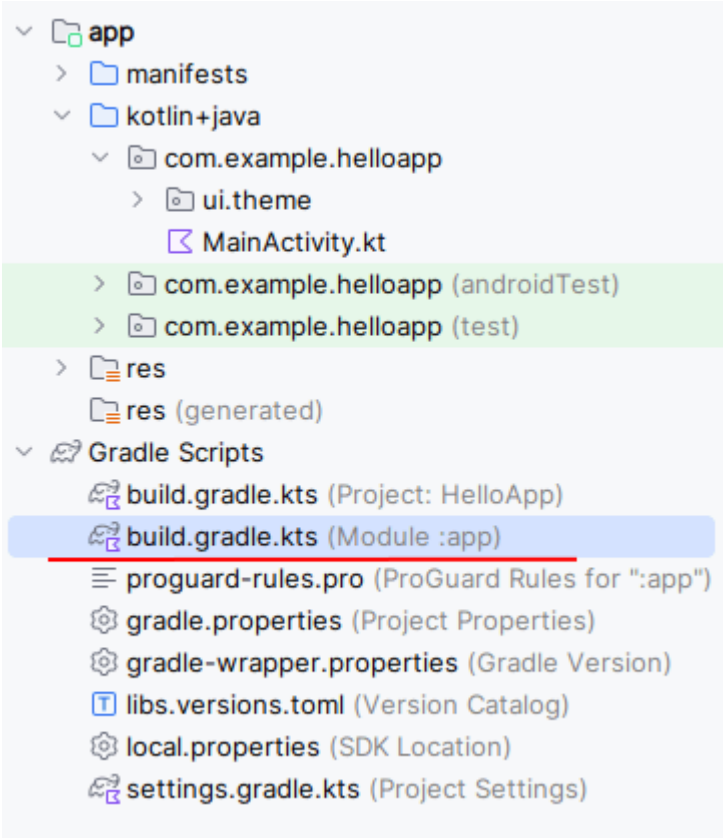
ConstraintLayout

Подключение ConstraintLayout

Элемент ConstraintLayout представляет контейнер, который позволяет размещать одни компоненты относительно других компонентов. ConstraintLayout предоставляет альтернативу использования нескольких вложенных друг в друга компонентов Column, Row и Box и может применяться при реализации более крупных макетов с более сложными требованиями к выравниванию.

Для использования ConstraintLayout в Compose необходимо добавить соответствующую зависимость в файл build.gradle. Так, откроем файйл build.gradle.kts (Module :app)



В этом файле где-то внизу найдем секцию dependencies, которая хранит добавленные в проект зависимости и которая выглядит примерно следующим образом:

```
dependencies {

    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.ui)
    implementation(libs.androidx.ui.graphics)
    implementation(libs.androidx.ui.tooling.preview)
    implementation(libs.androidx.material3)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
```

```
    androidTestImplementation(libs.androidx.espresso.core)
    androidTestImplementation(platform(libs.androidx.compose.bom))
    androidTestImplementation(libs.androidx.ui.test.junit4)
    debugImplementation(libs.androidx.ui.tooling)
    debugImplementation(libs.androidx.ui.test.manifest)
}
```

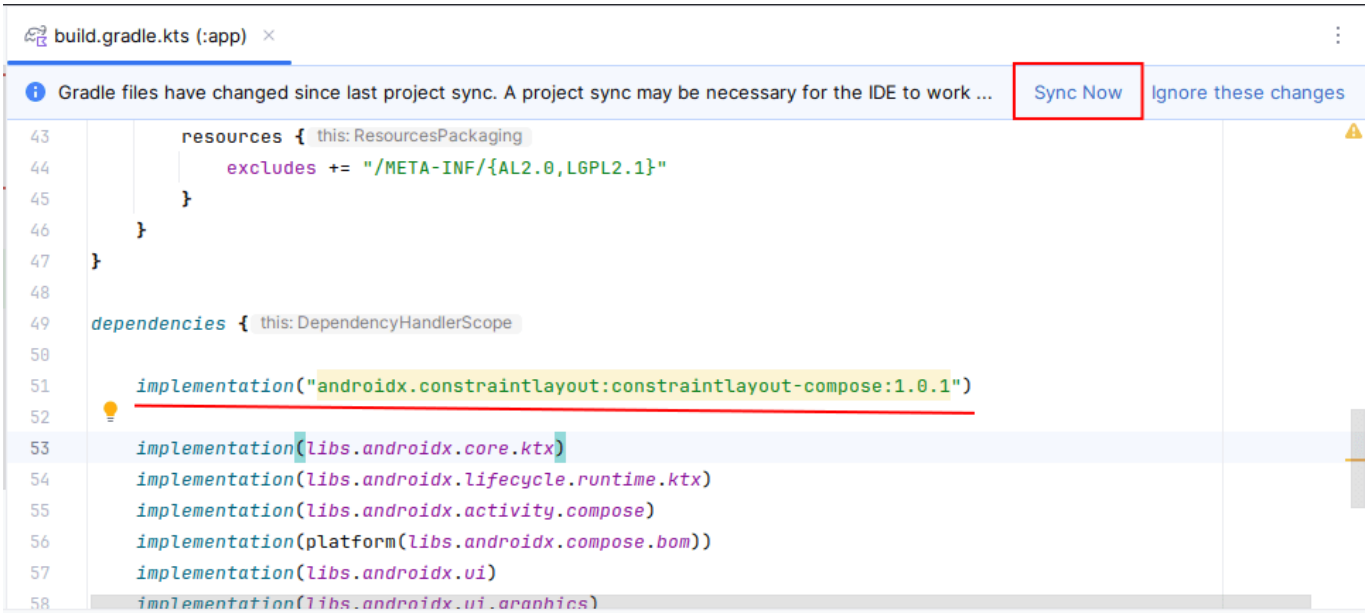
В самое начало этой секции добавим строку

```
implementation("androidx.constraintlayout:constraintlayout-compose:1.0.1")
```

То есть в итоге получится

```
dependencies {

    implementation("androidx.constraintlayout:constraintlayout-compose:1.0.1")

    implementation(libs.androidx.core.ktx)
    // остальные зависимости
    ....................................

}
```

Затем синхронизируем проект, нажав на кнопку Sync Now



После добавления зависимости мы можем использовать ConstraintLayout.

ConstraintLayout предоставляется в виде компонента аналогично другим компонентам в Compose:

```
@Composable
public inline fun ConstraintLayout(
    modifier: Modifier = COMPILED_CODE,
    optimizationLevel: Int = COMPILED_CODE,
    crossinline content: @Composable() (ConstraintLayoutScope.() -> Unit)
): Unit
```

Первый параметр представляет функции-модификаторы, применяемые к контейнеру. Второй параметр задает уровень оптимизации. И третий параметр задает внутренне содержимое. Соответственно мы можем вызывать ConstraintLayout следующим образом:

```
ConstraintLayout {

    // здесь вложенные компоненты

}
```

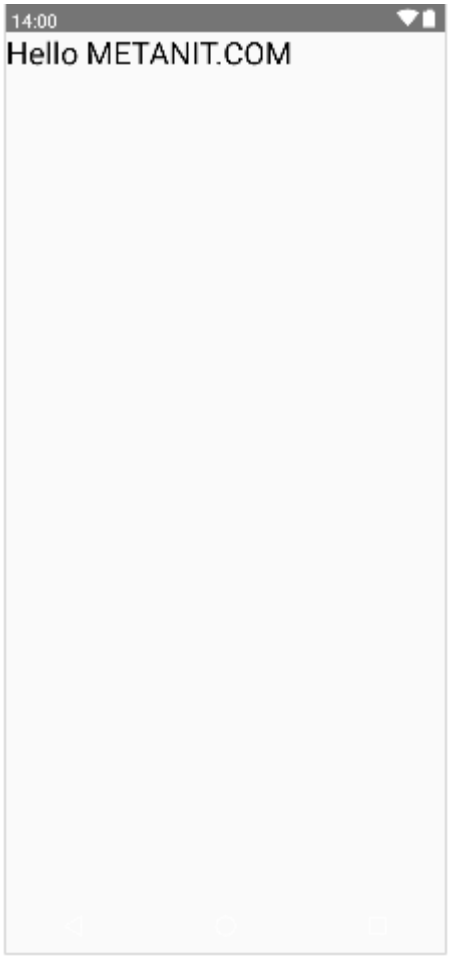
Можем передать в @Composable ConstraintLayout какие-нибудь модификаторы

```
ConstraintLayout(Modifier.size(300.dp).background(Color.DarkGray)) {

    // здесь вложенные компоненты

}
```

По умолчанию дочерний компонент позиционируется в ConstraintLayout в верхнем левом углу (при левосторонней ориентации). Например, добавим в ConstraintLayout текст:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material3.Text
import androidx.compose.ui.unit.sp
import androidx.constraintlayout.compose.ConstraintLayout

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout() {
                Text("Hello METANIT.COM", fontSize = 28.sp)
            }
        }
```

```
    }
}
```



Установка ограничений в ConstraintLayout

Хотя мы можем так делать, как в примере выше, но в этом смысла нет, поскольку в этом случае проще воспользоваться стандартными типами контейнеров типа Box. Ключевой момент ConstraintLayout заключается именно в установке ограничений для вложенных компонентов. Наиболее распространенная форма ограничения — это ограничения относительно родительского контейнера ConstraintLayout или другого компонента.

Для установки для компонента ограничений этому компоненту необходимо назначить ссылку. Назначение ссылки представляет двухэтапный процесс, который состоит из создания ссылки и последующего ее назначения компонентам перед применением ограничений. Ссылка представляет объект класса ConstrainedLayoutReference, и одну ссылку можно создать с помощью вызова функции createRef() и затем присвоить результат константе:

```
val box1 = createRef()
```

Для создания нескольких ссылок можно использовать функцию createRefs():

```
val (box1, text1, text2) = createRefs()
```

После создания ссылок они применяются к отдельным компонентам с помощью функции-модификатора constrainAs():

```
Modifier.constrainAs(
    ref: ConstrainedLayoutReference,
    constrainBlock: ConstrainScope.() -> Unit
)
```

Этот модификатор принимает два параметра:

- Первый параметр - это созданная ссылка в виде объекта ConstrainedLayoutReference
- Второй параметр представляет функцию, которая устанавливает ограничения.

Например, следующий код присваивает ссылку box1 компоненту Box:

```
ConstraintLayout {
    val box1 = createRef()
    Box(modifier = Modifier.constrainAs(box1) {
        // здесь идут ограничения
    })
}
```

Здесь в компоненте Box модификатору constrainAs() передается ссылка "box1". Таким образом, далее мы сможем ссылаться на этот компонент Box через ссылку box1.

ConstrainScope

Второй параметр модификатора constrainAs() представляет функцию, в которой и устанавливаются ограничения. Для этого внутри данной функции можно использовать свойства класса ConstrainScope/ConstrainedLayoutReference:

- absoluteLeft: левый край компонента
- absoluteRight: правый край компонента
- baseline: базовая линия компонента
- bottom: нижний край компонента
- top: верхний край компонента
- start: начало компонента (правый или левый крайв зависимости от направления текста)
- end: конец компонента (правый или левый крайв зависимости от направления текста)

Кроме того, ConstrainScope предоставляет специальное свойство parent, который представляет ссылку на родительский контейнер в виде объекта ConstrainedLayoutReference и через который можно обратиться к этим же свойствам родительского контейнера.

Одним из способов установить ограничения представляет вызов функции linkTo(). Она вызывается у выше рассмотренных свойств. Например:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout
import androidx.compose.ui.tooling.preview.Preview

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout {
                val rect = createRef()

Box(Modifier.size(200.dp).background(Color.DarkGray).constrainAs(rect) {
                    top.linkTo(parent.top, margin = 16.dp)
                })
            }
        }
    }
}
```

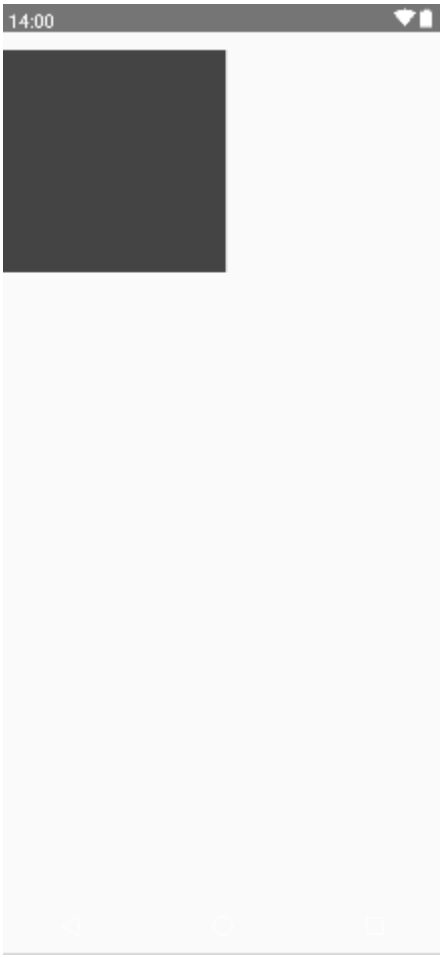
В данном случае строка

```
top.linkTo(parent.top, margin = 16.dp)
```

представляет установку ограничения для верхнего края компонента Box (свойство top). Соответственно выражение top.linkTo() говорит установить ограничение для верхнего края компонента Box.

Настройки ограничения передаются в вызов функции. Первый аргумент - относительно какой стороны надо установить ограничение, а второй аргумент - отступ. В данном случае первый аргумент - parent.top указывает на верхний край родительского контейнера, а второй аргумент - margin = 16.dp

задает отступ в 16 пикселей. Таким образом, этот код ограничивает верхний край компонента Box верхним краем родительского экземпляра ConstraintLayout с отступом в 16 пикселей:



Подобным образом можно установить и другие ограничения:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout(Modifier.fillMaxSize()) {
                val box1 = createRef()
```

```
Box(Modifier.size(200.dp).background(Color.DarkGray).constrainAs(box1) {
                    absoluteLeft.linkTo(parent.absoluteLeft, margin = 16.dp) //
отступ слева
                    top.linkTo(parent.top, margin = 16.dp)      // отступ сверху
                })
            }
        }
    }
}
```

Для установки ограничений ConstrainScope также предоставляет другую форму функции linkTo(), которая может передавать несколько ограничений в качестве параметров. Однако из ее форм:

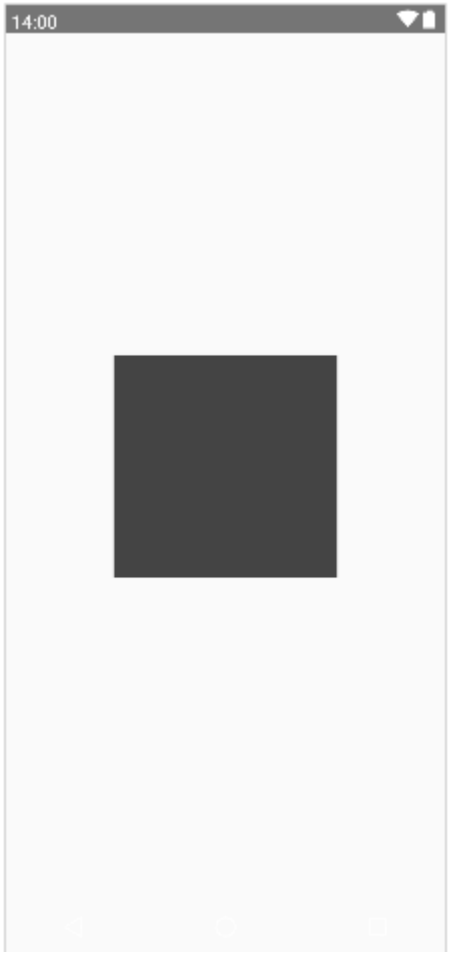
```
fun linkTo(
    start: ConstraintLayoutBaseScope.VerticalAnchor,
    top: ConstraintLayoutBaseScope.HorizontalAnchor,
    end: ConstraintLayoutBaseScope.VerticalAnchor,
    bottom: ConstraintLayoutBaseScope.HorizontalAnchor,
    startMargin: Dp = 0.dp,
    topMargin: Dp = 0.dp,
    endMargin: Dp = 0.dp,
    bottomMargin: Dp = 0.dp,
    startGoneMargin: Dp = 0.dp,
    topGoneMargin: Dp = 0.dp,
    endGoneMargin: Dp = 0.dp,
    bottomGoneMargin: Dp = 0.dp,
    horizontalBias: @FloatRange(from = 0.0, to = 1.0) Float = 0.5f,
    verticalBias: @FloatRange(from = 0.0, to = 1.0) Float = 0.5f
): Unit
```

Первый 4 параметра устанавливаются ограничения для соответственно сторон start, top, end, bottom текущего компонента, а следующие 4 параметра настраивают отступы.

Пример применения:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
```

```
import androidx.constraintlayout.compose.ConstraintLayout

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout(Modifier.fillMaxSize()) {
                val box1 = createRef()

Box(Modifier.size(200.dp).background(Color.DarkGray).constrainAs(box1) {
                    linkTo(start=parent.start,top=parent.top, end = parent.end,
bottom=parent.bottom, 16.dp, 16.dp, 16.dp, 16.dp)
                })
            }
        }
    }
}
```



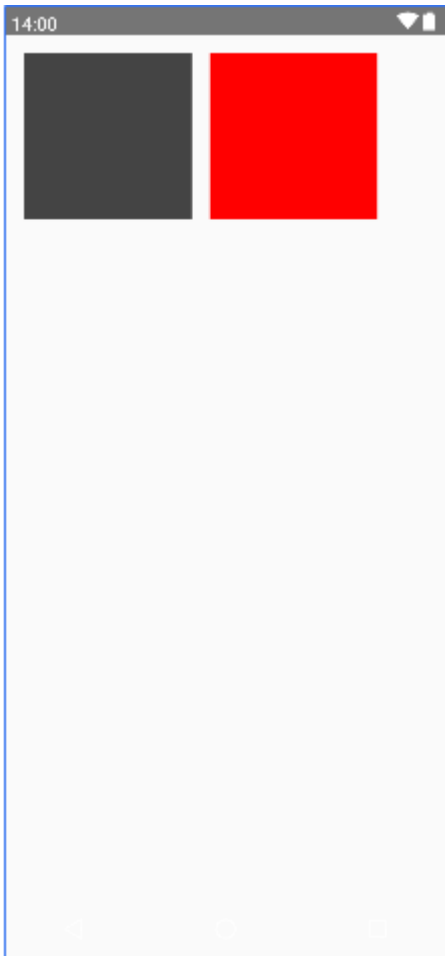
Ограничения относительно другого компонента

Аналогичным образом можно устанавливать ограничения относительно других компонентов:

```
package com.example.helloapp
```

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout(Modifier.fillMaxSize()) {
                val box1 = createRef()
                val box2 = createRef()

Box(Modifier.size(150.dp).background(Color.DarkGray).constrainAs(box1) {
                    absoluteLeft.linkTo(parent.absoluteLeft, margin = 16.dp) //
отступ слева
                    top.linkTo(parent.top, margin = 16.dp) // отступ сверху
                })

                Box(Modifier.size(150.dp).background(Color.Red).constrainAs(box2)
{
                    absoluteLeft.linkTo(box1.absoluteRight, margin = 16.dp) //
отступ слева от box1
                    top.linkTo(parent.top, margin = 16.dp) // отступ сверху
                })
            }
        }
    }
}
```

Здесь создается две ссылки - box1 и box2 для двух компонентов Box. Компонент box1 устанавливает ограничения относительно верхней и левой стороны контейнера. А компонент box2 устанавливает ограничение левой стороны относительно правой стороны компонента box1 с отступом в 16 пикселей.



Центрирование компонентов

Кроме функции linkTo() для установки ограничений можно применять ряд функций, который центрируют компонент относительно родительского контейнера или других компонентов:

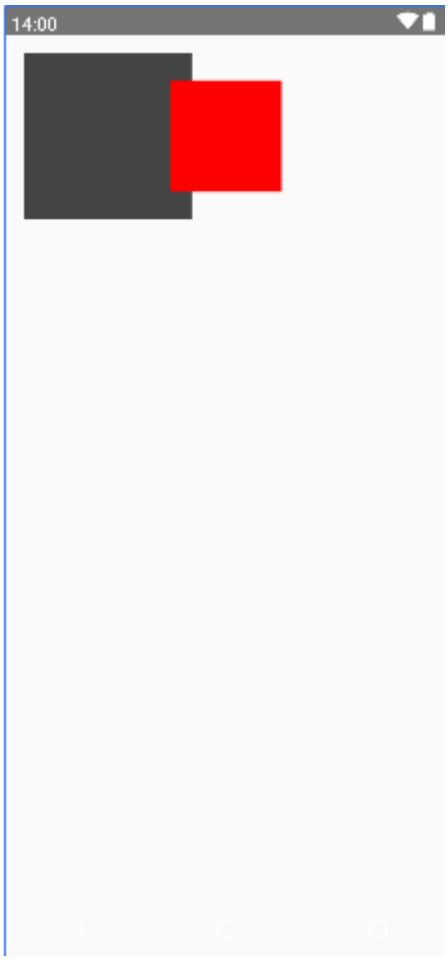
- centerHorizontallyTo(): располагает по центру по горизонтали
- centerVerticallyTo(): располагает по центру по вертикали
- centerAround(): центрирует по определенной стороне контейнера или другого компонента
- centerTo(): центрирует одновременно и по горизонтали и по вертикали

Например,

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
```

```
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout(Modifier.fillMaxSize()) {
                val box1 = createRef()
                val box2 = createRef()

Box(Modifier.size(150.dp).background(Color.DarkGray).constrainAs(box1) {
                    absoluteLeft.linkTo(parent.absoluteLeft, margin = 16.dp) //
отступ слева
                    top.linkTo(parent.top, margin = 16.dp) // отступ сверху
                })

                Box(Modifier.size(100.dp).background(Color.Red).constrainAs(box2)
{
                    centerVerticallyTo(box1)
                    centerHorizontallyTo(parent)
                })
            }
        }
    }
}
```

В данном случае box2 центрируется по горизонтали относительно родительского контейнера, а по вертикали располагается по центру box1:



Центрирование с помощью centerAround()

```
ConstraintLayout(Modifier.fillMaxSize()) {
    val box1 = createRef()
    val box2 = createRef()
    Box(Modifier.size(150.dp).background(Color.DarkGray).constrainAs(box1) {
        absoluteLeft.linkTo(parent.absoluteLeft, margin = 16.dp)
        top.linkTo(parent.top, margin = 16.dp)
    })

    Box(Modifier.size(100.dp).background(Color.Red).constrainAs(box2) {
        centerAround(box1.absoluteRight)
        centerAround(box1.bottom)
    })
}
```

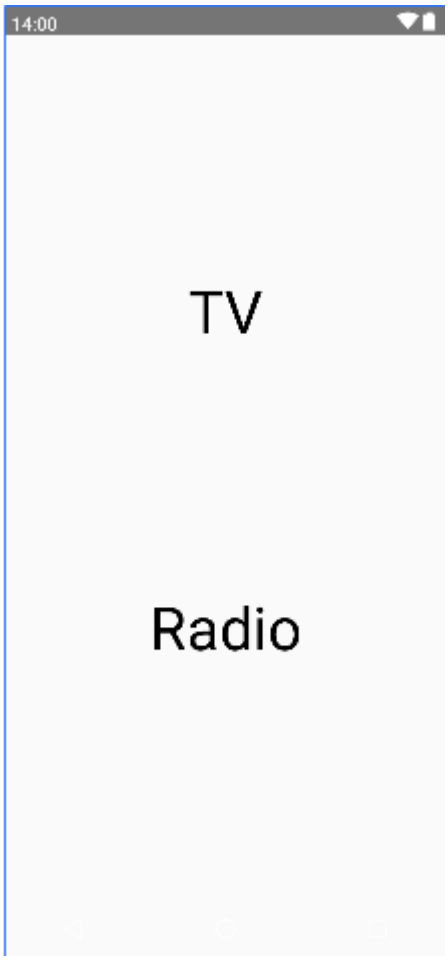
Взаимная установка ограничений

Различные компоненты могут взаимно устанавливать ограничения относительно друг друга. Например:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
```

```
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.constraintlayout.compose.ConstraintLayout
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout(Modifier.fillMaxSize()) {
                val (text1, text2) = createRefs()

                Text("TV", fontSize = 54.sp, modifier =
Modifier.constrainAs(text1) {
                    centerHorizontallyTo(parent)
                    top.linkTo(parent.top)
                    bottom.linkTo(text2.top)
                })

                Text("Radio", fontSize = 54.sp, modifier =
Modifier.constrainAs(text2) {
                    centerHorizontallyTo(parent)
                    top.linkTo(text1.bottom)
                    bottom.linkTo(parent.bottom)
                })
            }
        }
    }
}
```

Здесь оба компонента Text располагаются по центру контейнера по горизонтали. Но при этом верхняя граница text2 проходит по нижней границе text1, тогда как нижняя граница text1 - по верхней границе text2.



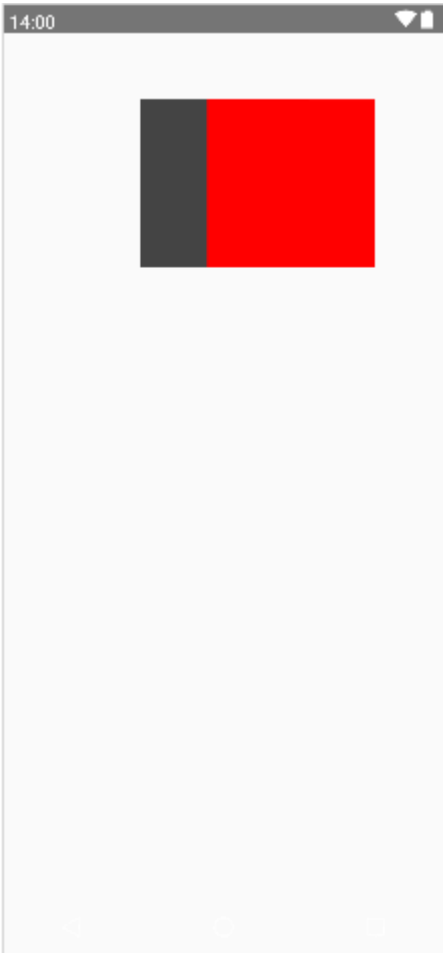
Смещение

Применение смещения позволяет перемещать компонент относительно доступного пространства:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout(Modifier.fillMaxSize()) {
                val (box1, box2) = createRefs()
```

```
    Box(Modifier.size(150.dp).background(Color.DarkGray).constrainAs(box1) {

                        top.linkTo(parent.top, margin = 60.dp)
                        linkTo(parent.start, parent.end)
                    })

                    Box(Modifier.size(150.dp).background(Color.Red).constrainAs(box2)
    {
                        top.linkTo(parent.top, margin = 60.dp)
                        linkTo(parent.start, parent.end, bias = 0.75f)
                    })
                }
            }
        }
    }
```

Здесь оба компонента Box по умолчанию позиционируются в одно и то же место, однако box2 при этом сдвинут на 75% доступной ширины:

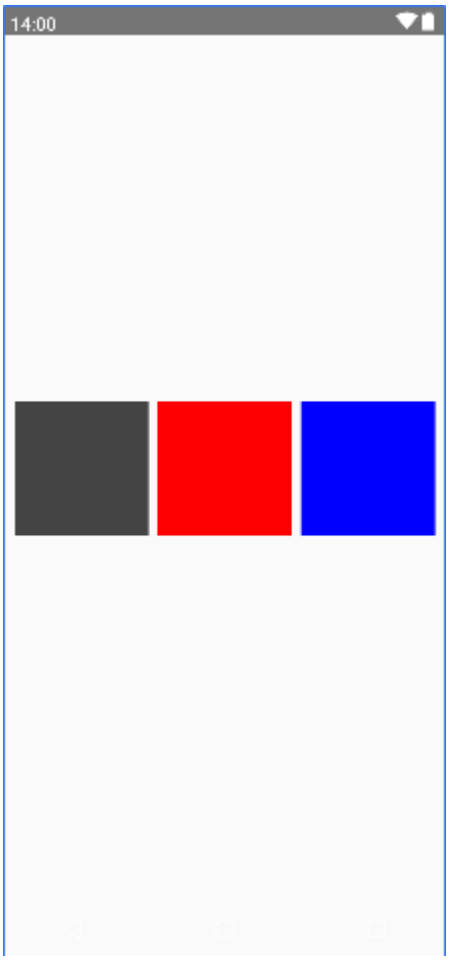


Создание цепочек компонентов

Цепочки или chains позволяют применить набор ограничений к группе компонентов и таким образом построить из них цепочку. Для этого применяются методы createHorizontalChain() или

createVerticalChain(), в которые передаются ссылки на компоненты. Например, следующий код создает горизонтальную цепочку между тремя компонентами Box:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout(Modifier.fillMaxSize()) {
                val (box1, box2, box3) = createRefs()
                createHorizontalChain(box1, box2, box3)

Box(Modifier.size(120.dp).background(Color.DarkGray).constrainAs(box1) {
                    centerVerticallyTo(parent)
                })
                Box(Modifier.size(120.dp).background(Color.Red).constrainAs(box2)
{
                    centerVerticallyTo(parent)
                })
                Box(Modifier.size(120.dp).background(Color.Blue).constrainAs(box3)
{
                    centerVerticallyTo(parent)
                })
            }
        }
    }
}
```



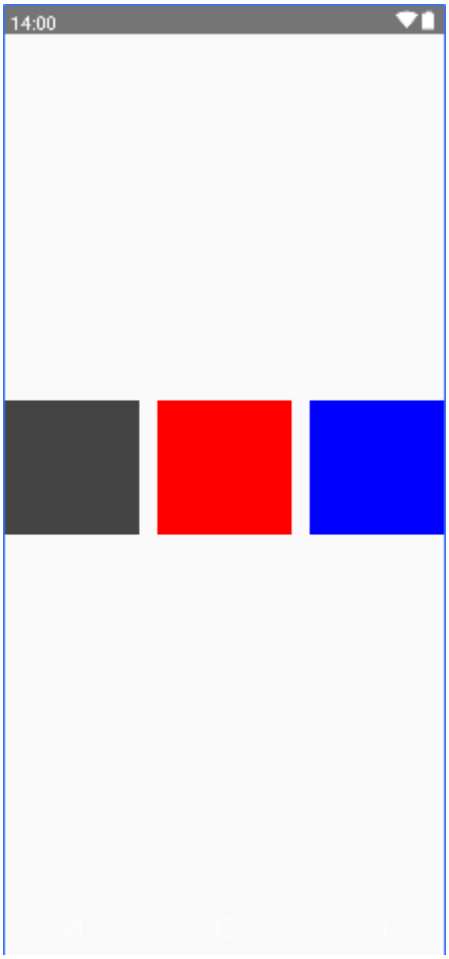
Цепочка компонентов может быть упорядочена в соответствии с одним из стилей, которые представляют класс ChainStyle:

- ChainStyle.Spread: пространство распределяется равномерно по всем компонентам, включая свободное пространство до первого компонента и после последнего компонента.
- ChainStyle.SpreadInside: пространство распределяется равномерно по всем компонентам, без какого-либо свободного места перед первым компонентом или после последнего компонента.
- ChainStyle.Packed: пространство распределяется до первого и после последнего компонента, компоненты упаковываются вместе без промежутков между собой.

Например, применим последний второй стиль:

```
import androidx.constraintlayout.compose.ChainStyle
.....................................
ConstraintLayout(Modifier.fillMaxSize()) {
    val (box1, box2, box3) = createRefs()
    createHorizontalChain(box1, box2, box3, chainStyle = ChainStyle.SpreadInside)
    Box(Modifier.size(120.dp).background(Color.DarkGray).constrainAs(box1) {
        centerVerticallyTo(parent)
    })
    Box(Modifier.size(120.dp).background(Color.Red).constrainAs(box2) {
        centerVerticallyTo(parent)
    })
    Box(Modifier.size(120.dp).background(Color.Blue).constrainAs(box3) {
```

```
            centerVerticallyTo(parent)
        })
    }
```



Направляющие линии guildeline

Направляющие линии guildelines в ConstraintLayout предоставляют горизонтальную или вертикальную опорную линию, относительно которой могут позиционироваться компоненты. Это может использоваться, когда группу компонентов необходимо выровнять относительно определенной линии оси. Положение направляющей может быть объявлено в процентах от высоты или ширины родительского контейнера или расположено с определенным смещением от определенной стороны:

- createGuidelineFromAbsoluteLeft(): создает направляющую линию от левой стороны контейнера
- createGuidelineFromAbsoluteRight(): создает направляющую линию от правой стороны контейнера
- createGuidelineFromBottom(): создает направляющую линию от нижней стороны контейнера
- createGuidelineFromTop(): создает направляющую линию от верхней стороны контейнера
- createGuidelineFromStart(): создает направляющую линию от начала контейнера
- createGuidelineFromEnd(): создает направляющую линию от конца контейнера

Все эти функции имеют две версии. Одна принимает параметр fraction - значение Float от 0.0 до 1.0, который представляет долю от направляющей до соответствующей стороны контейнера. Вторая версия принимает параметр offset - отступ в пикселях (Dp) между направляющей и соответствующей стороны контейнера.

Так, в следующем примере создается направляющая, которая расположена на 60% от начальной стороны контейнера (левой стороны при левосторонней ориентации):

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout(Modifier.fillMaxSize()) {
                val (box1, box2, box3) = createRefs()
                val guide = createGuidelineFromStart(fraction = .60f)

Box(Modifier.size(120.dp).background(Color.DarkGray).constrainAs(box1) {
                    top.linkTo(parent.top, margin = 30.dp)
                    end.linkTo(guide)
                })
                Box(Modifier.size(120.dp).background(Color.Red).constrainAs(box2)
{
                    top.linkTo(box1.bottom, margin = 20.dp)
                    start.linkTo(guide, margin = 40.dp)
                })
                Box(Modifier.size(120.dp).background(Color.Blue).constrainAs(box3)
{
                    top.linkTo(box2.bottom, margin = 40.dp)
                    end.linkTo(guide, margin = 60.dp)
                })
            }
        }
    }
}
```

Вначале создаем направляющую и сохраняем ссылку на нее в переменную guide:

```
val guide = createGuidelineFromStart(fraction = .60f)
```

Далее используем эту направляющую для установки ограничений. Например, для компонента Box с ссылкой box1 по направляющей определяется конечная сторона (правая сторона при левосторонней ориентации)

```
end.linkTo(guide)
```

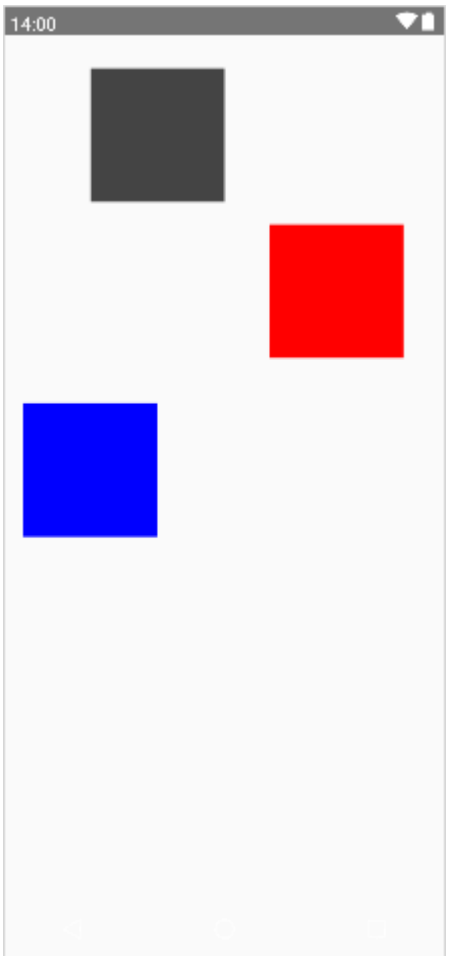
Та же сторона определяется и для компонента box3, только добавляется отступ в 60 пикселей влево:

```
end.linkTo(guide, margin = 60.dp)
```

А для box2 по направляющей guide определяется начальная сторона:

```
start.linkTo(guide, margin = 40.dp)
```

То есть направляющая guide выступает в качестве оси, вокруг которой расположены компоненты.



Аналогично можно было бы использовать смещение для образования направляющей линии, например, линия со смещением на 200 пикселей от начальной стороны контейнера:

```
val guide = createGuidelineFromStart(offset = 200.dp)
```

Барьеры

Барьеры ConstraintLayout создаются относительно определенной стороны одного или нескольких компонентов с помощью следующих функций:

- createStartBarrier()
- createEndBarrier()
- createTopBarrier()
- createBottomBarrier()

Каждой функции передается список компонентов, которым должен быть назначен барьер вместе с необязательным полем. В качестве результата функции возвращают ссылку на барьер, которую можно использовать для создания ограничений компонентов. Например:

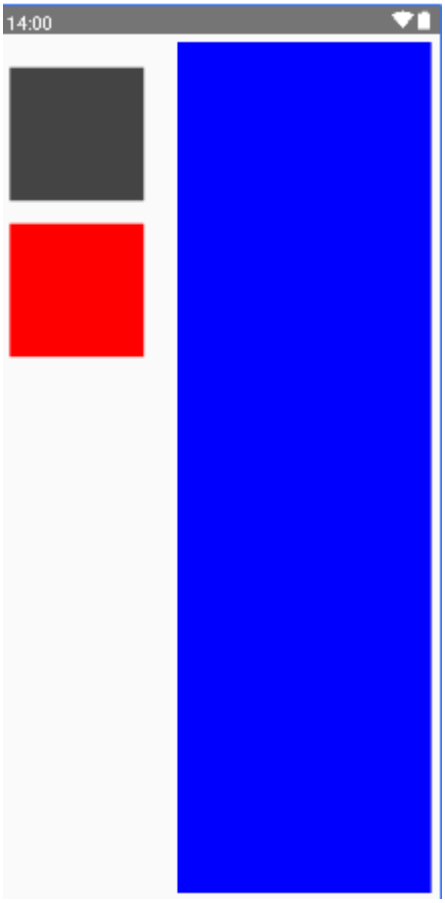
```
val barrier = createEndBarrier(box1, box2, margin = 30.dp)
```

Эта строка создает вертикальный барьер (начальный и конечный барьеры вертикальны, а верхний и нижний — горизонтальные), расположенный на расстоянии 30dp от конца кнопок button1 и button2. Если box1 и box2 имеют разную ширину, барьер будет находиться на расстоянии 30dp от конца самого широкого компонента в любой момент времени. Например, возьмем следующую программу:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout
import androidx.constraintlayout.compose.Dimension

class MainActivity : ComponentActivity() {
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout(Modifier.fillMaxSize()) {
                val (box1, box2, box3) = createRefs()

Box(Modifier.size(120.dp).background(Color.DarkGray).constrainAs(box1) {
                    top.linkTo(parent.top, margin = 30.dp)
                    start.linkTo(parent.start, margin = 8.dp)
                })
                Box(Modifier.size(120.dp).background(Color.Red).constrainAs(box2)
{
                    top.linkTo(box1.bottom, margin = 20.dp)
                    start.linkTo(parent.start, margin = 8.dp)
                })
                Box(Modifier.background(Color.Blue).constrainAs(box3) {
                    linkTo(parent.top, parent.bottom,topMargin = 8.dp,
bottomMargin = 8.dp)
                    linkTo(box1.end, parent.end, startMargin = 30.dp,endMargin =
8.dp)
                    width = Dimension.fillToConstraints
                    height = Dimension.fillToConstraints
                })
            }
        }
    }
}
```

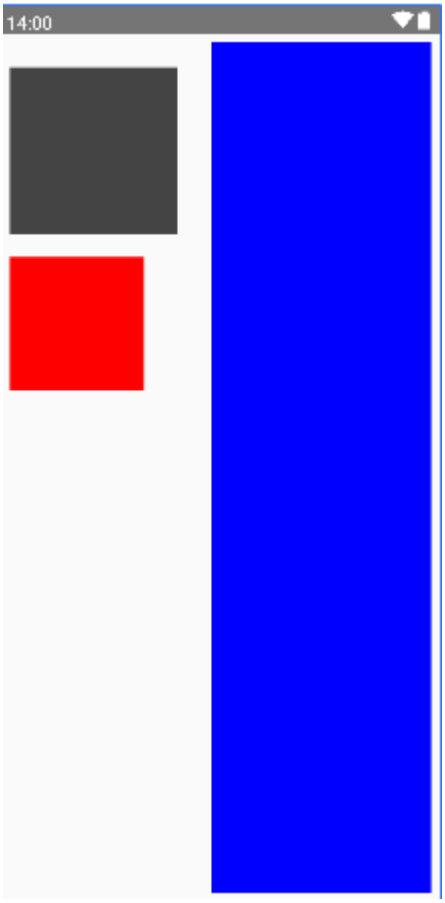
Здесь размер компонента box3 устанавливается таким, чтобы заполнять максимально доступное пространство, разрешенное его ограничениями. Это не только гарантирует, что компонент заполнит доступную высоту, но также позволит регулировать ширину в ответ на изменения размера box1 и box2. Для заполнения доступного пространства у box3 устанавливаются ограничения width и height на fillConstraints:



Здесь мы видим, что box3 действительно получает все доступное пространство. Поскольку его ограничение start зависит от box1.end, то при изменении ширины box1 также будет изменяться и ширина box3 автоматически. Например, если мы изменим ширину box1:

```
Box(Modifier.size(150.dp).background(Color.DarkGray).constrainAs(box1) { ...
```

то автоматически изменится ширина box3:



Однако от box2 компонент box3 никак не зависит. Поэтому изменение ширины box2 никак не затронет box3. И чтобы размеры box3 автоматически изменялись как при изменении box1, так и при изменении box2, применим барьеры:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout
import androidx.constraintlayout.compose.Dimension

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            ConstraintLayout(Modifier.fillMaxSize()) {
                val (box1, box2, box3) = createRefs()
                val barrier = createEndBarrier(box1, box2)
```

```
    Box(Modifier.size(120.dp).background(Color.DarkGray).constrainAs(box1) {
                        top.linkTo(parent.top, margin = 30.dp)
                        start.linkTo(parent.start, margin = 8.dp)
                    })
                    Box(Modifier.size(150.dp).background(Color.Red).constrainAs(box2)
    {
                        top.linkTo(box1.bottom, margin = 20.dp)
                        start.linkTo(parent.start, margin = 8.dp)
                    })
                    Box(Modifier.background(Color.Blue).constrainAs(box3) {
                        linkTo(parent.top, parent.bottom,topMargin = 8.dp,
    bottomMargin = 8.dp)
                        linkTo(box1.end, parent.end, startMargin = 30.dp,endMargin =
    8.dp)
                        start.linkTo(barrier, margin = 30.dp)
                        width = Dimension.fillToConstraints
                        height = Dimension.fillToConstraints
                    })
                }
            }
        }
    }
```

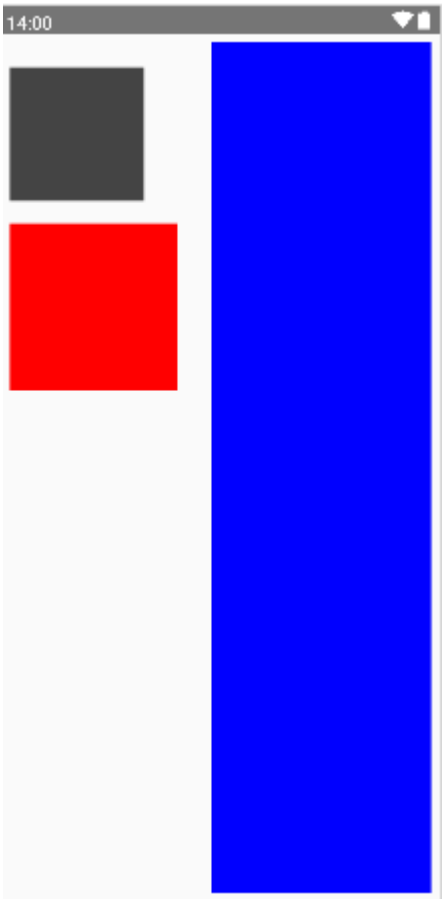
В данном случае создается вертикальный барьер по end для компонентов box1 и box2:

```
val barrier = createEndBarrier(box1, box2)
```

Далее устанавливаем этот барьер в качестве ограничения для начальной стороны для box3 со смещением в 30 пикселей:

```
start.linkTo(barrier, margin = 30.dp)
```

И теперь при изменении ширины box1 или box2 автоматически изменится ширина box3:



Наборы ограничений ConstraintSet

В предыдущих статьях все ограничения определялись внутри модификаторов, применяемых к отдельным компонентам. Но Jetpack Compose также позволяет объявлять ограничения отдельно в виде объекта ConstraintSet. Подобный набор ограничений затем можно передать в ConstraintLayout и применить к компонентам. Кроме того, создаваемые ограничения можно использовать повторно без необходимости дублировать объявления модификаторов. Это также обеспечивает гибкость при передачи различных наборов ограничений в зависимости от различных критериев. Например, контейнер может использовать разные наборы ограничений в зависимости от размера экрана или ориентации устройства.

Рассмотрим небольшой пример:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
```

```
import androidx.compose.ui.layout.layoutId
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout
import androidx.compose.ui.unit.Dp
import androidx.constraintlayout.compose.ConstraintSet

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            val constraints = myConstraintSet(margin = 8.dp)
            ConstraintLayout(constraints, Modifier.fillMaxSize()) {

Box(Modifier.size(200.dp).background(Color.DarkGray).layoutId("box1"))
            }
        }
    }
}
private fun myConstraintSet(margin: Dp): ConstraintSet {
    return ConstraintSet {
        val box1 = createRefFor("box1")
        constrain(box1) {
            linkTo(parent.top, parent.bottom, topMargin = margin, bottomMargin =
margin)
            linkTo(parent.start, parent.end, startMargin = margin, endMargin =
margin)
        }
    }
}
```

Прежде всего здесь определена функция myConstraintSet, которая принимает значение отступов и возвращает объект ConstraintSet:

```
private fun myConstraintSet(margin: Dp): ConstraintSet {
    return ConstraintSet {
```

Передача отступов позволяет сделать более гибкой настройку отступов в ограничениях, хотя это в принципе необязательно.

В функции ConstraintSet выполняется вызов функции createRefFor() для создания ссылки на любой компонуемый объект, к которому применяется набор ограничений. Причем в данном случае компонент будет иметь идентификатор "box1":

```
val box1 = createRefFor("box1")
```

Затем путем вызова функции constrain() создается набор ограничений. В эту функцию передается ссылка на компонент и далее в ней определяются ограничения.

```
constrain(box1) {
    linkTo(parent.top, parent.bottom, topMargin = margin, bottomMargin = margin)
    linkTo(parent.start, parent.end, startMargin = margin, endMargin = margin)
}
```

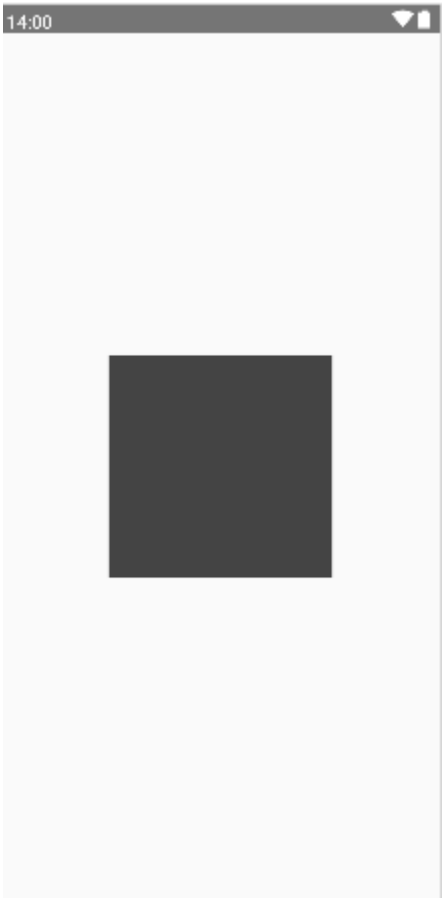
Создав набор ограничений, его можно передать в ConstraintLayout и применить к компонентам. Это включает в себя создание объекта ConstraintSet и его передачу его в ConstraintLayout:

```
val constraints = myConstraintSet(margin = 8.dp)
    ConstraintLayout(constraints, Modifier.fillMaxSize()) {
```

Для связывания ссылки на набор ограничений с компонентом применяется функции-модификатор layoutId(), в которую передается идентификатор компонента, к которому применяются ограничения:

```
Box(Modifier.size(200.dp).background(Color.DarkGray).layoutId("box1"))
```

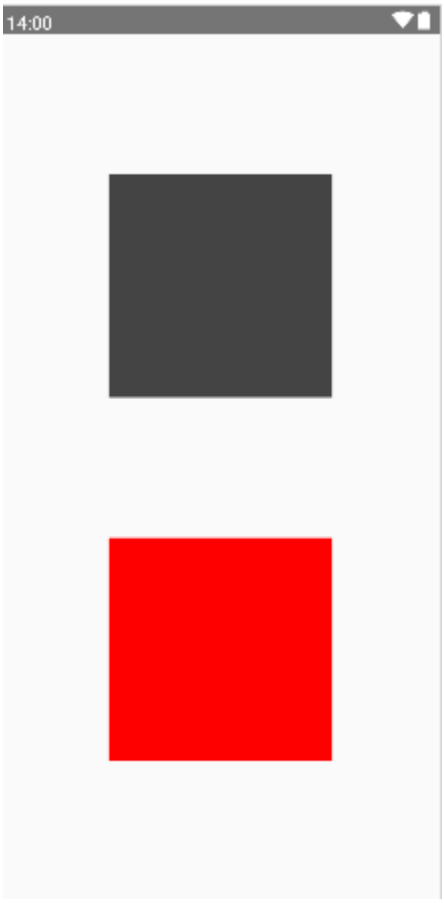
Обратите внимание, что здесь передается тот же идентификатор "box1", для которого устанавливались ограничения в ConstraintSet.



Аналогичным образом можно задавать ограничения в наборе для нескольких компонентов:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.layoutId
import androidx.compose.ui.unit.dp
import androidx.constraintlayout.compose.ConstraintLayout
import androidx.constraintlayout.compose.ConstraintSet

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            val constraints = myConstraintSet()
            ConstraintLayout(constraints, Modifier.fillMaxSize()) {

Box(Modifier.size(200.dp).background(Color.DarkGray).layoutId("box1"))
                Box(Modifier.size(200.dp).background(Color.Red).layoutId("box2"))
            }
        }
    }
}
private fun myConstraintSet(): ConstraintSet {
    return ConstraintSet {
        val box1 = createRefFor("box1")
        val box2 = createRefFor("box2")
        constrain(box1) {
            centerHorizontallyTo(parent)
            top.linkTo(parent.top)
            bottom.linkTo(box2.top)
        }
        constrain(box2) {
            centerHorizontallyTo(parent)
            top.linkTo(box1.bottom)
            bottom.linkTo(parent.bottom)
        }
    }
}
```

Теперь в функции myConstraintSet создаются две ссылки - box1 и box2, и для каждой из них создаются ограничения. Причем мы можем использовать эти ссылки для взаимной установки ограничений относительно друг друга.


14:00