|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

**Отчет по практической работе №3**

по дисциплине «Разработка мобильных приложений»

|  |  |
| --- | --- |
| **Выполнил:**  Студент группыИКБО-11-22 | Гришин А.В. |
| **Принял:**  Преподаватель | Степанов П.В. |

Москва 2024 г.

# **Изучите порядок выполнения приложения в проекте Android Studio, создание и применение графического интерфейса.**

**Режим кода и графический режим. Файл activity\_main.xml. Определение пространств имен XML. Текстовое поле TextView.**

Android Studio позволяет работать с визуальным интерфейсом как в режиме кода, так и в графическом режиме. Так, по умолчанию файл открыт в графическом режиме, и мы наглядно можем увидеть, как у нас примерно будет выглядеть экран приложения. И даже набросать с панели инструментов какие-нибудь элементы управления, например, кнопки или текстовые поля.

Android Studio позволяет работать с визуальным интерфейсом как в режиме кода, так и в графическом режиме. Так, по умолчанию файл открыт в графическом режиме, и мы наглядно можем увидеть, как у нас примерно будет выглядеть экран приложения. И даже набрасать с панели инструментов какие-нибудь элементы управления, например, кнопки или текстовые поля.

# TextView

Текстовое поле устанавливает текст с помощью атрибута android:text.

* + android:layout\_width устанавливает ширину виджета. Значение wrap\_content задает для виджета величину, достаточную для отображения в контейнере.
  + android:layout\_height устанавливает высоту виджета. Значение wrap\_content аналогично установке ширины задает для виджета высоту, достаточную для отображения в контейнере
  + android:text устанавливает текст, который будет выводиться в TextView (в данном случае это строка "Hello World!")
  + app:layout\_constraintLeft\_toLeftOf="parent": указывает, что левая граница элемента будет выравниваться по левой стороне контейнера ConstraintLayout

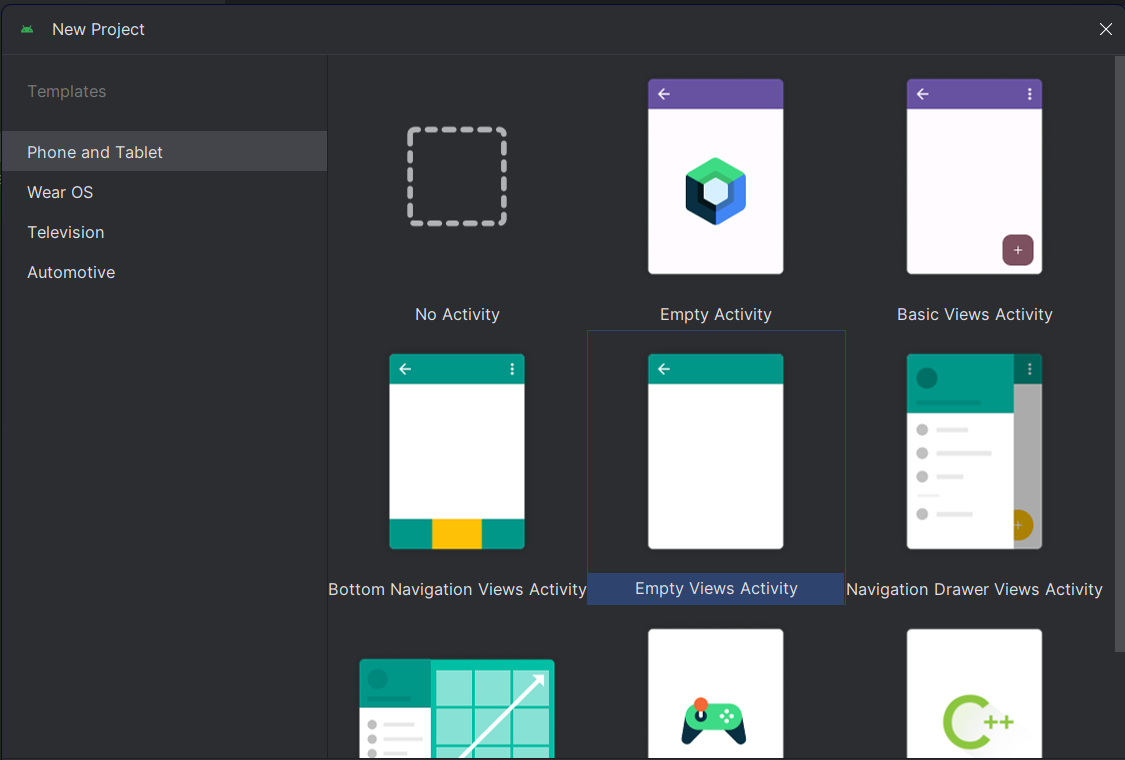
Обратите внимание, что этот атрибут определен в пространстве имен с префиксом app, то есть в ["h](http://schemas.android.com/apk/res-auto)t[tp://schemas.android.com/apk/res-auto](http://schemas.android.com/apk/res-auto)".

* + app:layout\_constraintTop\_toTopOf="parent": указывает, что верхняя граница элемента будет выравниваться по верхней стороне контейнера ConstraintLayout
  + app:layout\_constraintRight\_toRightOf="parent": указывает, что правая граница элемента будет выравниваться по правой стороне контейнера ConstraintLayout
  + app:layout\_constraintBottom\_toBottomOf="parent": указывает, что нижняя граница элемента будет выравниваться по нижней стороне контейнера ConstraintLayout

Стоит отметить, что последние четыре атрибута вместе будут приводить к расположению TextView по центру экрана.

Таким образом, при запуске приложения сначала запускается класс MainActivity, который в качестве графического интерфейса устанавливает разметку из файла activity\_main.xml. И поскольку в этой разметке прописан элемент TextView, который представляет некоторый текст, то мы и увидим его текст на экране смартфона.

# **Для работы с визуальными элементами создать новый проект. В качестве шаблона проекта выбераем Empty Activity.**



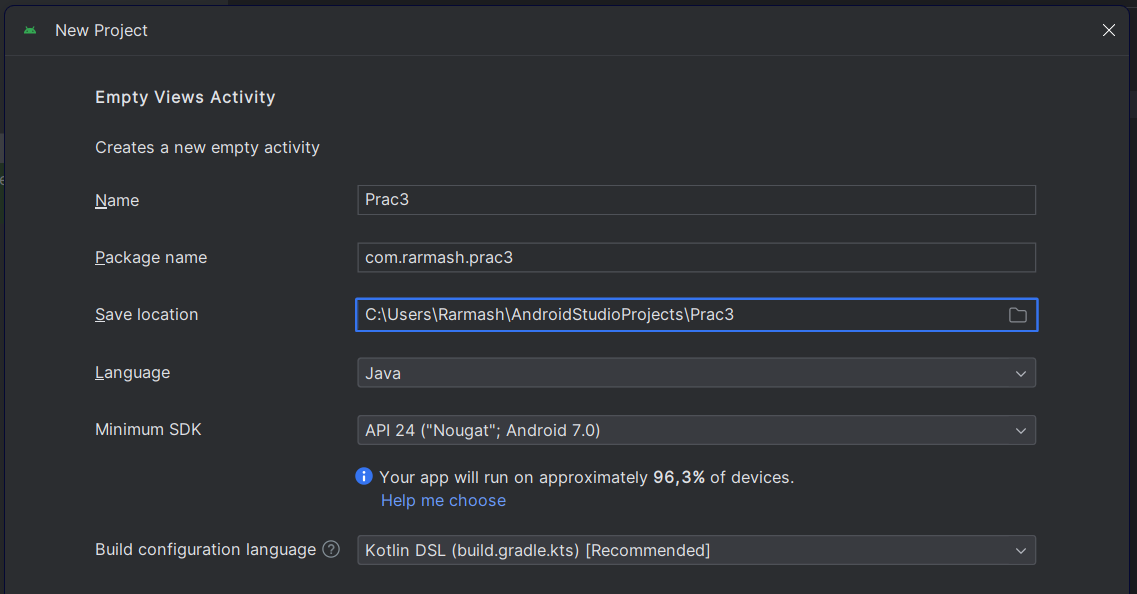


Рисунок 1 – Создание проекта

# **Создать интерфейс в коде java. (создать элементы управления программно в коде java)**

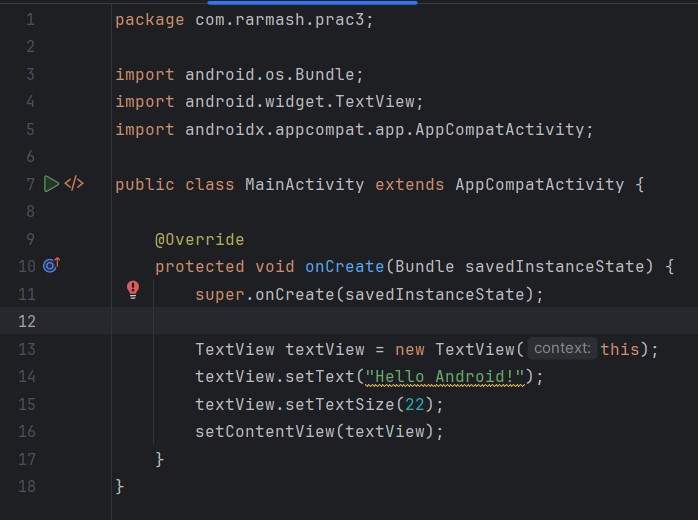


Рисунок 2 – Определение простейшего интерфейса

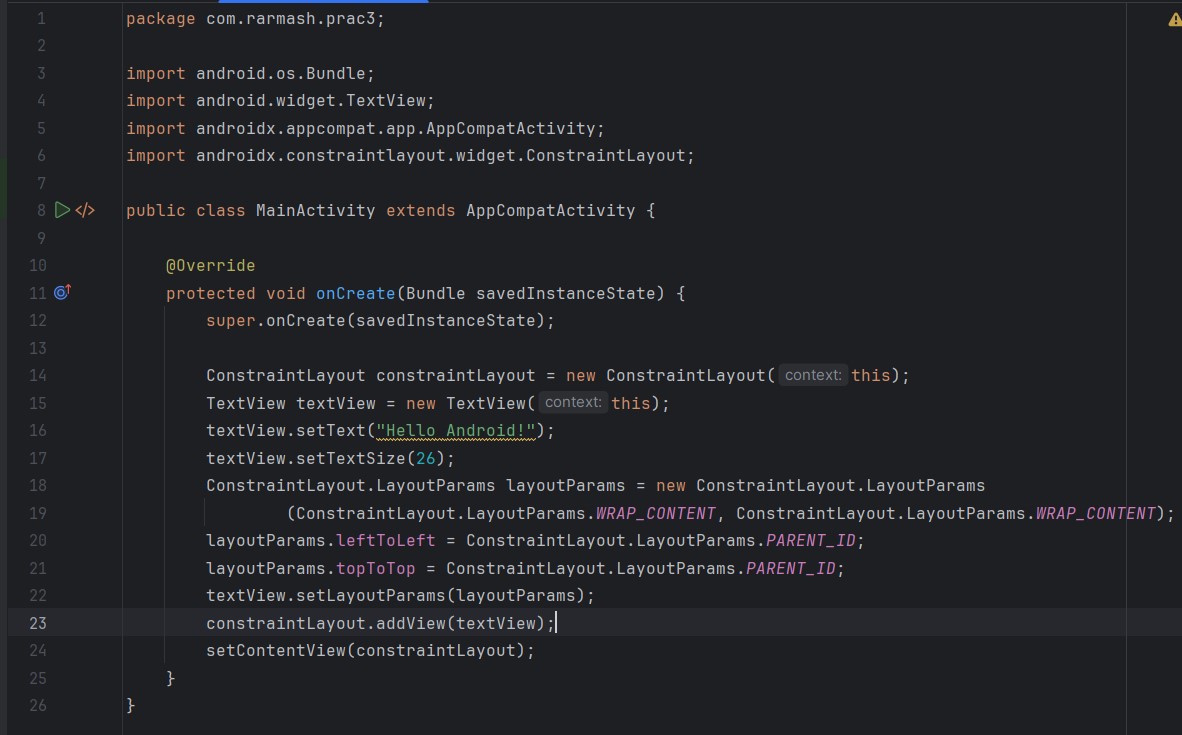


Рисунок 3 – TextView, вложенный в ConstraintLayout

# **Объявить элементы интерфейса в XML. Определить интерфейс в файле XML activity\_main.xml**

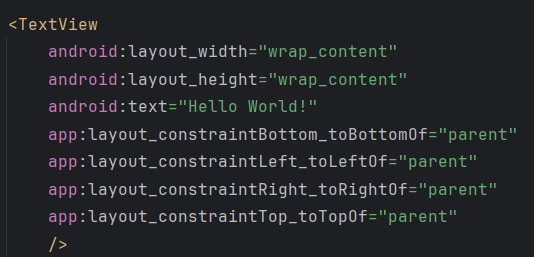


Рисунок 4 – Файл разметки по умолчанию

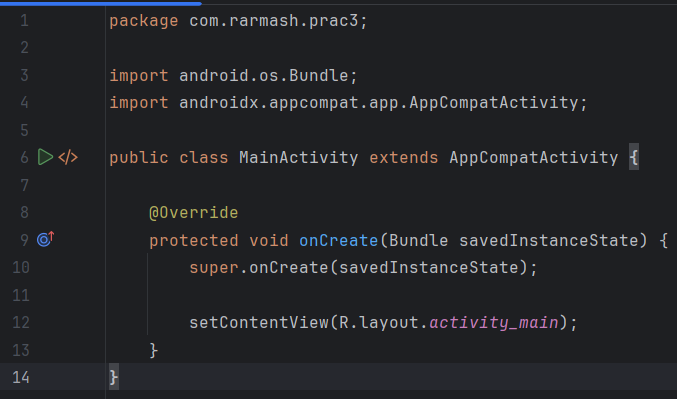


Рисунок 5 – Определение кода в классе

# **Добавить в проект новый файл разметки интерфейса**

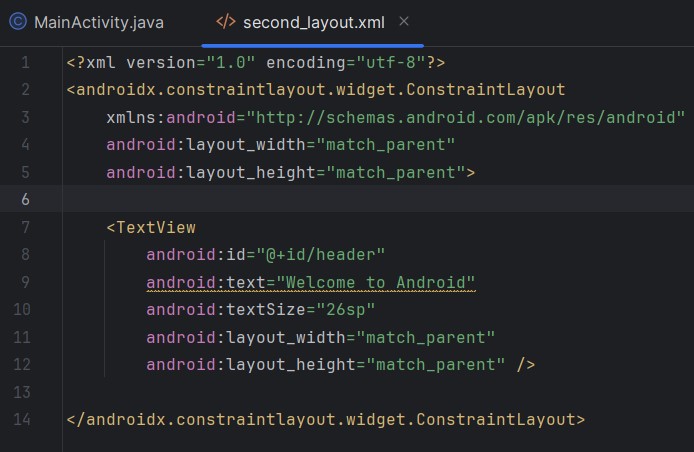


Рисунок 6 – Добавлен новый файл разметки

# **Реализовать получение и управление визуальными элементами в коде.**

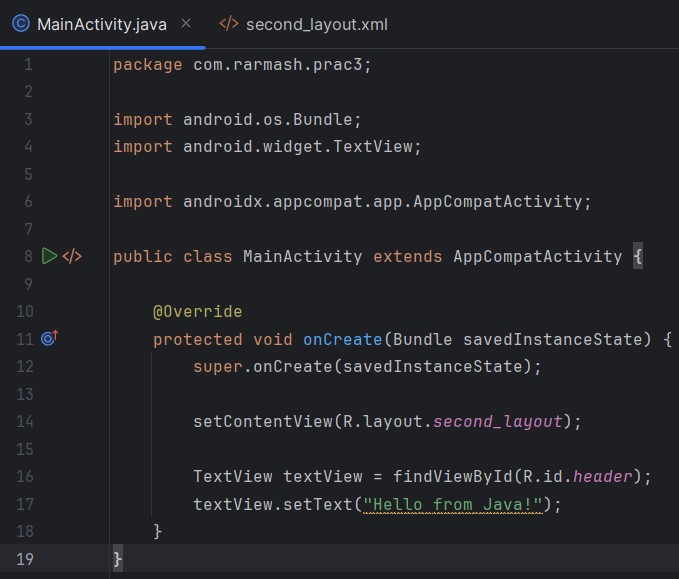


Рисунок 7 – Изменение кода

# **Определить размеры экрана вашего устройства на Android.**

При разработке приложений под Android мы можем использовать различные типы измерений:

* px: пиксели текущего экрана. Однако эта единица измерения не рекомендуется, так как реальное представление внешнего вида может изменяться в зависимости от устройства; каждое устройство имеет определенный набор пикселей на дюйм, поэтому количество пикселей на экране может также меняться
* dp: (device-independent pixels) независимые от плотности экрана пиксели.

Абстрактная единица измерения, основанная на физической плотности экрана с разрешением 160 dpi (точек на дюйм). В этом случае 1dp = 1px. Если размер экрана больше или меньше, чем 160dpi, количество пикселей, которые применяются для отрисовки 1dp соответственно увеличивается или уменьшается. Например, на экране с 240 dpi 1dp=1,5px, а на экране с 320dpi 1dp=2px. Общая формула для получения количества физических пикселей из dp: px = dp \* (dpi / 160)

* sp: (scale-independent pixels) независимые от масштабирования пиксели. Допускают настройку размеров, производимую пользователем. Рекомендуются для работы со шрифтами.
* pt: 1/72 дюйма, базируются на физических размерах экрана
* mm: миллиметры
* in: дюймы

# **Установить размеры в коде Java.**

Основная проблема, связанная с размерами, связана с их установкой в коде Java. Например, некоторые методы принимают в качестве значения физические пиксели, а не device-independent pixels. В этом случае может потребоваться перевести значения из одного типа единиц в другой.

Параметр unit представляет тип единиц, из которой надо получить значение в пикселях. Тип единиц описывается одной из констант TypedValue:

* + COMPLEX\_UNIT\_DIP - dp или независимые от плотности экрана пиксели
  + COMPLEX\_UNIT\_IN - in или дюймы
  + COMPLEX\_UNIT\_MM - mm или миллиметры
  + COMPLEX\_UNIT\_PT - pt или точки
  + COMPLEX\_UNIT\_PX - px или физические пиксели
  + COMPLEX\_UNIT\_SP - sp или независимые от масштабирования пиксели (scale-independent pixels)

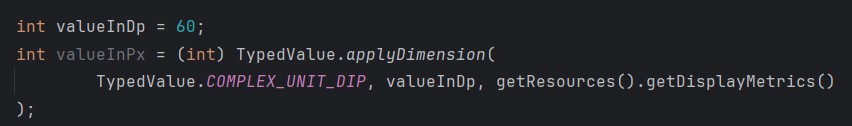


Рисунок 8 – Получение пикселей

# **Установить ширину и высоту визуальных элементов в XML ( match\_parent, wrap\_content)**

Все визуальные элеметы, которые мы используем в приложении, как правило, упорядочиваются на экране с помощью контейнеров. В Android подобными контейнерами служат такие классы как RelativeLayout, LinearLayout, GridLayout, TableLayout, ConstraintLayout, FrameLayout. Все они по разному располагают элементы и управляют ими, но есть некоторые общие моменты при компоновке визуальных компонентов, которые мы сейчас рассмотрим.

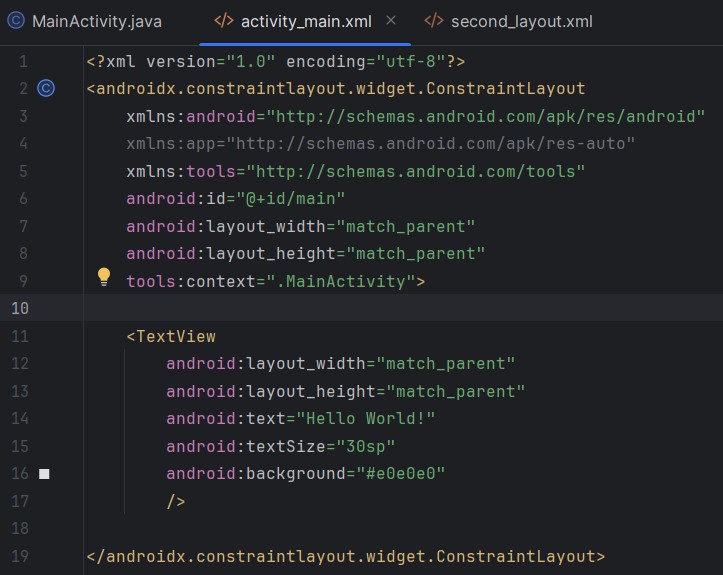


Рисунок 9 – Использование match\_parent

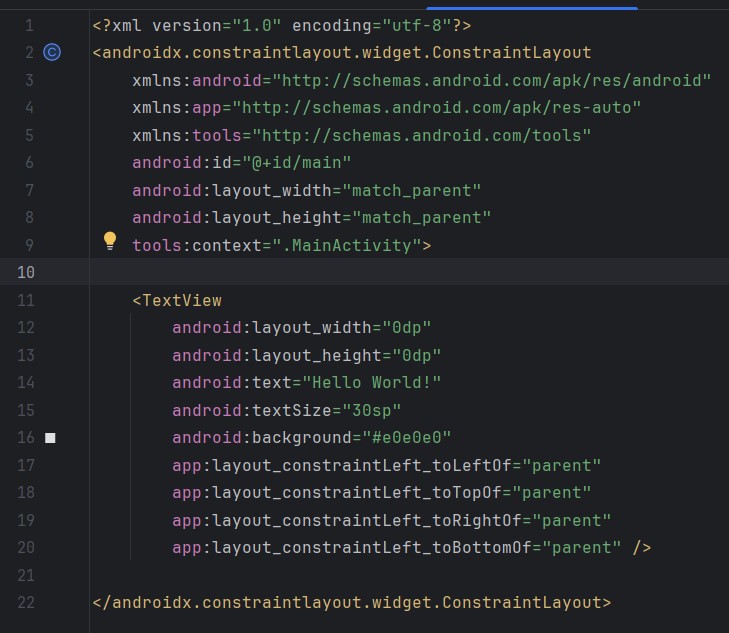


Рисунок 10 – Использование 0dp

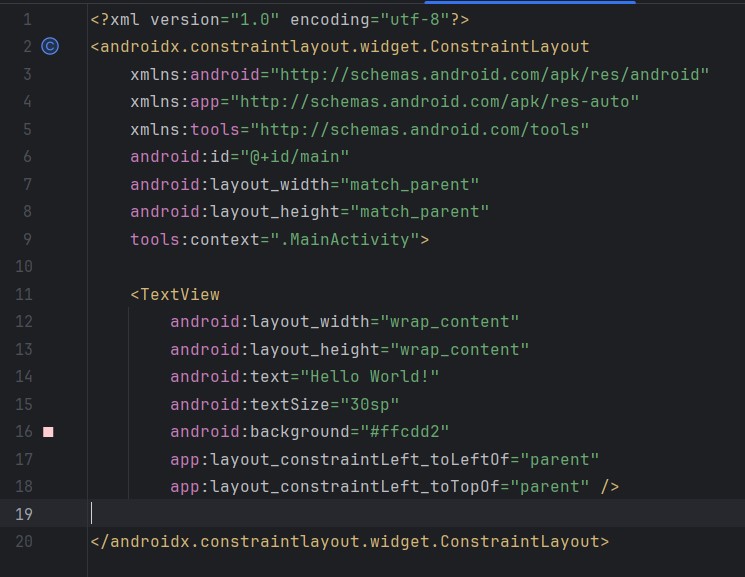


Рисунок 11 – Использование wrap\_content

# **Реализовать программную установку ширины и высоты**

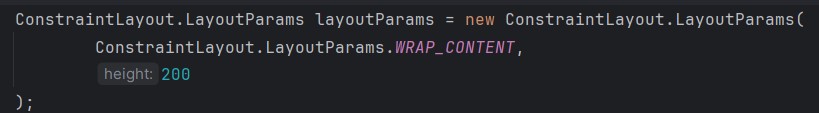


Рисунок 12 – Установка высоты и ширины с помощью кода

# **Реализовать внутренние и внешние отступы в XML (Padding, Margin)**

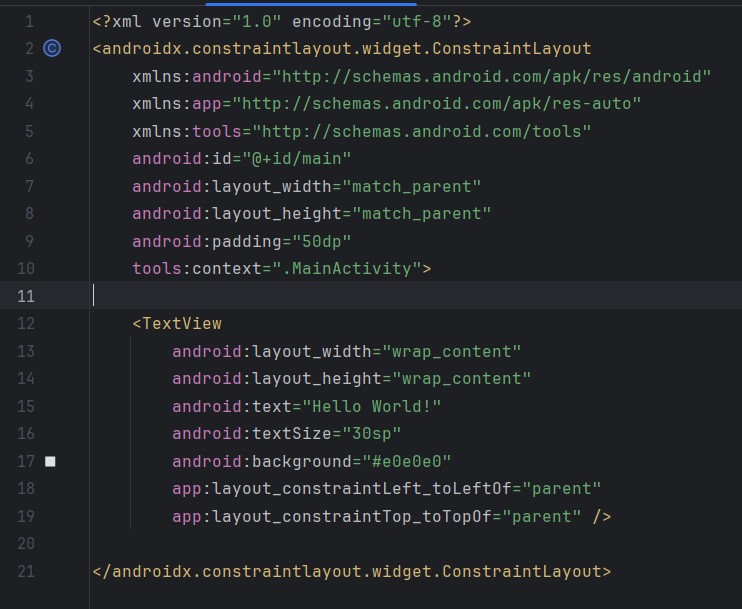


Рисунок 13 – Применение Padding

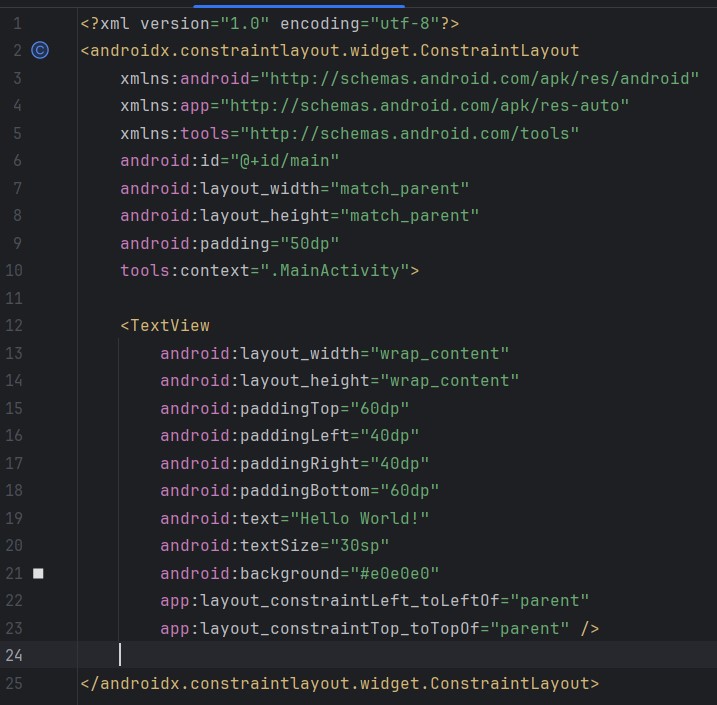


Рисунок 14 – Установка точных отступов

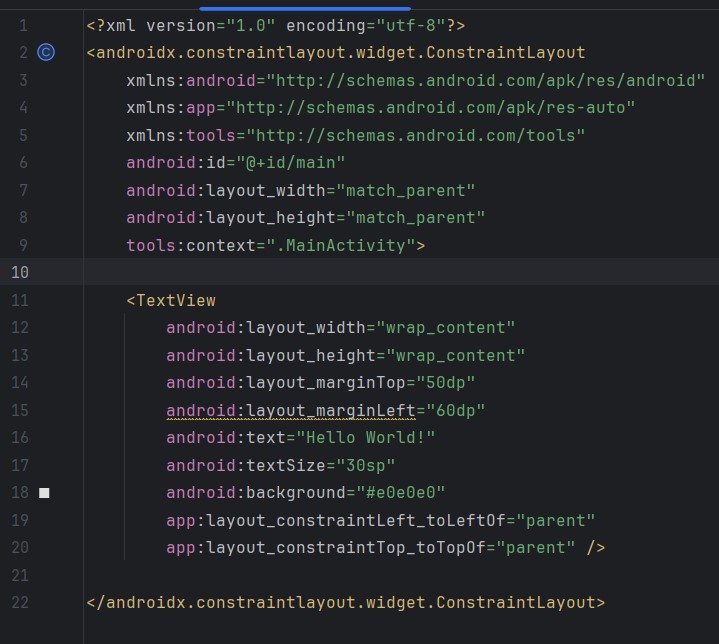


Рисунок 15 – Применение Margin

# **Реализовать программную установку внутренних и внешних отступов**

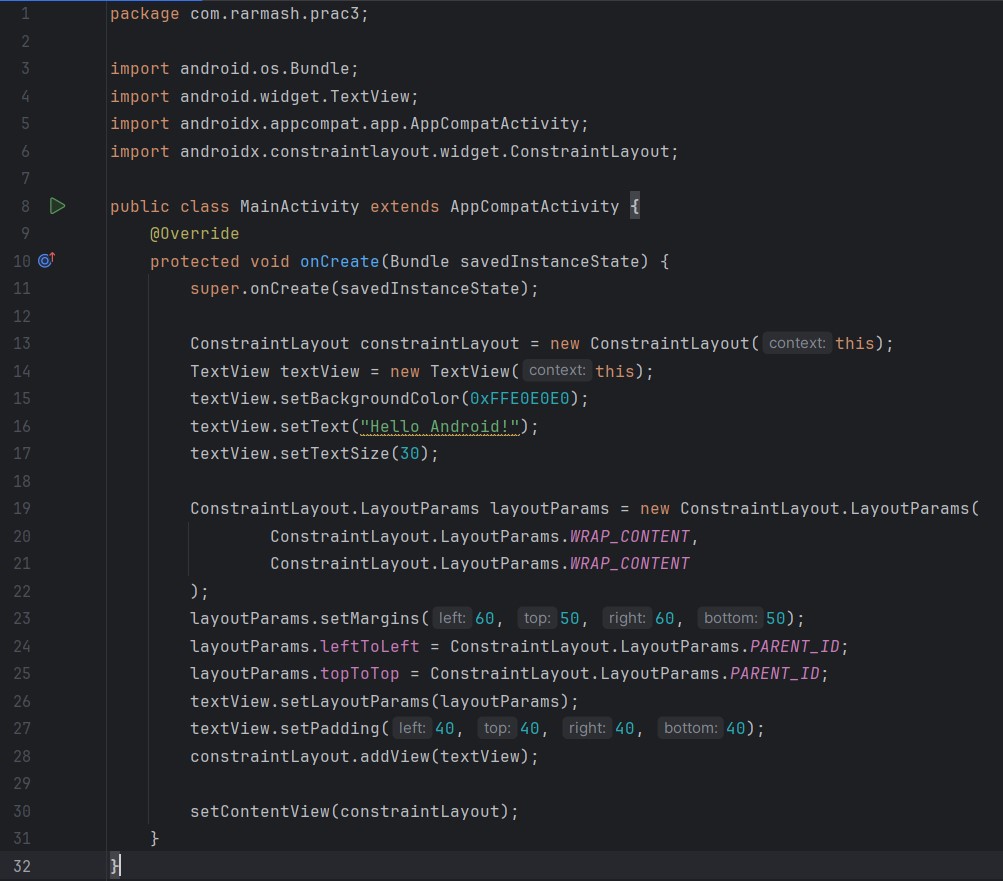
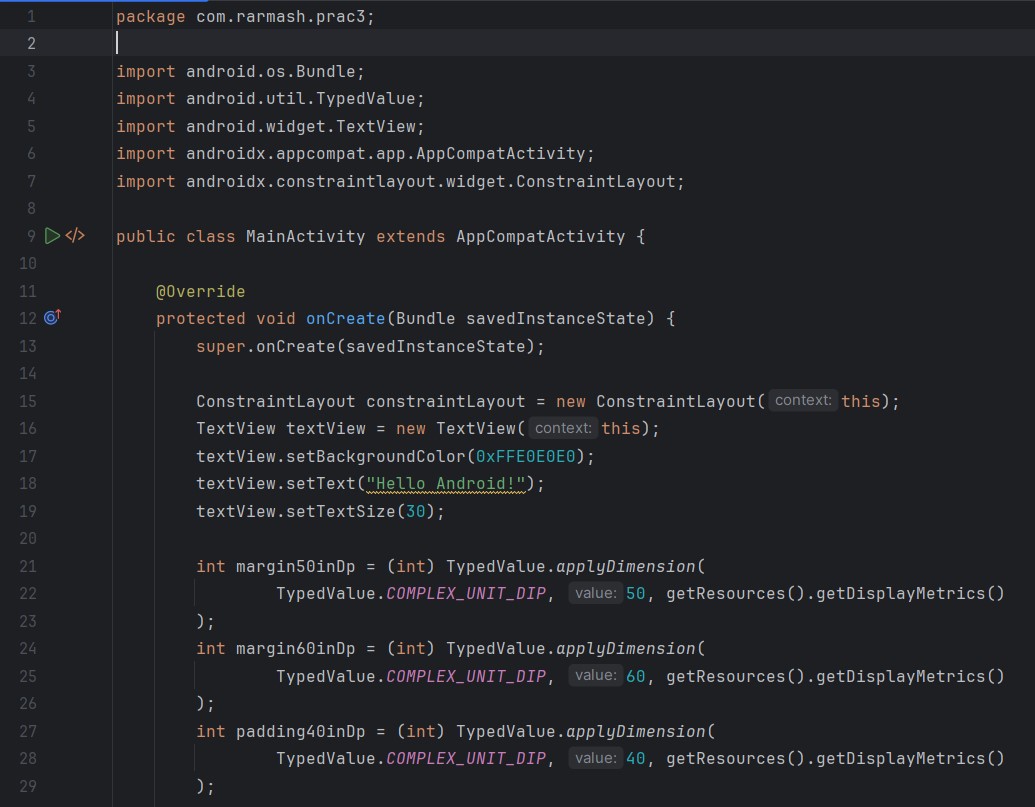


Рисунок 16 – Программная установка отступов



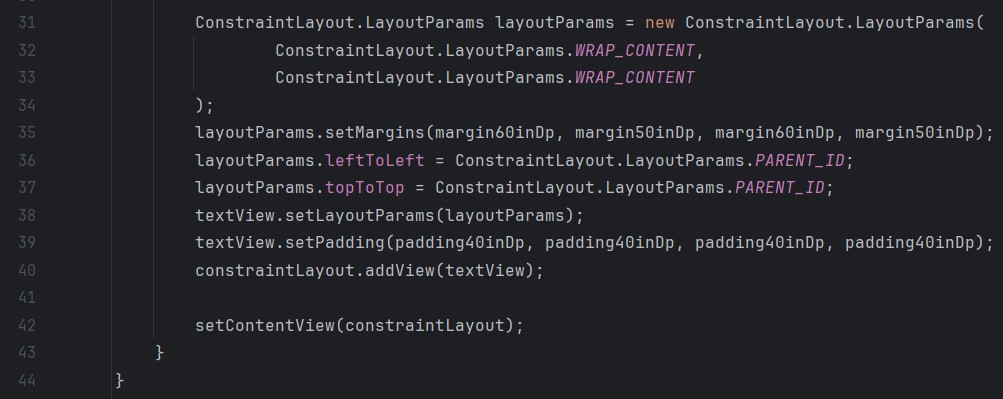


Рисунок 17 – Использование dp в коде

# **Реализуйте примеры применения контейнера ConstraintLayout. (позиционирование, сдвиг, горизонтальная цепочка элементов, вес элемента, горизонтальный ряд виджетов, вертикальная цепочка, программное создание ConstraintLayout)**

ConstraintLayout представляет контейнер, который позволяет создавать гибкие и масштабируемые визуальные интерфейсы.

Для позиционирования элемента внутри ConstraintLayout необходимо указать ограничения (constraints). Есть несколько типов ограничений. В частности, для установки позиции относительно определенного элемента испльзуются следующие ограничения:

* layout\_constraintLeft\_toLeftOf: левая граница позиционируется относительно левой границы другого элемента
* layout\_constraintLeft\_toRightOf: левая граница позиционируется относительно правой границы другого элемента
* layout\_constraintRight\_toLeftOf: правая граница позиционируется относительно левой границы другого элемента
* layout\_constraintRight\_toRightOf: правая граница позиционируется относительно правой границы другого элемента
* layout\_constraintTop\_toTopOf: верхняя граница позиционируется относительно верхней границы другого элемента
* layout\_constraintTop\_toBottomOf: верхняя граница позиционируется относительно нижней границы другого элемента
* layout\_constraintBottom\_toBottomOf: нижняя граница позиционируется относительно нижней границы другого элемента
* layout\_constraintBottom\_toTopOf: нижняя граница позиционируется относительно верхней границы другого элемента
* layout\_constraintBaseline\_toBaselineOf: базовая линия позиционируется относительно базовой линии другого элемента
* layout\_constraintStart\_toEndOf: элемент начинается там, где завершается другой элемент
* layout\_constraintStart\_toStartOf: элемент начинается там, где начинается другой элемент layout\_constraintEnd\_toStartOf: элемент завершается там, где начинается другой элемент
* layout\_constraintEnd\_toEndOf: элемент завершается там, где завершается другой элемент

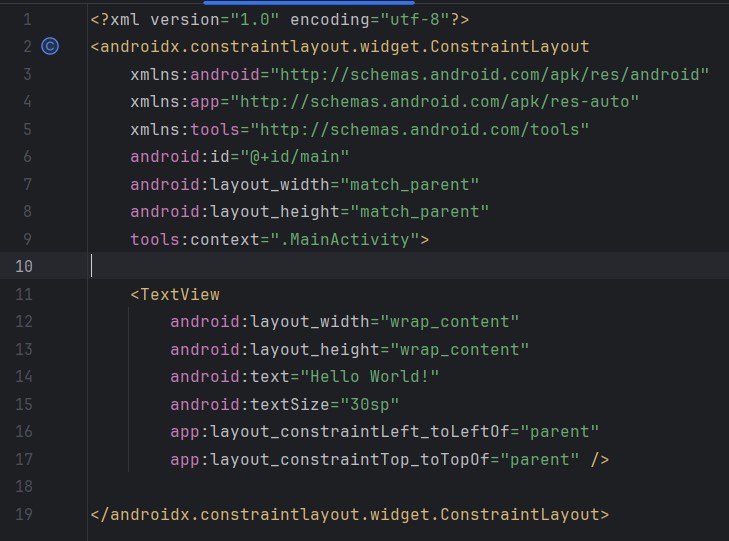


Рисунок 18 – Простейший пример



Рисунок 19 – Применение сдвига

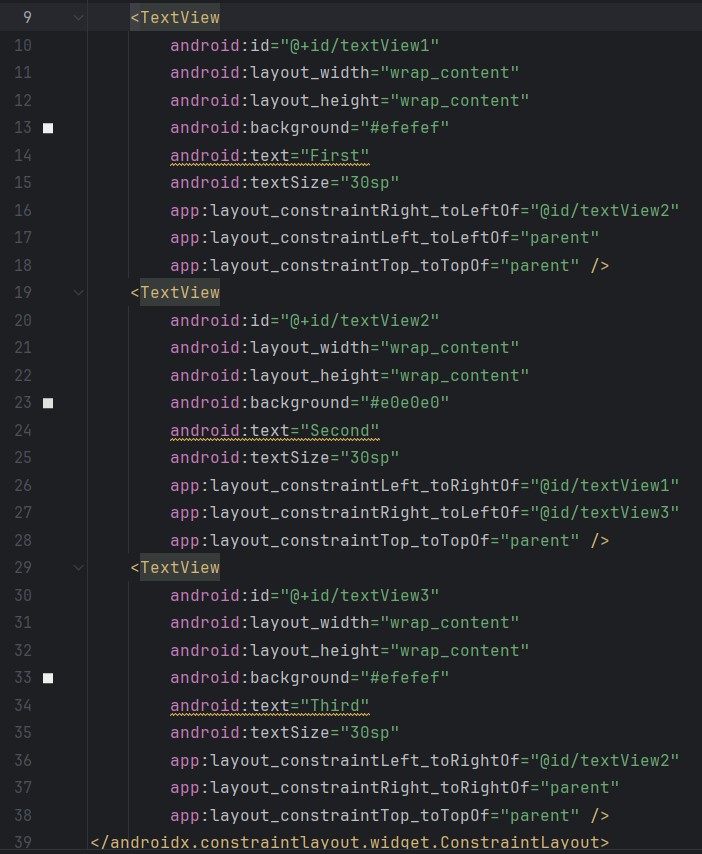


Рисунок 20 – Горизонтальная цепочка элементов

Кроме того, ConstraintLayout позволяет настроить положение элементов внутри цепочки. Для этого применяется атрибут layout\_constraintHorizontal\_chainStyle, который может принимать следующие значения:

* spread: значение по умолчанию, при котором элементы цепочки равномерно растягиваются по всей длине цепочки, как в примере выше
* spread\_inside: первый и последний элемент цепочки примыкают к границами контейнера
* packed: элементы цепочки располагаются вплотную друг к другу.



Рисунок 21 – Применение spread\_inside



Рисунок 22 – Применение packed



Рисунок 23 – Установка веса элементов



Рисунок 24 – Один элемент в 2 раза больше другого



Рисунок 25 – Вертикальная цепочка

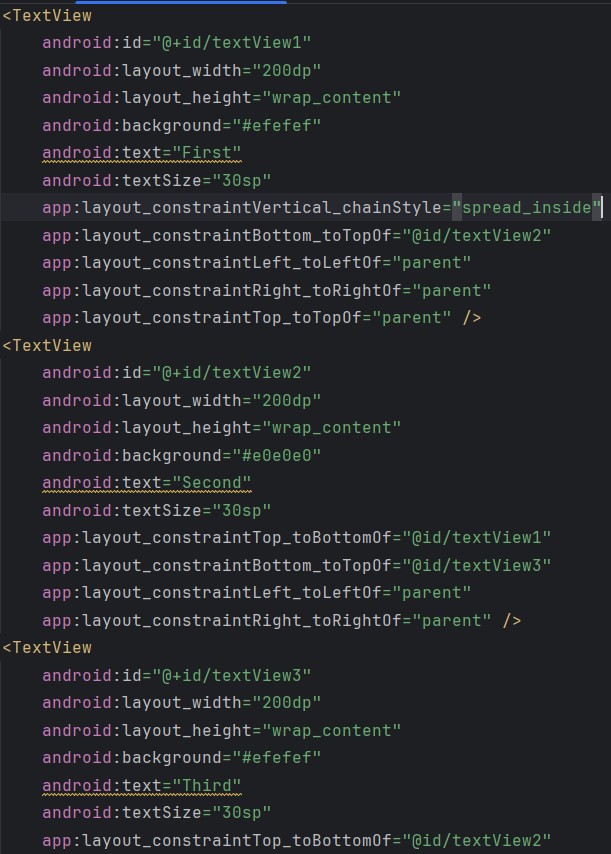


Рисунок 26 – Применение layout\_constraintVertical\_chainStyle



Рисунок 27 – Использование веса по вертикали

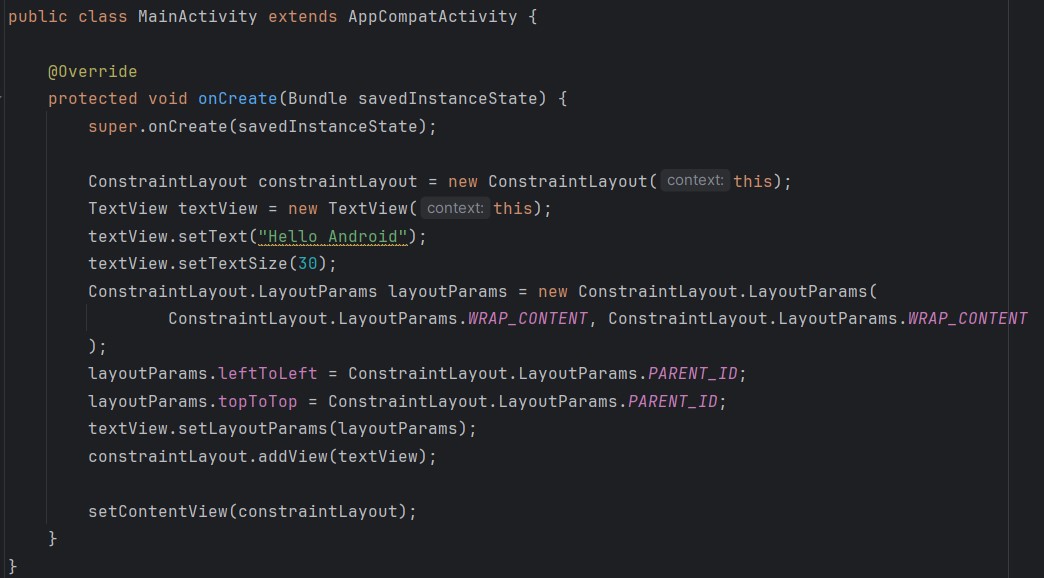


Рисунок 28 – Программное создание ConstraintLayout

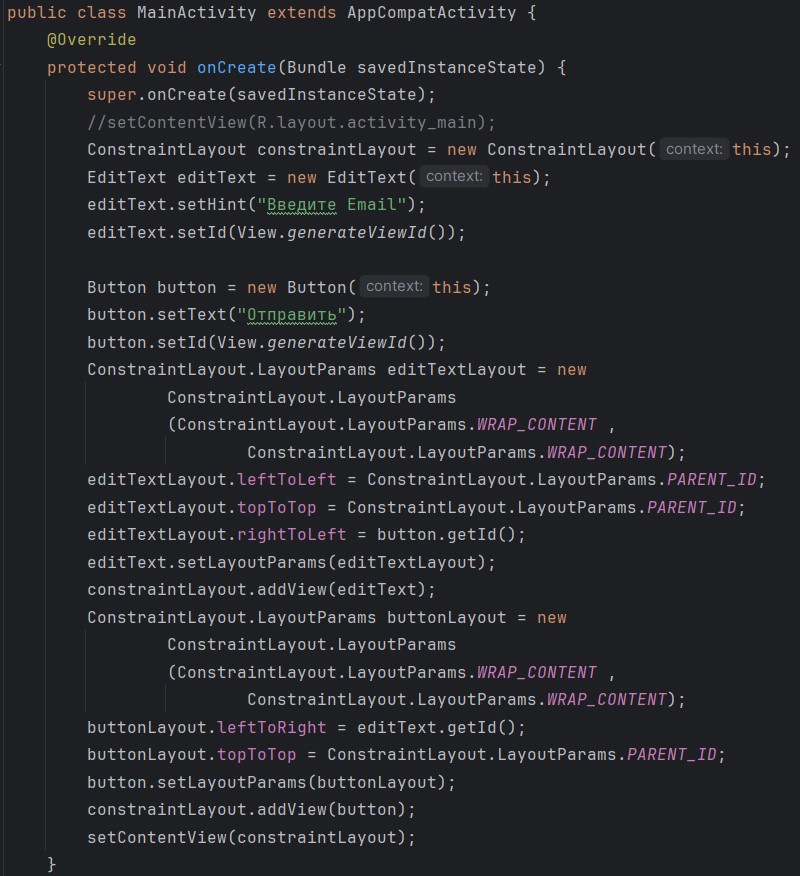


Рисунок 29 – Установка расположения элементов относительно друг друга

# **Ссылка на мой GitHub**

Репозиторий – «RMP\_prac3»

Ссылка - https://github.com/Rarmash/RMP\_prac3