваны независимо друг от друга. **Компоненты Android-приложения** – это самостоятельные элементы в структуре приложения, определяющие работу приложения в целом. Существует 4 типа компонентов Android-приложения: активити, сервис (служба), поставщик контента, приемник широковещательных сообщений (намерений).

Активити (англ. Activity) в русскоязычных источниках также называют активностью, деятельностью или операцией. Активити представляет собой визуальный пользовательский интерфейс для приложения. Каждое активити — это отдельное окно приложения.

Для описания архитектуры расположения элементов пользовательского интерфейса конкретного активити используются компоновки. **Компоновка (макет, разметка)** может быть объявлена двумя способами: в специальном xml-файле или в программном коде активити путем создания объектов классов View и ViewGroup. Эти два способа можно совмещать, например, объявить элементы интерфейса в xml-файле и управлять их свойствами в программном коде.

Объявление элементов интерфейса в xml-файле позволяет отделить представление (англ. view) приложения от кода, управляющего поведением приложения (англ. controller), в соответствии с парадигмой MVC (англ. Model-View-Controller, Модель-Представление-Контроллер). Отделение

представления от контроллера позволяет изменять интерфейс активити без внесения исправлений в исходный код и его повторной компиляции. Также есть возможность создавать разные xml-файлы компоновок для разных размеров и ориентаций экрана, для разных языков и т.д. Помимо этого, использование xml-компоновок упрощает визуализацию структуры интерфейса пользователя и упрощает его отладку.

Для всех элементов пользовательского интерфейса (например, кнопок, текстовых полей, переключателей) базовым классом является **класс View** (представление). Для коллекции объектов View базовым является **класс ViewGroup** (группа представлений). Объект ViewGroup является контейнером, который может содержать объекты View и ViewGroup. Класс ViewGroup также является базовым для подклассов компоновок, например, линейной компоновки (англ. LinearLayout).

В xml-файле компоновки должен быть равно **один корневой элемент,** который должен быть потомком класса View или ViewGroup. Чаще всего в качестве корневого элемента используется один из стандартных классов компоновок: LinearLayout (линейная компоновка), RelativeLayout (относительная компоновка), ConstraintLayout (компоновка на основе ограничений), WebView (представление веб-страницы), AdapterView (адаптивное, заполняемое во время выполнения представление).

XML-файл компоновки приложения хранится в каталоге res/layout/ проекта Android (рис. 1). Для назначения компоновки активити необходимо в методе обратного вызова onCreate () для данного активити вызвать метод setContentView () и передать ему в качестве параметра ссылку на ресурс компоновки в следующем виде: R.layout.layout_name, где layout_name – имя файла компоновки без расширения .xml (листинг 1).

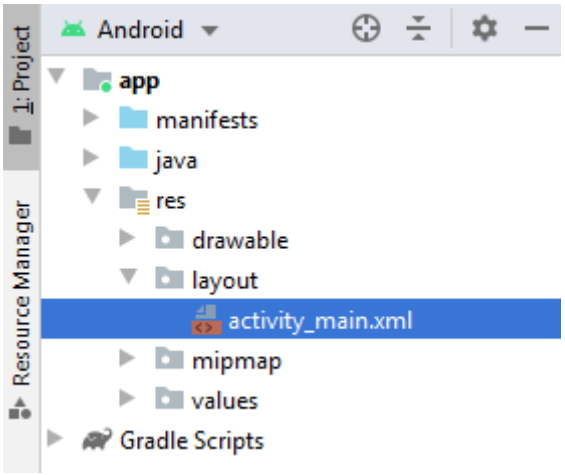

Рисунок 1. Расположение XML-файла компоновки.

Листинг 1. MainActivity.java

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Линейная компоновка

При использовании линейной компоновки дочерние элементы, находящиеся внутри корневого элемента **LinearLayout**, располагаются либо в ряд по горизонтали, либо в столбец по вертикали, в зависимости от параметра android:orientation.

При формировании интерфейса пользователя компоновка может содержать ещё одну или несколько вложенных компоновок. Например, внутрь линейной компоновки с вертикальной ориентацией могут быть вложены компоновки с горизонтальной ориентацией. Однако не рекомендуется использовать сложную иерархию компонентов, так как это замедляет отрисовку интерфейса.

Рассмотрим пример (листинг 2).

Листинг 2. Компоновка activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 1" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:id="@+id/button2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Button 2" />
        <Button
            android:id="@+id/button3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Button 3" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:id="@+id/button4"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Button 4" />
        <Button
        android:id="@+id/button5"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button 5" />
        <Button
            android:id="@+id/button6"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Button 6" />
    </LinearLayout>
</LinearLayout>
```

В данном примере внутри компоновки LinearLayout с вертикальной ориентацией располагается кнопка и два вложенных элемента LinearLayout с горизонтальной ориентацией, в которых в свою очередь помещены две и три кнопки соответственно (рис 2).

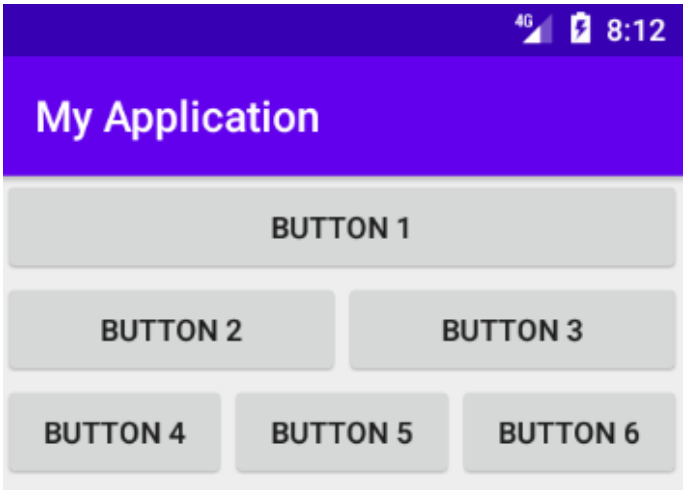

Рисунок 2. Линейная компоновка.

Рассмотрим атрибуты, использованные в данной компоновке (табл. 1).

Таблица 1. Атрибуты элементов в xml-компоновке.

Атрибут	**Назначение атрибута**	**Пример значения атрибута**	**Описание значения атрибута**
android:orientation	Задаёт расположение элементов внутри компоновки по горизонтали или по вертикали	vertical	Расположение элементов по вертикали
		horizontal	Расположение элементов по горизонтали
android:layout_width	Задаёт ширину элемента	match_parent	Ширина по размеру родительского контейнера
		wrap_content	Ширина по размеру содержимого
		100dp	Ширина в 100 плотностно-независимых пикселей
android:layout_height	Задаёт высоту элемента	match_parent	Высота по размеру родительского контейнера
		wrap_content	Высота по размеру содержимого
		100dp	Высота в 100 плотностно-независимых пикселей

android:layout_weight	Назначает индивидуальный вес для дочернего элемента	1	Вес равен единице
android:id	Задаёт уникальный идентификатор	@+id/new_id	Создание идентификатора new_id
android:text	Задаёт текст, отображаемый в элементе	Текст	Фиксированный текст (англ. hardcoded text)
		@string/name	Ссылка на строковый ресурс с именем name

Атрибут android:layout_weight назначает индивидуальный вес для дочернего элемента. Этот атрибут определяет "важность" объекта View и позволяет этому элементу расширяться, чтобы заполнить любое оставшееся пространство в родительском объекте View. Заданный по умолчанию вес является нулевым.

Например, если есть три кнопки, и двум из них объявлен вес со значением 1, в то время как третьей не дается никакого веса (0), третья кнопка без веса не будет расширяться и займет область, определяемую размером текста, отображаемого на ней. Другие две расширятся одинаково, чтобы заполнить остаток пространства, не занятого третьей кнопкой. Если третьей кнопке присвоить вес 2 (вместо 0), она будет объявлена как "более важная", чем две других, так что третья кнопка получит 50% общего пространства, в то время как первые две получат по 25% общего пространства.

В **атрибутах android:layout_width и android:layout_height** для задания ширины могут использоваться следующие единицы измерения:

- px (англ. pixels) — пиксели;
- dp (англ. density-independent pixels) — независимые от плотности пиксели;
- sp (англ. scale-independent pixels) — независимые от масштабирования пиксели;
- in (англ. inches) — дюймы;

- pt (англ. points) — 1/72 дюйма;
- mm (англ. millimeters) — миллиметры.

Одна из ошибок, которую следует избегать при создании компоновок, — это использование пикселей (px) для определения расстояний или размеров элементов. Определение размеров с помощью пикселей создаёт проблемы, поскольку разные экраны имеют разную плотность пикселей, и одно и то же количество пикселей может соответствовать разным физическим размерам на разных устройствах.

Независимые от плотности пиксели (плотностно-независимые пиксели) — это виртуальные пиксели, которые следует использовать при определении разметки интерфейса пользователя, чтобы выразить положения и позиции в разметке способом, независящим от плотности экрана устройства (рис. 3).

Плотностно-независимый пиксель равен одному физическому пикселю на экране с плотностью 160 dpi (точек на дюйм), что является базовой плотностью для экранов среднего размера. Во время работы система пересчитывает любые масштабирования единиц dp, при необходимости, на основе фактической плотности используемого экрана. Перевести dp единицы в экранные пиксели, использую следующую формулы:

$$px = \frac{dp \times dpi}{160}, \quad (1)$$

где px — количество экранных писклей,

dp — количество плотностно-независимых пикселей,

dpi — плотность экрана мобильного устройства.

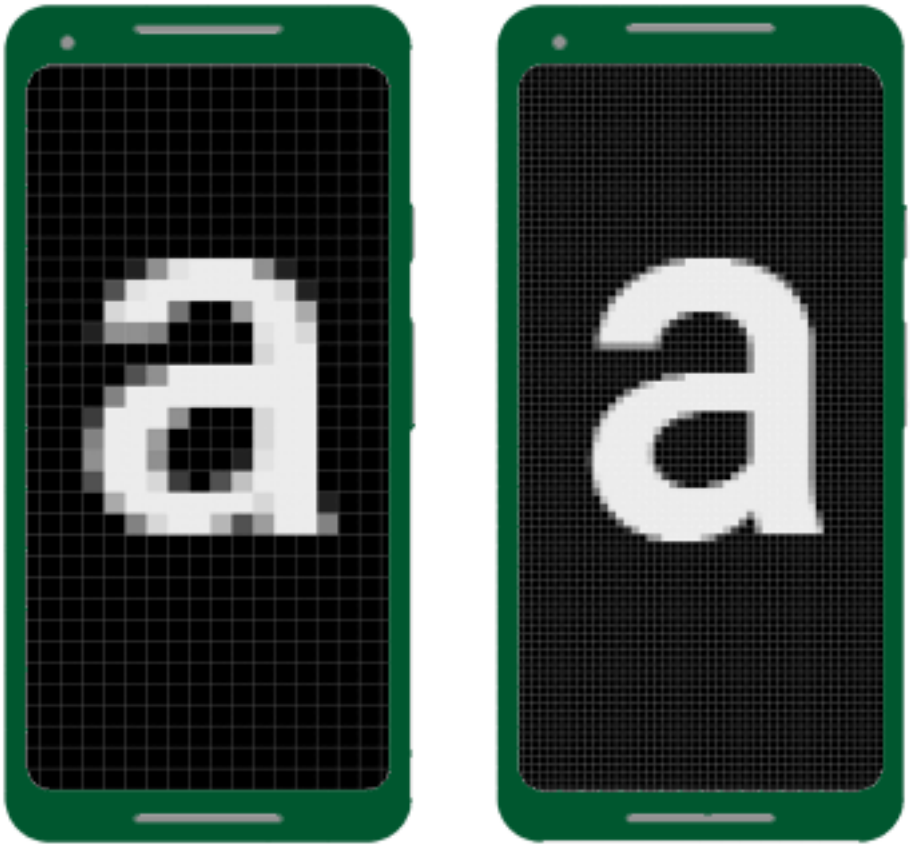

Рисунок 3. Два экрана одинакового размера, но с разной плотностью экрана.

Например, на экране с плотностью 320 dpi, 1 dp равен 2 экранным пикселям. При проектировании графического интерфейса приложения следует всегда использовать единицы dp, чтобы быть уверенным в его корректном отображении на экранах с различной плотностью.

Для задания размеров текста рекомендуется использовать единицы sp — **независимые от масштабирования пиксели,** которые, как и dp, зависят от плотности экрана, но результат будет масштабироваться также в соответствии с настройками размера шрифта пользователя.

Относительная компоновка

При использовании относительной компоновки **RelativeLayout** расположение дочерних объектов задаётся относительно родительского объекта или относительно соседних дочерних элементов (по идентификатору элемента).

В RelativeLayout дочерние элементы расположены так, что, если первый элемент расположен по центру экрана, другие элементы, выровненные относительно первого элемента, будут выровнены относительно центра экрана. При таком расположении, при объявлении компоновки в XML-файле, элемент, на который будут ссылаться для позиционирования другие объекты, должен быть объявлен раньше, чем элементы, которые обращаются к нему по его идентификатору.

Для создания **идентификатора** используется атрибут **android:id**. В качестве значения атрибута должно быть указано сочетание символов @+id/ и уникальное имя идентификатора, например, атрибут android:id="@+id/button1" создаёт идентификатор элемента с именем button1.

Если в программном коде мы не работаем с некоторыми элементами пользовательского интерфейса, создавать идентификаторы для них необязательно, однако определение идентификаторов для объектов важно при создании RelativeLayout. В компоновке RelativeLayout расположение элемента View (представления) может определяться относительно другого элемента (таблица 2), на который ссылаются через его уникальный идентификатор:

android:layout_toLeftOf="@id/TextView1"

Обратите внимание, что атрибуты, которые обращаются к идентификаторам относительных элементов (например, layout_toLeftOf), используют синтаксис относительного ресурса @id/id.

Таблица 2. Атрибуты, использующиеся для определения расположения элемента в RelativeLayout.

Атрибут	**Описание**
android:layout_above	Располагает нижний край данного представления над заданным по id представлением
android:layout_alignBaseline	Помещает базовую линию представления на базовую линию заданного представления

android:layout_alignBottom	Делает нижний край представления совпадающим с нижним краем заданного представления
android:layout_alignEnd	Делает конечный край представления совпадающим с конечным краем заданного представления
android:layout_alignLeft	Делает левый край представления совпадающим с левым краем заданного представления
android:layout_alignParentBottom	Если true, то нижний край представления совпадает с нижним краем родителя
android:layout_alignParentEnd	Если true, то конечный край представления совпадает с конечным краем родителя
android:layout_alignParentLeft	Если true, то левый край представления совпадает с левым краем родителя
android:layout_alignParentRight	Если true, то правый край представления совпадает с правым краем родителя
android:layout_alignParentStart	Если true, то начальный край представления совпадает с начальным краем родителя
android:layout_alignParentTop	Если true, то верхний край представления совпадает с верхним краем родителя
android:layout_alignRight	Делает правый край представления совпадающим с правым краем заданного представления
android:layout_alignStart	Делает начальный край представления совпадающим с начальным краем заданного представления
android:layout_alignTop	Делает верхний край представления совпадающим с верхним краем заданного представления
android:layout_alignWithParentIfMissing	Если установлено значение true, родительский элемент будет использоваться в качестве привязки, когда привязка не может быть найдена для layout_toLeftOf, layout_toRightOf и т.д.
android:layout_below	Располагает верхний край представления под заданным представлением
android:layout_centerHorizontal	Если true, центрирует дочерний элемент по горизонтали внутри родителя
android:layout_centerInParent	Если true, центрирует дочерний элемент по горизонтали и вертикали внутри родителя

android:layout_centerVertical	Если true, центрирует дочерний элемент по вертикали внутри родителя
android:layout_toEndOf	Помещает начальный край представления в конец заданного представления
android:layout_toLeftOf	Позиционирует правый край представления слева от заданного представления
android:layout_toRightOf	Позиционирует левый край представления справа от заданного представления
android:layout_toStartOf	Позиционирует конечный край представления в начало заданного представления

Следует уточнить, что понимается под начальным и конечным краями представления. Для большинства языков начальным краем является левый, а конечным краем ─ правый. При направлении письма справа налево (англ. RTL — Right-to-left), например, в арабском языке, иврите, начальным краем является правый, а конечным краем ─ левый.

При использовании RelativeLayout расположение элементов зависит от разрешения и ориентации экрана мобильного устройства. Если вы будете использовать RelativeLayout в собственных приложениях, всегда проверяйте внешний вид окна для различных разрешений и ориентации экрана, поскольку возможно наложение элементов друг на друга.

Макет на основе ограничений

Макет или **компоновка на основе ограничений** (ConstraintLayout) стал доступен для устройств начиная с уровня Android API 9. Он позволяет создавать большие и сложные компоновки с плоской иерархией представлений (без вложенных групп представлений).

ConstraintLayout похож на RelativeLayout в том, что для всех вложенных элементов задаётся их размещение относительно друг друга или родительского контейнера, но он более гибкий и имеет лучшую производительность, чем RelativeLayout, и его проще использовать с **визуальным редактором макетов** в Android Studio. С его помощью можно

создать компоновку целиком путем перетаскивания объектов в режиме Design вместо редактирования xml-файла компоновки (рис. 4).

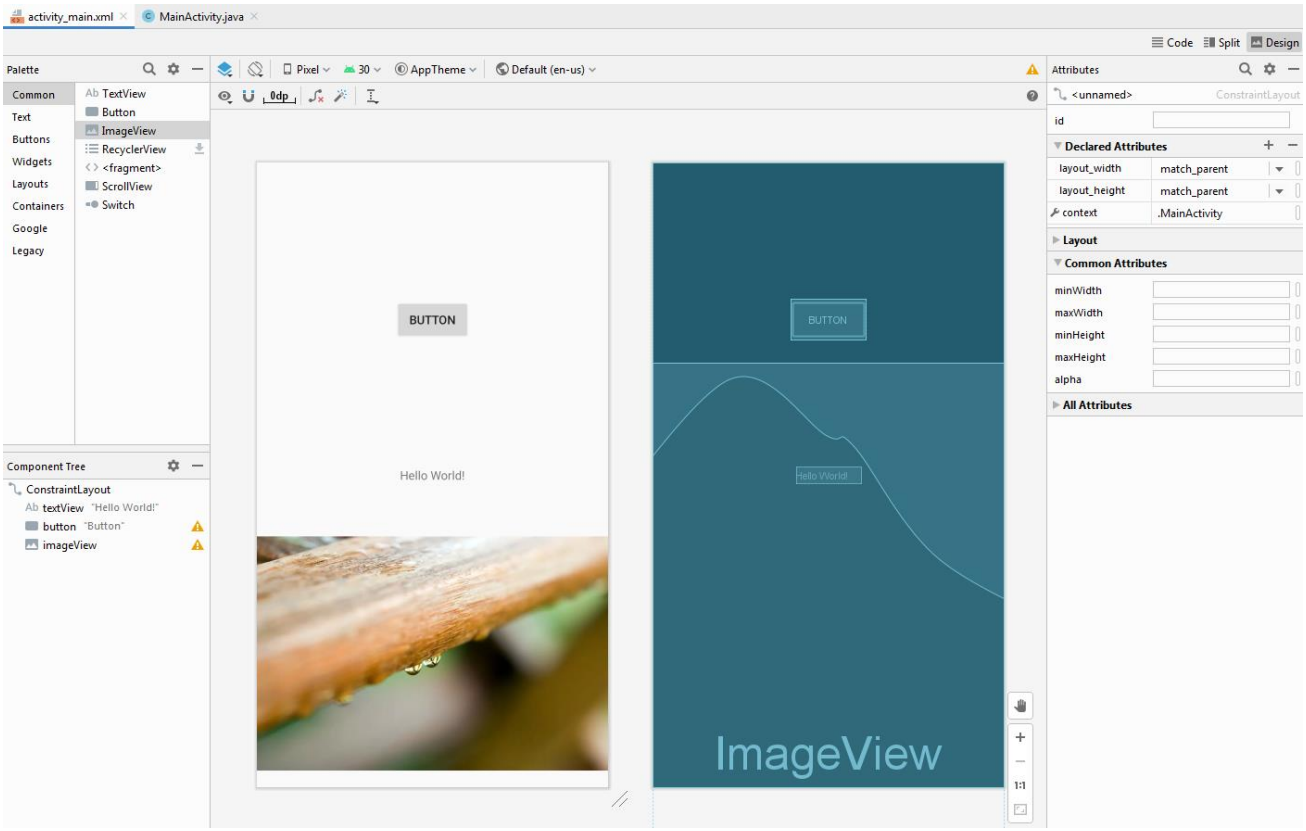

Рисунок 4. Визуальный редактор макетов в Android Studio.

Для переключения между режимом визуального редактирования компоновки и режимом редактирования xml-кода используются кнопки Code, Split и Design в верхнем правом углу редактора.

В таблице 3 приведено соответствие атрибутов элементов RelativeLayout и ConstraintLayout, задающих одинаковое положение элементов.

Таблица 3. Соответствие атрибутов элементов RelativeLayout и ConstraintLayout.

Атрибут RelativeLayout	**Атрибут (атрибуты) ConstraintLayout**
android:layout_above="@id/view"	app:layout_constraintBottom_toTopOf="@id/view"
android:layout_alignBaseline= "@id/view"	app:layout_constraintBaseline_toBaselineOf= "@id/view"
android:layout_alignBottom="@id/view"	app:layout_constraintBottom_toBottomOf= "@id/view"
android:layout_alignEnd="@id/view"	app:layout_constraintEnd_toEndOf="@id/view"
android:layout_alignLeft="@id/view"	app:layout_constraintLeft_toLeftOf="@id/view"
android:layout_alignParentBottom="true"	app:layout_constraintBottom_toBottomOf="parent"

android:layout_alignParentEnd="true"	app:layout_constraintEnd_toEndOf="parent"
android:layout_alignParentLeft="true"	app:layout_constraintLeft_toLeftOf="parent"
android:layout_alignParentRight="true"	app:layout_constraintRight_toRightOf="parent"
android:layout_alignParentStart="true"	app:layout_constraintStart_toStartOf="parent"
android:layout_alignParentTop="true"	app:layout_constraintTop_toTopOf="parent"
android:layout_alignRight="@id/view"	app:layout_constraintRight_toRightOf="@id/view"
android:layout_alignStart="@id/view"	app:layout_constraintStart_toStartOf="@id/view"
android:layout_alignTop="@id/view"	android:layout_alignTop="@id/view"
android:layout_below="@id/view"	app:layout_constraintTop_toBottomOf="@id/view"
android:layout_centerHorizontal="true"	app:layout_constraintLeft_toLeftOf="parent" app:layout_constraintStart_toStartOf="parent" app:layout_constraintRight_toRightOf="parent" app:layout_constraintEnd_toEndOf="parent"
android:layout_centerInParent="true"	app:layout_constraintBottom_toBottomOf="parent" app:layout_constraintLeft_toLeftOf="parent" app:layout_constraintStart_toStartOf="parent" app:layout_constraintRight_toRightOf="parent" app:layout_constraintEnd_toEndOf="parent" app:layout_constraintTop_toTopOf="parent"
android:layout_centerVertical="true"	app:layout_constraintBottom_toBottomOf="parent" app:layout_constraintTop_toTopOf="parent"
android:layout_toEndOf="@id/view"	app:layout_constraintStart_toEndOf="@id/view"
android:layout_toLeftOf="@id/view"	app:layout_constraintRight_toLeftOf="@id/view"
android:layout_toRightOf="@id/view"	app:layout_constraintLeft_toRightOf="@id/view"
android:layout_toStartOf="@id/view"	app:layout_constraintEnd_toStartOf="@id/view"

Базовые виджеты и обработка событий

Виджет — это объект View, который служит интерфейсом для взаимодействия с пользователем. Фактически, виджеты — это элементы управления. В Android имеется набор готовых виджетов, таких как кнопки, переключатели, текстовые поля и т.д., с помощью которых можно быстро сформировать пользовательский интерфейс приложения.

Текстовые поля в Android представлены двумя классами: **TextView** и **EditText**. TextView предназначен для отображения текста без возможности его редактирования пользователем, а также может использоваться как элемент для отображения текстовых данных в контейнерных виджетах для отображения списков. EditText ─ это поле для ввода текста пользователем. EditText является потомком TextView.

Ширина и высота TextView и EditText задаются как и для любого объекта View с помощью атрибутов android:layout_width и android:layout_height. Если необходимо обращаться к этим элементам в программном коде, например, для изменения их свойств или получения введенного текст, то необходимо задать идентификатор android:id. Для задания текста, отображаемого элементом, используется атрибут android:text. В него может передаваться как сам текст в фиксированном виде, так и ссылка на **строковый ресурс**.

Некоторые объекты (изображения, строковые константы, цвета, анимация, стили и т.д.) принято хранить за пределами исходного кода в виде отдельных файлов ресурсов. Их легче поддерживать, обновлять и редактировать. Строковые константы хранятся в файле strings.xml в папке res/values.

Листинг 3. Строковые ресурсы strings.xml

```
<resources>
    <string name="app_name">App Name</string>
</resources>
```

По умолчанию в файле strings.xml создаётся корневой элемент resources, внутри которого размещается единственный элемент string, содержащий имя приложения. В атрибуте name элемента string указывается имя строкового ресурса. Все другие строковые ресурсы должны размещаться внутри корневого элемента resources.

Для задания текста, отображаемого элементом TextView или EditText (или другими потомками TextView, например, кнопкой Button), в качестве ссылки на строковый ресурс, в xml-коде компоновки в атрибут android:text передаётся ссылка в следующем виде:

android:text="@string/string_name"

Для тех же целей в программном java-коде необходимо найти элемент с помощью метода findViewById по его идентификатору и изменить текст методом setText. Например:

```
TextView text = (TextView)findViewById(R.id.text);
text.setText(R.string.text1);
```

В таблице 4 представлены часто использующиеся xml-атрибуты TextView и java-методы, позволяющие задать те же свойства.

Таблица 4. Соответствие XML-атрибутов и методов в классах представлений.

XML-атрибут	**Java-метод**	**Назначение**
android:text	setText	Назначить текст
android:textColor	setTextColor	Назначить цвет текста
android:textColorHighlight	setHighlightColor	Назначить цвет выделения текста
android:textSize	setTextSize	Назначить размер текста
android:textStyle	setTypeface (null, int)	Назначить стиль текста (жирный, курсив, нормальный)
android:typeface	setTypeface (Typeface)	Назначить тип шрифта (с засечками, без засечек, моноширинный)

Основной метод класса EditText — getText (), который возвращает текст, содержащийся в элементе EditText.

Для отображения графики предназначен виджет ImageView. Как и элемент TextView, который является базовым виджетом для текстового наполнения пользовательского интерфейса, ImageView является базовым элементом для графического наполнения и может использоваться в контейнерных виджетах для отображения графики.

Класс ImageView может загружать изображения из различных источников, как из ресурсов приложения, так и из внешних источников. В этом классе существует несколько методов для загрузки изображений:

- setImageResource (int resId) — загружает изображение по его идентификатору ресурса;
- setImageURI (Uri uri) — загружает изображение по его URI;

- setImageBitmap (Bitmap bitmap) — загружает растровое изображение.

Ресурсы изображения хранятся в папке res/drawable. Для загрузки ресурса изображения в коде, можно передать идентификатор ресурса в виде параметра метода. Например, с помощью метода setImageResource () можно указать использование виджетом ImageView ресурса res/drawable/image.png:

Листинг 4. Задание источника изображения в ImageView в коде

```
ImageView myImage = (ImageView) findViewById(R.id.imageview);
myImage.setImageResource(R.drawable.image);
```

При добавлении в компоновку элемента ImageView можно указать ссылку на источник изображения в атрибуте android:src, например:

Листинг 5. Задание источника изображения в ImageView в xml

```
<ImageView
    android:id="@+id/imageview"
    android:src="@drawable/image"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

Кнопки и флажки (рис. 5) на платформе Android представлены следующими основными классами:

- Button (кнопка).
- CheckBox (переключатель).
- ToggleButton (кнопка включено/выключено).
- RadioButton (радиокнопка).
- ImageButton (кнопка с изображением).

Почти у всех вышеперечисленных виджетов базовым классом является TextView, от которого они наследуют многочисленные методы для отображения и форматирования текста. Исключение составляет виджет ImageButton, который является потомком класса ImageView и представляет собой кнопку с изображением без текста.

Рисунок 5. Кнопки и флажки

Класс Button (кнопка) — это самый простой из всех элементов управления и при этом самый используемый. Чаще всего кнопка требует написания кода обработки события нажатия onClick.

Классы CheckBox, ToggleButton и RadioButton являются потомками класса CompoundButton, который в свою очередь является потомком Button. Класс CompoundButton представляет базовую функциональность для кнопок с двумя состояниями — checked (выделено/включено) и unchecked (не выделено/не включено). При нажатии состояние кнопки изменяется на противоположное. Чаще всего кнопка требует написания кода обработки события изменения состояния onCheckedChanged. Виджеты RadioButton обычно используются в составе группы — контейнера RadioGroup, внутри которого только одна RadioButton может иметь состояние checked.

Для организации взаимодействия виджетов с пользователем необходимо определить обработчик события и зарегистрировать его для данного элемента. Класс View содержит коллекцию вложенных интерфейсов, которые называются On…Listener (), в каждом из которых объявлен единственный абстрактный метод. Этот метод необходимо переопределить в вашем классе. Его будет вызывать система Android, когда пользователь будет взаимодействовать с экземпляром View (например, с кнопкой), к которому был подсоединен слушатель события. Класс View содержит несколько вложенных интерфейсов, в том числе:

- OnClickListener — слушатель события нажатия;
- OnLongClickListener — слушатель события длинного нажатия;
- OnFocusChangeListener — слушатель события изменения фокуса ввода;
- OnKeyListener — слушатель события нажатия аппаратной кнопки;
- OnTouchListener — слушатель события касания и перемещения пальца.

В классе CompoundButton содержится вложенный интерфейс:

- OnCheckedChangeListener — слушатель события изменения состояния CompoundButton.

Если требуется, например, чтобы кнопка (с идентификатором myButton) получила уведомление о нажатии пользователем на неё, необходимо в классе активити реализовать интерфейс OnClickListener и определить его метод обратного вызова onClick, куда будет помещен код обработки нажатия кнопки (код поменяет текст в TextView с идентификатором myText на «Вы нажали на кнопку»), и зарегистрировать слушатель события с помощью метода setOnClickListener (листинг 6).

Листинг 6. Обработка нажатия на кнопку.

```
Button myButton = (Button)findViewById(R.id.myButton);
TextView myText = (TextView)findViewById(R.id.myText);
myButton.setOnClickListener(new View.OnClickListener()
{
    public void onClick(View v)
    {
        myText.setText("Вы нажали на кнопку");
    }
});
```

Динамическое создание элементов управления

В некоторых случаях во время выполнения программы нам может потребоваться создание компоновки и элементов управления в коде программы.

В программном коде при создании компоновки используются те же принципы, как и при создании компоновки в xml-файле. Платформа Android в пакете android.widget предоставляет набор классов для создания компоновок способом, аналогичным объявлению в xml-файле.

Для создания корневого элемента разметки создаётся экземпляр класса разметки, в конструктор которого передается контекст приложения. Контекст приложения — это набор информации о среде, в которой выполняется приложение. В классе окна приложения контекст приложения можно получить вызовом метода getApplicationContext (). Например, создать линейную компоновку можно следующим образом:

```
LinearLayout layout = new LinearLayout(getApplicationContext());
```

В классе, реализующем Activity, необязательно явно вызывать метод getApplicationContext () для получения контекста приложения, вместо этого можно использовать ссылку на данный экземпляр класса this, который уже содержит контекст приложения.

Далее задаются параметры компоновки, путём создания экземпляра класса LinearLayout.LayoutParams и передачи его в метод setLayoutParams () следующим образом:

```
LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(
LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
layout.setLayoutParams (params);
```

Для задания способа расположения дочерних элементов в линейной компоновке (вертикально или горизонтально) есть метод setOrientation (), параметром которого служат две целочисленные константы, HORIZONTAL (0) и VERTICAL (1).

Дочерние виджеты, например, кнопки, текстовые поля создаются аналогично корневой компоновке. Константы, которые будут использоваться в качестве идентификаторов объявляются в классе Activity:

```
private static final int ID_BUTTON = 101;
```

Листинг 7. Динамическое создание элементов управления.

```
package com.example.dynamicui;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup.LayoutParams;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity implements
View.OnClickListener {
    private static final int ID_BUTTON1 = 101;
    private static final int ID_BUTTON2 = 102;
    private static final int ID_TEXT = 103;
    private TextView text;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        layout.setLayoutParams(new LinearLayout.LayoutParams(
                LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
        Button button1 = new Button(this);
        button1.setText("Кнопка 1");
        button1.setId(ID_BUTTON1);
        button1.setLayoutParams(new LinearLayout.LayoutParams(
                LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
        button1.setOnClickListener(this);
        Button button2 = new Button(this);
        button2.setText("Кнопка 2");
        button2.setId(ID_BUTTON2);
        button2.setLayoutParams(new LinearLayout.LayoutParams(
                LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
        button2.setOnClickListener(this);
        text = new TextView(this);
        text.setId(ID_TEXT);
        text.setText("Событие: ");
        text.setLayoutParams(new LinearLayout.LayoutParams(
                LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
        layout.addView(button1);
        layout.addView(button2);
        layout.addView(text);
        this.addContentView(layout, new LinearLayout.LayoutParams(
                LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
    }
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case ID_BUTTON1:
                text.setText("Нажатие на первую кнопку"); break;
            case ID_BUTTON2:
                text.setText("Нажатие на вторую кнопку"); break; }
    }
}
```

Задание

В листинге 8 частично приведен код MainActivity.java из приложения, реализующего простой редактор текста, позволяющий изменить стиль, размер, цвет и тип шрифта для текста. В компоновке используются элементы TextView, EditText, Button, ToggleButton, RadioButton, ImageButton. Приложение представлено на рисунке 6.

Изучите код, представленный в листинге 6 и на его основе разработайте аналогичное приложение. В представленном коде имеются пропущенные фрагменты, которые обозначены звездочками (***). Пропущено может быть произвольное количество строк.

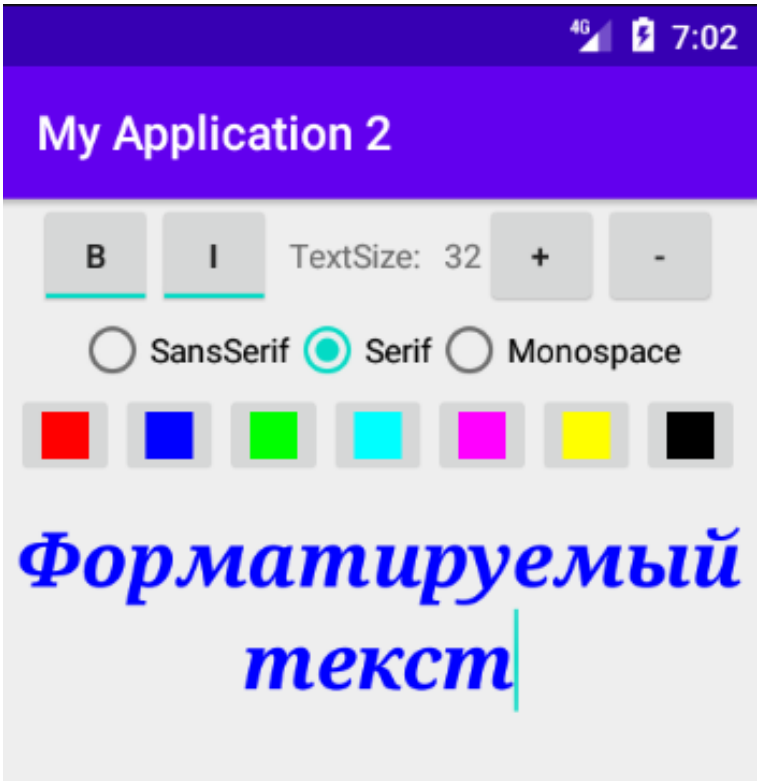

Рисунок 6. Простой текстовый редактор.

Листинг 8. Фрагменты кода приложения

```
package com.example.myapplication2;
import androidx.appcompat.app.AppCompatActivity;

import android.graphics.Color;
import android.os.Bundle;
import android.graphics.Typeface;
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.TextView;
***

public class MainActivity extends AppCompatActivity implements
OnClickListener {
    private float mTextSize = 20;
    private EditText mEdit;
    private TextView tSize;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mEdit =(EditText)findViewById(R.id.edit_text);
        tSize =(TextView)findViewById(R.id.size);
        ToggleButton buttonB = (ToggleButton)findViewById(R.id.button_b);
        ToggleButton buttonI = (ToggleButton)findViewById(R.id.button_i);
        Button buttonSans = (Button)findViewById(R.id.button_sans);
        Button buttonSer = (Button)findViewById(R.id.button_serif);
        Button buttonMono = (Button)findViewById(R.id.button_monospace);
        Button buttonPlus = (Button)findViewById(R.id.button_plus);
        Button buttonMinus = (Button)findViewById(R.id.button_minus);
        ImageButton bRed = (ImageButton)findViewById(R.id.red);
        ImageButton bGreen = (ImageButton)findViewById(R.id.green);
        ***
        buttonB.setOnClickListener(this);
        buttonI.setOnClickListener(this);
        buttonSans.setOnClickListener(this);
        buttonSer.setOnClickListener(this);
        ***
        bRed.setOnClickListener(this);
        ***
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button_plus:
                if (mTextSize < 72) {
                    mTextSize += 2;
                    mEdit.setTextSize(mTextSize);
                    tSize.setText(String.format("%.0f", mTextSize));
                }
                break;
            case R.id.button_minus:
                if (mTextSize > 18) {
                    mTextSize-=2;
                    ***
                }
                break;
            case R.id.button_b:
                if (mEdit.getTypeface().getStyle()== Typeface.ITALIC)
```

```
mEdit.setTypeface(mEdit.getTypeface(), Typeface.BOLD_ITALIC);
                else if (mEdit.getTypeface().getStyle()==
Typeface.BOLD_ITALIC) mEdit.setTypeface(mEdit.getTypeface(),
Typeface.ITALIC);
                else if (mEdit.getTypeface().getStyle()== Typeface.BOLD)
mEdit.setTypeface(Typeface.create(mEdit.getTypeface(), Typeface.NORMAL));
                else mEdit.setTypeface(mEdit.getTypeface(),
Typeface.BOLD);
                break;
            case R.id.button_i:
                ***
                break;
            case R.id.button_sans:
                if (mEdit.getTypeface().getStyle()== Typeface.ITALIC)
                    mEdit.setTypeface(Typeface.SANS_SERIF,
Typeface.ITALIC);
                else if (mEdit.getTypeface().getStyle()==
Typeface.BOLD_ITALIC) mEdit.setTypeface(Typeface.SANS_SERIF,
Typeface.BOLD_ITALIC);
                else if (mEdit.getTypeface().getStyle()== Typeface.BOLD)
mEdit.setTypeface(Typeface.SANS_SERIF, Typeface.BOLD);
                else mEdit.setTypeface(Typeface.SANS_SERIF,
Typeface.NORMAL);
                break;
            ***
            case R.id.cyan:
                mEdit.setTextColor(Color.parseColor("#00FFFF"));
                break;
            case R.id.magenta:
                mEdit.setTextColor(Color.parseColor("#FF00FF"));
                break;
            case R.id.yellow:
                mEdit.setTextColor(Color.parseColor("#FFFF00"));
                break;
            case R.id.black:
                mEdit.setTextColor(Color.parseColor("#000000"));
                break;
        }
    }
}
```

Контрольные вопросы:

1. Что такое Android Studio?
2. Что такое компоновка?
3. Какие типы компоновок Вы знаете?
4. В чём особенности компоновки LinearLayout?
5. В чём особенности компоновки ConstraintLayout?
6. В чём особенности компоновки RelativeLayout?

7. За что отвечает атрибут android:gravity?
8. За что отвечает атрибут android:id?
9. Что обозначает запись @+id в атрибуте android:id?
10. Какой метод содержит интерфейс View.OnClickListener?
11. Каково назначение метода findViewById?
12. Каково назначение метода setTypeface?
13. Каково назначение метода setTextSize?
14. Какой интерфейс необходимо реализовать для программного отслеживания изменения состояния элемента CheckBox?
15. В чём схожи и чем отличаются элементы CheckBox и RadioButton?
16. Каково назначение метода isChecked?
17. Каково назначение метода setOnClickListener?
18. Что такое ToggleButton?
19. Каково назначение метода onCheckedChanged?
20. Чем виджет ImageButton отличается от Button?
21. Каково назначение метода setImageResource?