Анимация

Для применения анимации в приложении Jetpack Compose предоставляет специальный API - Animation API. Этот API состоит из классов и функций, которые предоставляют широкие возможности по созданию анимации. Рассмотрим ключевые функции и классы Animation API.

Так, Compose Animation API предоставляет ряд анимаций состояния компонентов. В частности, это функции анимации для значений типов Bounds, Color, Dp, Float, Int, IntOffset, IntSize, Offset, Rect и Size. Подобные функции покрывают большинство потребностей в анимации компонентов.

Подобные функции анимаций используют одно и то же соглашение об именах. В частности, все они называются по шаблону:

```
animate*AsState
```

где символ * представляет типо состояния, которое анимируется. Например, если нужно анимировать изменение цвета (например, цвета фона компонента), то применяется функция animateColorAsState(). В реальности, функции передается целевое (конечное) значение, которое должно получить состояние. И функция анимируется переход от текущего значения к целевому значению. Рассмотрим ряд подобных функций.

Анимация Dp. animateDpAsState

Функция animateDpAsState() выполняет анимацию значений Dp, которые могут применяться для установки размеров, отступов и т.д. Она имеет следующие параметры:

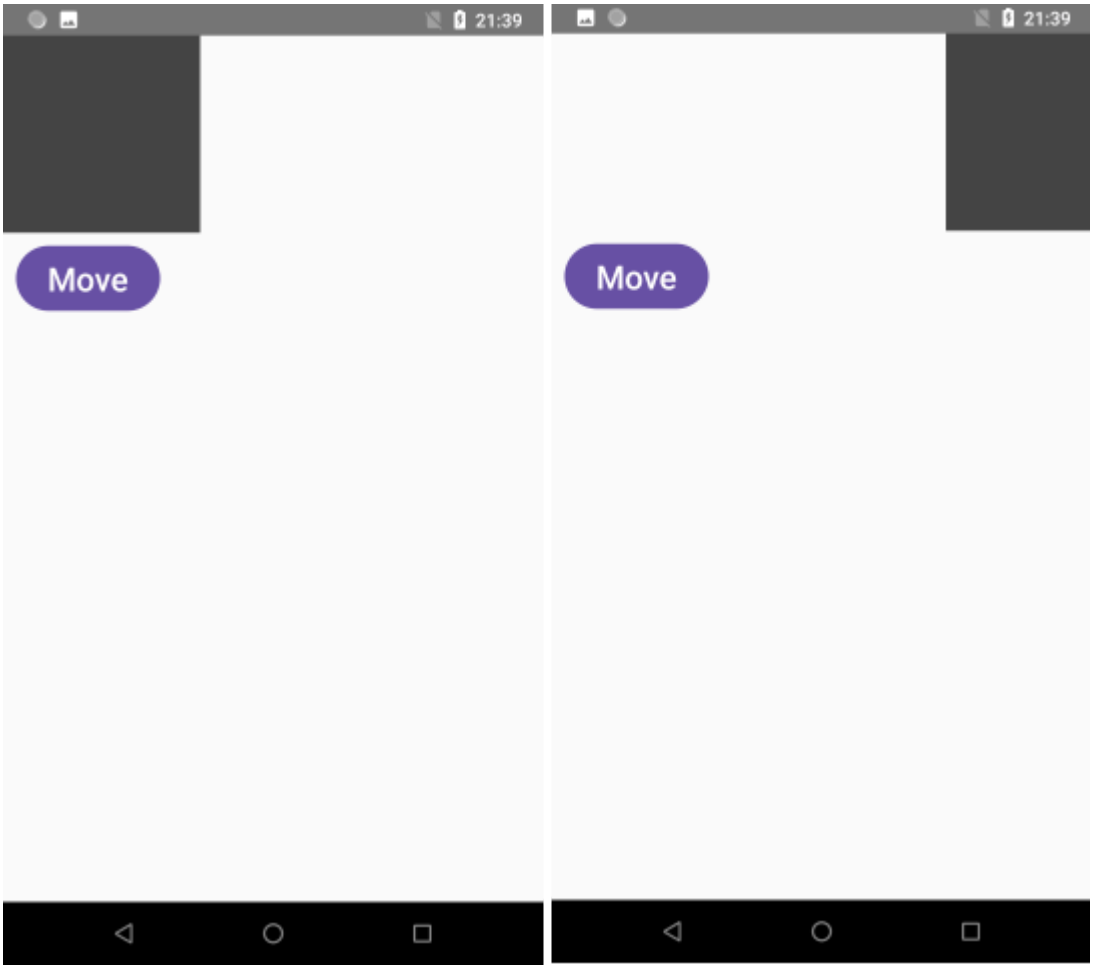
```
@Composable
public fun animateDpAsState(
    targetValue: Dp,
    animationSpec: AnimationSpec<Dp> = dpDefaultSpring,
    label: String = "DpAnimation",
    finishedListener: ((Dp) -> Unit)? = null
): State<Dp>
```

- targetValue: значение Dp, к которому надо выполнить переход
- animationSpec: применяемая анимация в виде объекта AnimationSpec
- label: название анимации
- finishedListener: функция, которая выполняется при завершении анимации

Обязательным параметром является только targetValue. Рассмотрим простейший пример, в котором анимируется отступ слева, за счет чего возникает иллюзия движения объекта:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val startOffset = 0   // начальная позиция
            val endOffset = 300   // конечная позиция
            val boxWidth = 150      // ширина компонента

            var boxState by remember { mutableStateOf(startOffset)}

            val offset by animateDpAsState(targetValue = boxState.dp)

            Column(Modifier.fillMaxWidth()) {

Box(Modifier.padding(start=offset).size(boxWidth.dp).background(Color.DarkGray))

                Button({boxState = if (boxState==startOffset) {endOffset} else
{startOffset} },
                    Modifier.padding(10.dp)) {
                    Text("Move", fontSize = 25.sp)
                }
            }
        }
    }
}
```

В данном случае по нажатию на кнопку происходит перемещение компонента с помощью анимации значений Dp:



Здесь в начале определяем некоторые значения, которые будут вычисляться для изменения позиции компонента Box:

```
val startOffset = 0   // начальная позиция
val endOffset = 400   // конечная позиция
val boxWidth = 200      // ширина компонента
```

То есть наш Box имеет ширину boxWidth и будет перемещаться от позиции startOffset до endOffset.

Далее определяем состояние, от которого будет зависеть позиция компонента Box:

```
var boxState by remember { mutableStateOf(startOffset)}
```

Это состояние по умолчанию равно startOffset.

Затем определяем анимацию позиции Box с помощью функции animateDpAsState()

```
val offset: Dp by animateDpAsState(targetValue = boxState.dp)
```

Параметр targetValue на основании boxState определяет позицию, к которой надо выполнить переход. В данном случае все просто - boxState будет хранить отступ, и здесь мы его преобразуем в значение Dp.

Функция animateDpAsState() возвращает значение типа State, и с помощью оператора by из него извлекаем непосредственно значение Dp и передаем его переменной offset.

Далее используем значение offset, полученное из функции animateDpAsState(), для установки отступа от начала контейнера для компонента Box:

```
Box(Modifier.padding(start=offset)      // устанавливаем отступ
        .size(boxWidth.dp)
        .background(Color.DarkGray))
```

И для запуска анимации на кнопку Move вешаем обработчик нажатия, который переключает значение boxState со startOffset на endOffset и обратно.

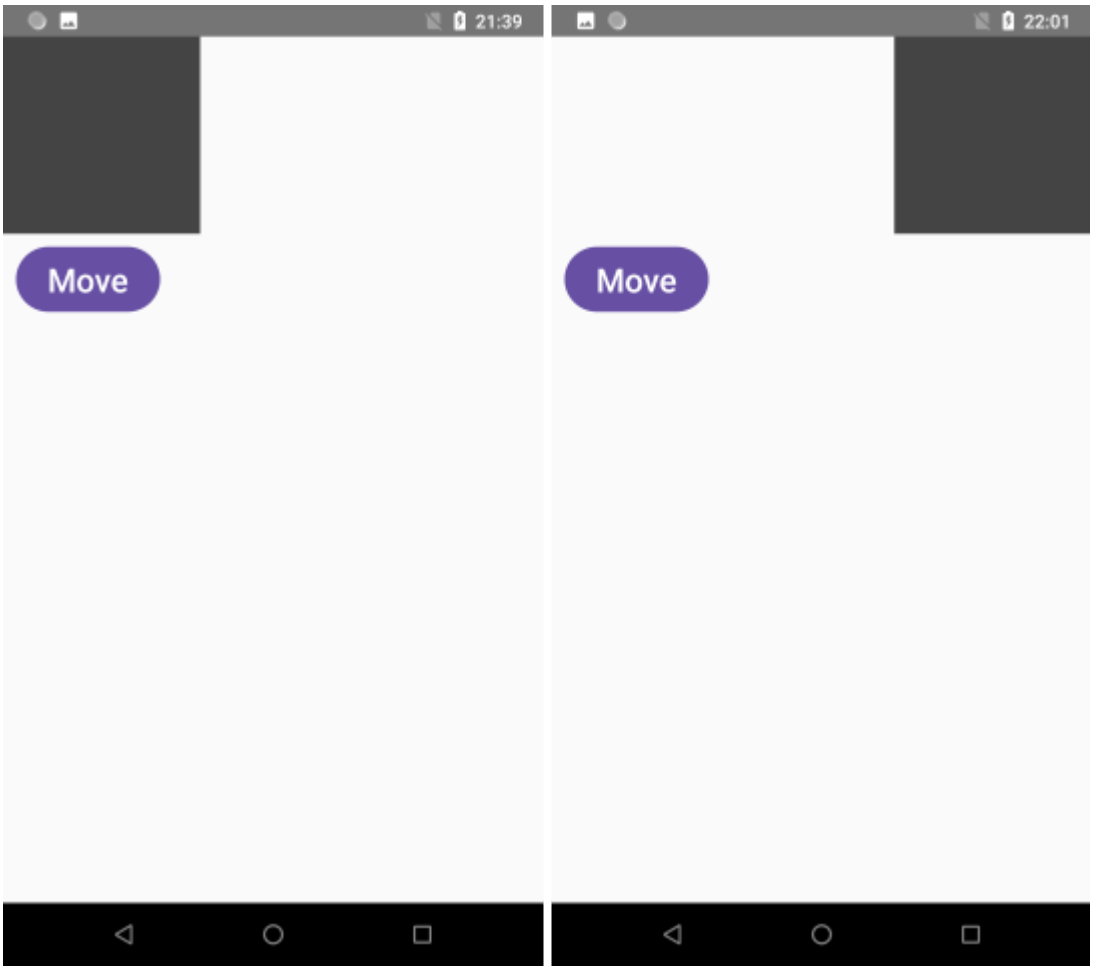
```
Button({boxState = if (boxState==startOffset) {endOffset} else {startOffset} },
```

То есть при нажатии на кнопку изменится значение boxState. Поскольку функция animateDpAsState зависит от boxState и на его основании устанавливает целевое значение, к которому надо перейти, то будет выполняться анимация для перехода к новому целевому значению.

Подобным образом мы можем анимировать любые параметры на основе Dp, не только отсутпы. Однако чуть улучшим предыдущий код. Так, выше значение endOffset взято случайным образом. Теперь сделаем движение Box до конца экрана:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
```

```
import androidx.compose.ui.unit.sp
import androidx.compose.ui.platform.LocalConfiguration


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val boxWidth = 150      // ширина компонента
            val startOffset = 0   // начальная позиция
            val endOffset = LocalConfiguration.current.screenWidthDp - boxWidth //
конечная позиция

            var boxState by remember { mutableStateOf(startOffset)}

            val offset by animateDpAsState(targetValue = boxState.dp)

            Column(Modifier.fillMaxWidth()) {

Box(Modifier.padding(start=offset).size(boxWidth.dp).background(Color.DarkGray))

                Button({boxState = if (boxState==startOffset) {endOffset} else
{startOffset} },
                    Modifier.padding(10.dp)) {
                    Text("Move", fontSize = 25.sp)
                }
            }
        }
    }
}
```



Теперь для вычисления конечной позиции применяется свойство LocalConfiguration.current.screenWidthDp, которое возвращает ширину экрана:

```
val endOffset = LocalConfiguration.current.screenWidthDp - boxState
```

Кроме того, при вычислении конечной позиции учитываем ширину компонента, чтобы он не вылез за экран.

Функция tween. Время и сглаживание анимации

Большинство функций, которые устанавливают анимацию, имеют параметр animationSpec, который представляет интерфейс AnimationSpec. Данному параметру передается объект одного из классов, которые реализуют AnimationSpec. В частности, с помощью этого параметра можно настроить продолжительность анимации, задержку, эффект отскок, повторение и замедление анимации и ряд других моментов.

Например, рассмотренная в прощлой теме функция animateDpAsState(), которая выполняет анимацию значений Dp, также имеет подобный параметр:

```
@Composable
public fun animateDpAsState(
    targetValue: Dp,
```

```
    animationSpec: AnimationSpec<Dp> = dpDefaultSpring,   // установка анимации
    label: String = "DpAnimation",
    finishedListener: ((Dp) -> Unit)? = null
): State<Dp>
```

Для создания объекта AnimationSpec Jetpack Compose предоставляет множество специальных функций и значений. Рассмотрим наиболее используемую из них - функцию tween(). Она имеет следующую сигнатуру:

```
public fun <T> tween(
    durationMillis: Int = DefaultDurationMillis,
    delayMillis: Int = 0,
    easing: Easing = FastOutSlowInEasing
): TweenSpec<T>
```

Она принимает следующие параметры:

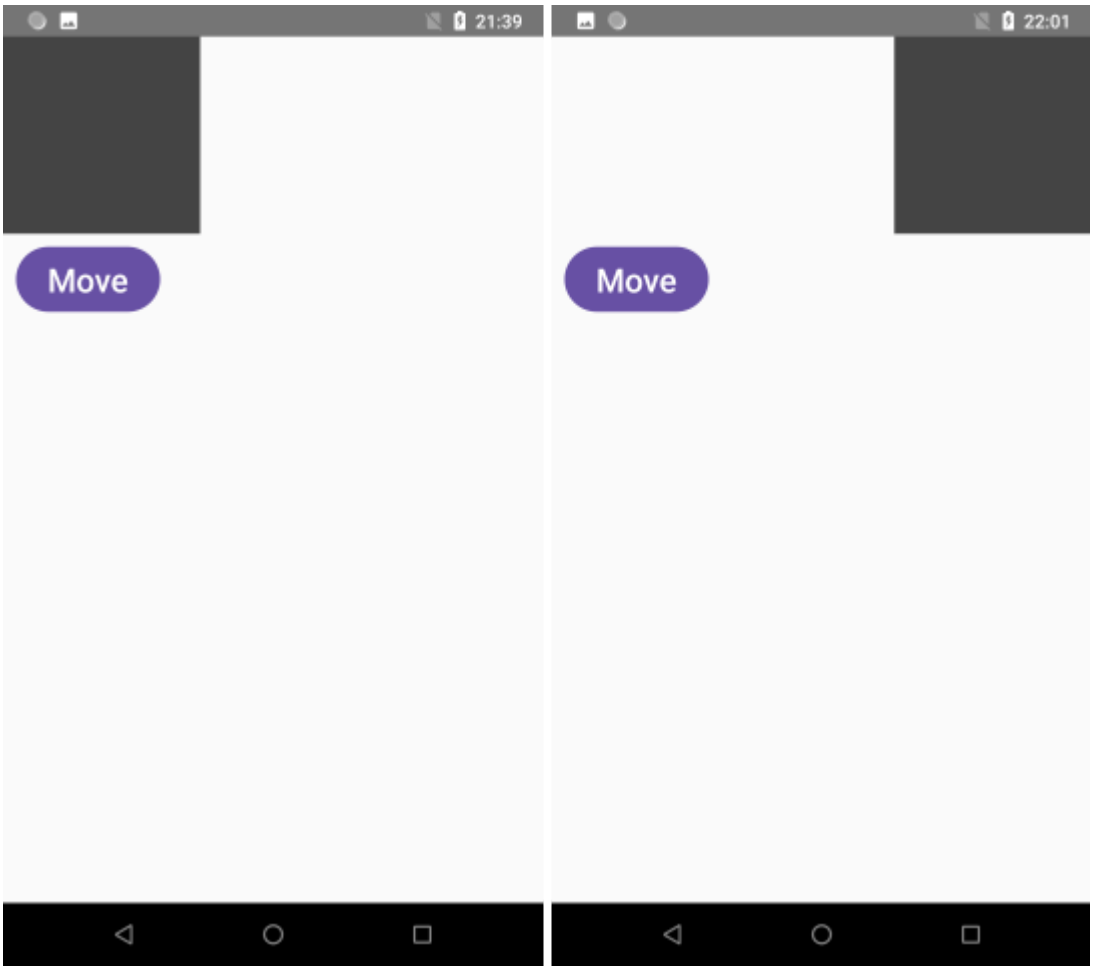
- durationMillis: длительность анимации в миллисекундах
- delayMillis: устанавливает начальную задержку
- easing: позволяет ускорять и замедлять анимацию

Наиболее распространенный сценарий использования функции tween() - настройка времени анимации. Например:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.animation.core.tween
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
```

```
import androidx.compose.ui.platform.LocalConfiguration


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val boxWidth = 150      // ширина компонента
            val startOffset = 0   // начальная позиция
            val endOffset = LocalConfiguration.current.screenWidthDp - boxWidth //
конечная позиция

            var boxState by remember { mutableStateOf(startOffset)}

            val offset by animateDpAsState(
                targetValue = boxState.dp,
                animationSpec = tween(durationMillis = 5000) // анимация длится 5
секунд
            )

            Column(Modifier.fillMaxWidth()) {

Box(Modifier.padding(start=offset).size(boxWidth.dp).background(Color.DarkGray))

                Button({boxState = if (boxState==startOffset) {endOffset} else
{startOffset} },
                    Modifier.padding(10.dp)) {
                    Text("Move", fontSize = 25.sp)
                }
            }
        }
    }
}
```

В данном случае по нажатию на кнопку происходит перемещение компонента с помощью анимации значений Dp:



Здесь в начале определяем некоторые значения, которые будут вычисляться для изменения позиции компонента Box:

```
val boxWidth = 150      // ширина компонента
val startOffset = 0   // начальная позиция
val endOffset = LocalConfiguration.current.screenWidthDp - boxWidth // конечная позиция
```

То есть наш Box имеет ширину boxWidth и будет перемещаться от позиции startOffset до endOffset.

Далее определяем состояние, от которого будет зависеть позиция компонента Box:

```
var boxState by remember { mutableStateOf(startOffset)}
```

Затем определяем анимацию позиции Box с помощью функции animateDpAsState()

```
val offset by animateDpAsState(
    targetValue = boxState.dp,
    animationSpec = tween(durationMillis = 5000) // анимация длится 5 секунд
)
```

Параметр targetValue на основании boxState определяет позицию, к которой надо выполнить переход. А параметру animationSpec присваивается значение функции tween(), в которую передается число 5000. То есть анимация будет длиться 5 миллисекунд.

Результат функцит animateDpAsState() передаем в переменну offset и далее используем ее для установки отступа от начала контейнера для компонента Box:

```
Box(Modifier.padding(start=offset)      // устанавливаем отступ
        .size(boxWidth.dp)
        .background(Color.DarkGray))
```

И для запуска анимации на кнопку Move вешаем обработчик нажатия, который переключает значение boxState со startOffset на endOffset и обратно.

```
Button({boxState = if (boxState==startOffset) {endOffset} else {startOffset} },
```

easing

Параметр easing в функции tween() позволяет ускорять и замедлять анимацию. В качестве значения можно передать одно из значений, определенных в пакете androidx.compose.animation.core.:

- FastOutSlowInEasing
- LinearOutSlowInEasing
- FastOutLinearEasing
- LinearEasing
- CubicBezierEasing

Рассмотрим их применение:

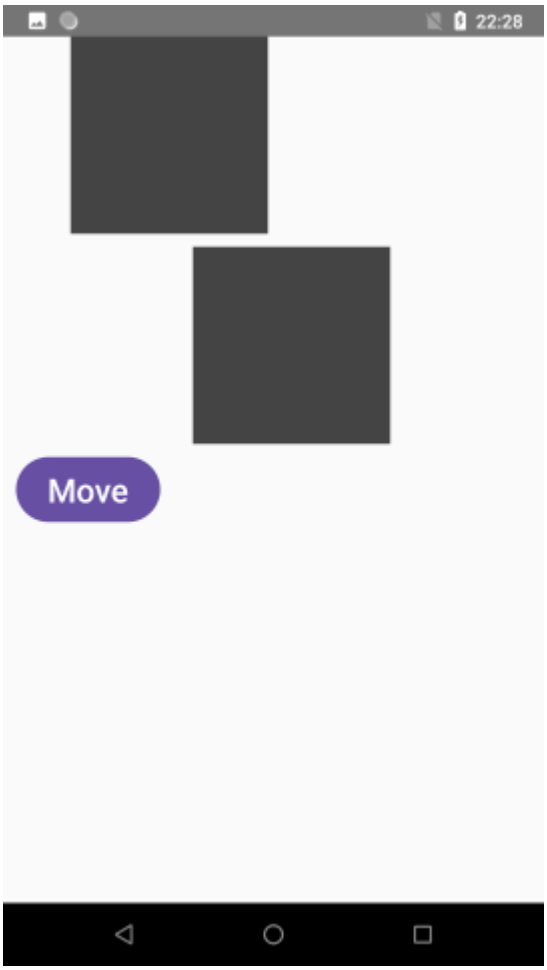
```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.FastOutSlowInEasing
import androidx.compose.animation.core.LinearOutSlowInEasing
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.animation.core.tween
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
```

```
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.compose.ui.platform.LocalConfiguration


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val boxWidth = 150
            val startOffset = 0
            val endOffset = LocalConfiguration.current.screenWidthDp - boxWidth

            var boxState by remember { mutableStateOf(startOffset)}

            val offset by animateDpAsState(
                targetValue = boxState.dp,
                animationSpec = tween(durationMillis = 5000, easing =
FastOutSlowInEasing) // анимация длится 5 секунд
            )
            val offset1 by animateDpAsState(
                targetValue = boxState.dp,
                animationSpec = tween(durationMillis = 5000, easing =
LinearOutSlowInEasing) // анимация длится 5 секунд
            )

            Column(Modifier.fillMaxWidth()) {

Box(Modifier.padding(start=offset).size(boxWidth.dp).background(Color.DarkGray))
                Box(Modifier.padding(start=offset1,
top=10.dp).size(boxWidth.dp).background(Color.DarkGray))

                Button({boxState = if (boxState==startOffset) {endOffset} else
{startOffset} },
                    Modifier.padding(10.dp)) {
                    Text("Move", fontSize = 25.sp)
                }
            }
        }
    }
}
```

В данном случае я добавил второй компонент Box для сравнения. Для каждого из них применяется анимация в течение 5 секунд, однако значение параметров easing отличается:

```
val offset by animateDpAsState(
    targetValue = boxState.dp,
    animationSpec = tween(durationMillis = 5000, easing = FastOutSlowInEasing) //
анимация длится 5 секунд
)
val offset1 by animateDpAsState(
    targetValue = boxState.dp,
    animationSpec = tween(durationMillis = 5000, easing = LinearOutSlowInEasing)
// анимация длится 5 секунд
)
```

Что повлияет на принцип изменение отступа:



Если для параметра easing применяется значение CubicBezierEasing, то в конструктор надо передать координаты двух контрольных точек кривой Безье, на основании которой расчитывается анимация:

```
import androidx.compose.animation.core.CubicBezierEasing
.....................
 val offset by animateDpAsState(
    targetValue = boxState.dp,
    animationSpec = tween(durationMillis = 5000, easing = CubicBezierEasing(0f,
```

```
1f, 0.5f,1f))
)
```

Функция repeatable и повторение анимации

Для повторения анимации применяется класс RepeatableSpec (который реализует интерфейс AnimationSpec). Объект этого класса можно получить с помощью функции repeatable():

```
public fun <T> repeatable(
    iterations: Int,
    animation: DurationBasedAnimationSpec<T>,
    repeatMode: RepeatMode = RepeatMode.Restart,
    initialStartOffset: StartOffset = StartOffset(0)
): RepeatableSpec<T>
```

Она принимает следующие параметры:

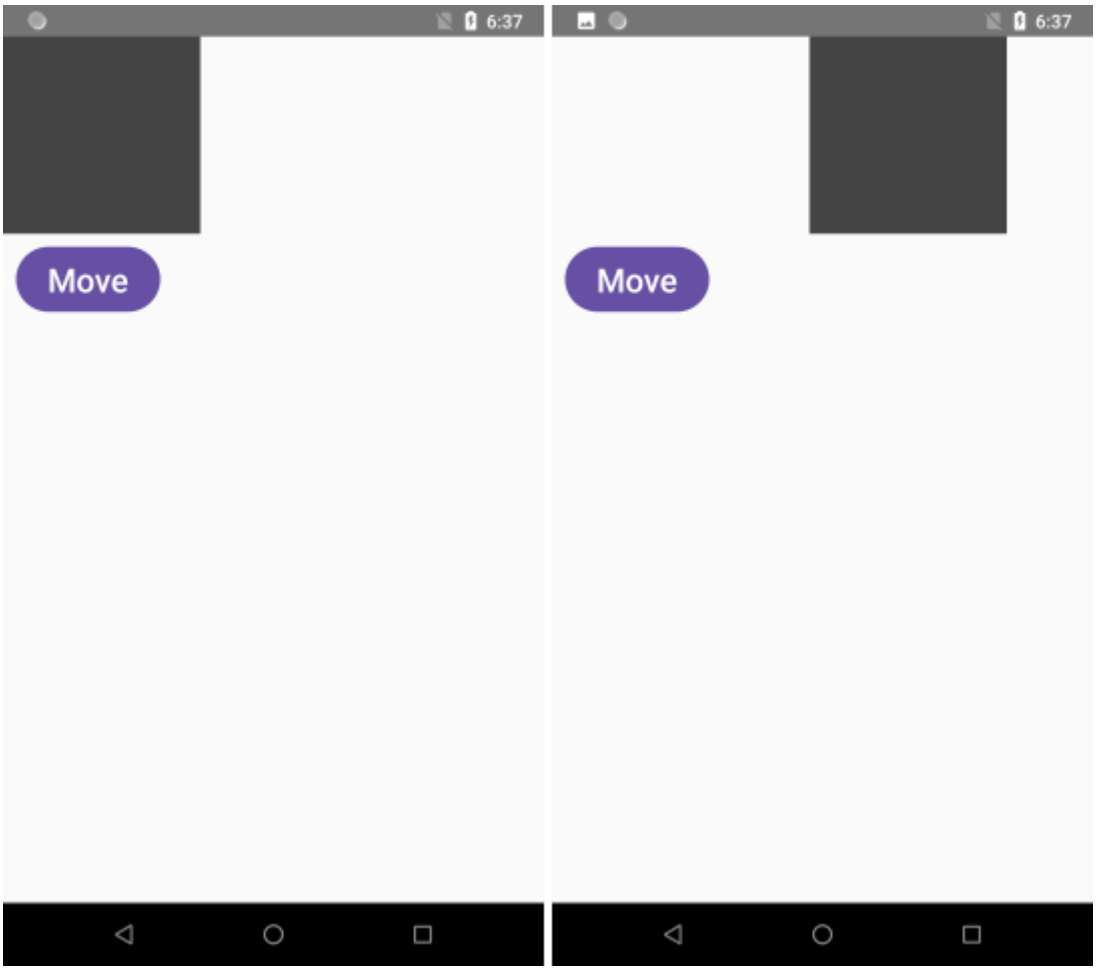
- iterations: количество повторений
- animation: спецификация анимации. Можно использовать другие функции, которые возвращают DurationBasedAnimationSpec, например, функцию tween()
- repeatMode: значение RepeatMode, которое указывает, должна ли анимация выполняться от начала до конца (значение RepeatMode.Restart) или должна выполняться в обратном порядке - от конца к началу (значение RepeatMode.Reverse)
- initialStartOffset: смещение относительно начала

В данном случае первые два параметра обязательные. Например, повторим анимацию 3 раза:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.animation.core.repeatable
import androidx.compose.animation.core.tween
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
```

```
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.compose.ui.platform.LocalConfiguration


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val boxWidth = 150
            val startOffset = 0
            val endOffset = LocalConfiguration.current.screenWidthDp - boxWidth

            var boxState by remember { mutableStateOf(startOffset)}

            val offset by animateDpAsState(
                targetValue = boxState.dp,
                animationSpec = repeatable(3, animation = tween(2000))
            )

            Column(Modifier.fillMaxWidth()) {

Box(Modifier.padding(start=offset).size(boxWidth.dp).background(Color.DarkGray))

                Button({boxState = if (boxState==startOffset) {endOffset} else
{startOffset} },
                    Modifier.padding(10.dp)) {
                    Text("Move", fontSize = 25.sp)
                }
            }
        }
    }
}
```

В данном случае по нажатию на кнопку запускается анимация, которая перемещает компонента Box от начала до конца по горизонтали, причем процесс повторяется три раза:



Здесь в начале определяем некоторые значения, которые будут вычисляться для изменения позиции компонента Box:

```
val boxWidth = 150      // ширина компонента
val startOffset = 0   // начальная позиция
val endOffset = LocalConfiguration.current.screenWidthDp - boxWidth // конечная
позиция
```

То есть наш Box имеет ширину boxWidth и будет перемещаться от позиции startOffset до endOffset.

Далее определяем состояние, от которого будет зависеть позиция компонента Box:

```
var boxState by remember { mutableStateOf(startOffset)}
```

Затем определяем анимацию позиции Box с помощью функции animateDpAsState()

```
val offset by animateDpAsState(
    targetValue = boxState.dp,
    animationSpec = repeatable(3, animation = tween(2000))
)
```

Параметр targetValue на основании boxState определяет позицию, к которой надо выполнить переход. А параметру animationSpec присваивается значение функции repeatable(). Первый аргумент функции - число 3 указывает на количество повторений. А параметру animation получает результат функции tween(), которая устанавливает время анимации - 2000 миллисекунд.

Результат функцит animateDpAsState() передаем в переменну offset и далее используем ее для установки отступа от начала контейнера для компонента Box:

```
Box(Modifier.padding(start=offset)      // устанавливаем отступ
        .size(boxWidth.dp)
        .background(Color.DarkGray))
```

И для запуска анимации на кнопку Move вешаем обработчик нажатия, который переключает значение boxState со startOffset на endOffset и обратно.

```
Button({boxState = if (boxState==startOffset) {endOffset} else {startOffset} },
```

Однако при выполнения анимации после ее завершения компонент Box резко перемещается на начальную позицию и с нее уже начинает вторую итерацию анимации. Параметр repeatMode при значении RepeatMode.Reverse позволяет задать воспроизведение анимации в обратном порядке:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.RepeatMode
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.animation.core.repeatable
import androidx.compose.animation.core.tween
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.compose.ui.platform.LocalConfiguration
```

```
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val boxWidth = 150
            val startOffset = 0
            val endOffset = LocalConfiguration.current.screenWidthDp - boxWidth

            var boxState by remember { mutableStateOf(startOffset)}

            val offset by animateDpAsState(
                targetValue = boxState.dp,
                // 3 раза повторяем анимацию в течение 2 секунд, повторяем в
обратном порядке
                animationSpec = repeatable(3, animation = tween(2000), repeatMode
= RepeatMode.Reverse)
            )

            Column(Modifier.fillMaxWidth()) {

Box(Modifier.padding(start=offset).size(boxWidth.dp).background(Color.DarkGray))

                Button({boxState = if (boxState==startOffset) {endOffset} else
{startOffset} },
                    Modifier.padding(10.dp)) {
                    Text("Move", fontSize = 25.sp)
                }
            }
        }
    }
}
```

Функция spring и эффект отскока

Встроенная функция spring() позволяет добавить к анимации эффект отскока, подобно тому, как мяч ударяется об землю и после этого еще несколько раз подпрыгивает, делая несколько отскоков, пока совсем не остановит свое движение. Функция spring() имеет следующие параметры:

```
public fun <T> spring(
    dampingRatio: Float = Spring.DampingRatioNoBouncy,
    stiffness: Float = Spring.StiffnessMedium,
    visibilityThreshold: T? = null
): SpringSpec<T>
```

- dampingRatio: степень затухания - определяет скорость, с которой затухает эффект подпрыгивания. Определяется как значение типа Float где 1.0 представляет отсутствие отскока, а 0.1 — самый высокий отскок. Вместо использования значений Float при настройке степени затухания также доступны следующие предопределенные константы:

- - DampingRatioHighBouncy
- - DampingRatioLowBouncy
- - DampingRatioMediumBouncy
- - DampingRatioNoBouncy

- stiffness: жесткость при отскоке. При использовании меньшей жесткости диапазон движения при подпрыгивании будет больше. Для определения жесткости можно использовать ряд встроенных констант:

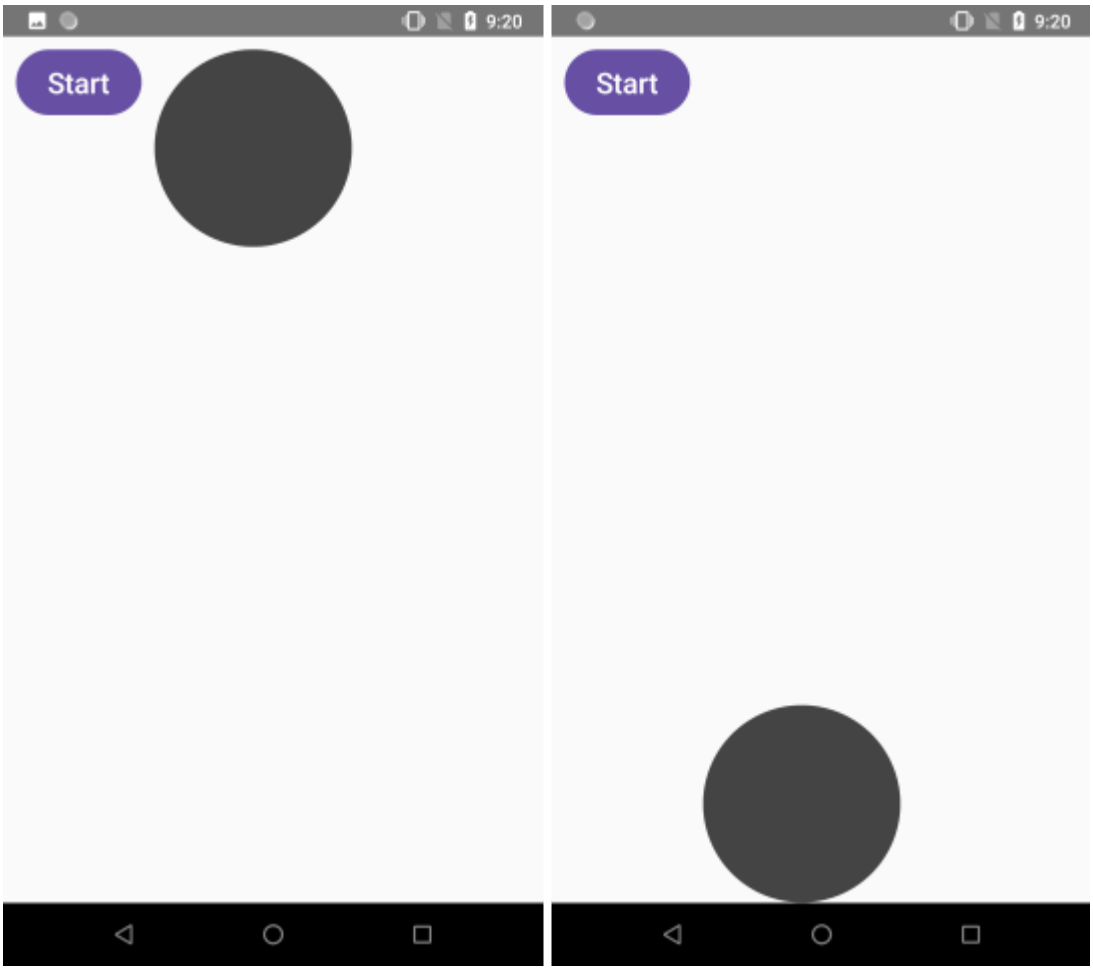
- - StiffnessHigh
- - StiffnessLow
- - StiffnessMedium
- - StiffnessMediumLow
- - StiffnessVeryLow
- - visibilityThreshold: предел видимости

Для демонстрации эффекта отскока рассмотрим следующее приложение:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.animation.core.spring
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalConfiguration
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
```

```
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val circleHeight = 150     // высота (диаметр) круга
            val startOffset = 10      // начальный отступ
            val endOffset = LocalConfiguration.current.screenHeightDp -
circleHeight     // предельная позиция
            var circleOffset by remember { mutableStateOf(startOffset)}
            val offset: Dp by animateDpAsState(
                targetValue = circleOffset.dp,
                animationSpec =if (circleOffset==endOffset) {
                    spring(dampingRatio = 0.3f)     // сильный отскок
                } else {
                    spring(dampingRatio = 1.0f)     // отсутствие отскока
                }
            )

            Row(Modifier.fillMaxSize()) {
                Button({circleOffset = if (circleOffset==startOffset) endOffset
else startOffset },
                    Modifier.padding(10.dp)) {
                    Text("Start", fontSize = 22.sp)
                }

Box(Modifier.padding(top=offset).size(circleHeight.dp).clip(CircleShape).backgroun
d(Color.DarkGray))
            }
        }
    }
}
```

Здесь определяется компонент Box в виде круга, и по нажатию на кнопку запускается анимация, в результате которой компонент движется вниз, пока не ударится о низ устройства.



Для целей анимации определяем ряд переменных, как начальная и конечная позиция круга и его диаметр (высота)

```
val circleHeight = 150     // высота (диаметр) круга
val startOffset = 10      // начальный отступ
val endOffset = LocalConfiguration.current.screenHeightDp - circleHeight     //
предельная позиция
```

Далее определем состояние, которое будет хранить текущий отступ круга:

```
var circleOffset by remember { mutableStateOf(startOffset)}
```

Затем создаем анимацию с помощью функции animateDpAsState(), которая будет определять отступ:

```
val offset: Dp by animateDpAsState(
    targetValue = circleOffset.dp,

    animationSpec =if (circleOffset==endOffset) {
        spring(dampingRatio = 0.3f)     // сильный отскок
    } else {
        spring(dampingRatio = 1.0f)     // отсутствие отскока
```

```
    }
)
```

Здесь прежде всего определяем targetValue - значение, к которому надо перейти в процессе анимации. Ему передается значение переменной circleOffset, преобразованное в значение Dp. Так как, поскольку по нажатию на кнопку мы будем изменять значение circleOffset со startOffset на endOffset и обратно, то впоследствии circleOffset будет хранить следующее значение, к которому надо перейти.

Параметр animationSpec, который задает анимацию, также использует условное выражение. Если значение, к которому надо перейти, является конечным - endOffset, то с помощью функции spring() в конце формируем отскок. Значение 0.3f можно характеризовать как сильный отскок. Иначе если мы находимся на конечной позиции, когда круг упал вниз, и его движением с отскоком завершилось, то просто возвращаем круг в начальную точку. В этом случае нам отскок не нужен, поэтому передаем в функцию spring() значение 1.0f

Для запуска анимации определяем кнопку, которая меняет значение circleOffset и тем самым вызывает анимацию:

```
Button({circleOffset = if (circleOffset==startOffset) endOffset else startOffset
},
```

И в конце идет собственно компонент Box, стилизованный под круг:

```
Box(Modifier.padding(top=offset).size(circleHeight.dp).clip(CircleShape).backgroun
d(Color.DarkGray))
```

Его значение отступа с верху привязано к переменной offset, которая генерируется функцией animateDpAsState()

В примере выше в функцию spring() передавались числовые значения, и также можно передавать предустановленные константы:

```
import androidx.compose.animation.core.Spring.DampingRatioHighBouncy
import androidx.compose.animation.core.Spring.DampingRatioNoBouncy
.....................................

val offset: Dp by animateDpAsState(
    targetValue = circleOffset.dp,
    animationSpec =if (circleOffset==endOffset) {
        spring(dampingRatio = DampingRatioHighBouncy)     // сильный отскок
    } else {
        spring(dampingRatio = DampingRatioNoBouncy)     // отсутствие отскока
    }
)
```

Аналогичным образом можно использовать другие параметры функции spring(). Например, установим жесткость:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.Spring.DampingRatioMediumBouncy
import androidx.compose.animation.core.Spring.DampingRatioNoBouncy
import androidx.compose.animation.core.Spring.StiffnessHigh
import androidx.compose.animation.core.Spring.StiffnessVeryLow
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.animation.core.spring
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalConfiguration
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val circleHeight = 150
            val startOffset = 10
            val endOffset = LocalConfiguration.current.screenHeightDp -
circleHeight
            var circleOffset by remember { mutableStateOf(startOffset)}
            val offset: Dp by animateDpAsState(
                targetValue = circleOffset.dp,
                animationSpec =if (circleOffset==endOffset) {
                    spring(dampingRatio = DampingRatioMediumBouncy, stiffness =
StiffnessVeryLow)  // жесткость StiffnessVeryLow
                } else {
                    spring(dampingRatio = DampingRatioNoBouncy)
```

```
                }
            )
            val offset1: Dp by animateDpAsState(
                targetValue = circleOffset.dp,
                animationSpec =if (circleOffset==endOffset) {
                    spring(dampingRatio = DampingRatioMediumBouncy, stiffness =
StiffnessHigh)  // жесткость StiffnessHigh
                } else {
                    spring(dampingRatio = DampingRatioNoBouncy)
                }
            )
            Row(Modifier.fillMaxSize()) {
                println("Offset: ${offset.value}")
                Button({circleOffset = if (circleOffset==startOffset) endOffset
else startOffset },
                    Modifier.padding(10.dp)) {
                    Text("Start", fontSize = 22.sp)
                }

Box(Modifier.padding(top=offset).size(circleHeight.dp).clip(CircleShape).backgroun
d(Color.DarkGray))

Box(Modifier.padding(top=offset1).size(circleHeight.dp).clip(CircleShape).backgrou
nd(Color.DarkGray))
            }
        }
    }
}
```

Функция keyframes и анимация по ключевым кадрам

Ключевые кадры (keyframes) позволяют применять различные значения длительности и замедления в определенных точках временной шкалы анимации. Ключевые кадры применяются к анимации через параметр animationSpec и определяются с помощью функции keyframes():

```
public fun <T> keyframes(
    init: KeyframesSpec.KeyframesSpecConfig<T>.() -> Unit
): KeyframesSpec<T>
```

Эта функция возвращает объект KeyframesSpec принимает другую функцию, которая содержит данные о ключевых кадрах.

Определение анимации по ключевым кадрам содержит свойства durationMillis (общее время анимации) и delayMillis (задержка анимации - необязательное свойство), а также определения ключевых кадров. Каждый ключевой кадр содержит метку времени, которая указывает, какая часть общей анимации должна быть завершена в этот момент в зависимости от типа единицы состояния

(например, Float, Dp, Int и т. д.). Эти временные метки создаются посредством вызовов функции at(). Например:

```
animationSpec = keyframes {

        durationMillis = 1000
        100.dp.at(10)
        110.dp.at(500)
        200.dp.at(700)
    }
```

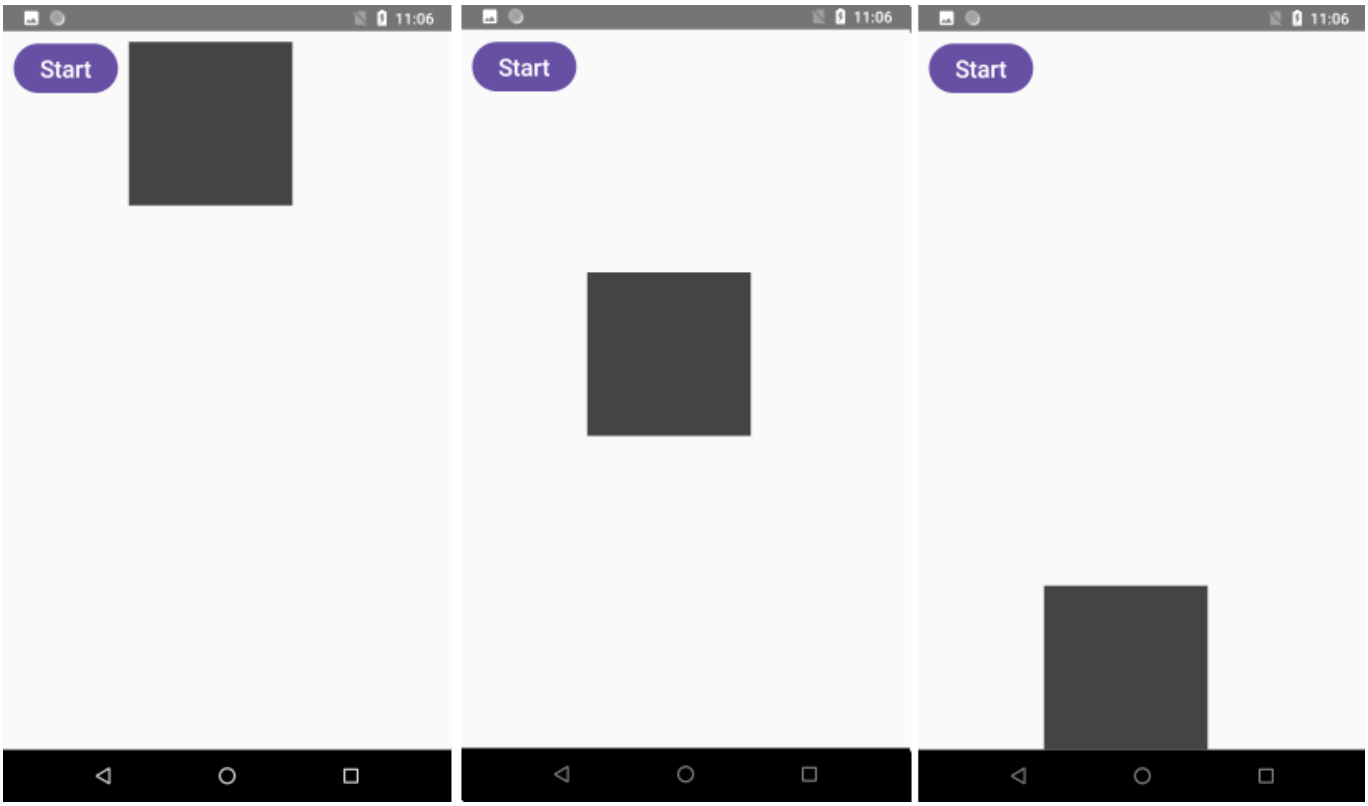
Здесь общее время анимации (durationMillis) 1000 миллисекунд. Для этой анимации задается три ключевых кадра. К примеру, первый кадр 100.dp.at(10) указывает, что смещение в 100.dp надо достигнуть через 10 миллисекунд. При 500 миллисекундах смещение должно составлять 110dp и, наконец, 200dp по истечении 700 миллисекунд. Это оставляет 300 миллисекунд для завершения оставшейся анимации.

Рассмотрим применение анимации по ключевым кадрам на следующем примере:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.animateDpAsState
import androidx.compose.animation.core.keyframes
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalConfiguration
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
```

```
            val boxHeight = 150         // высота Box
            val startOffset = 10        // начальный отступ
            val endOffset = LocalConfiguration.current.screenHeightDp - boxHeight
// конечный отступ
            var boxState by remember { mutableStateOf(startOffset)}

            val offset: Dp by animateDpAsState(
                targetValue = boxState.dp,
                animationSpec = keyframes {
                    durationMillis = 1000
                    if (boxState==endOffset) {
                        100.dp.at(100)
                        110.dp.at(500)
                        200.dp.at(800)
                    }
                }
            )

            Row(Modifier.fillMaxSize()) {
                println("Offset: ${offset.value}")
                Button({boxState = if (boxState==startOffset) endOffset else
startOffset },
                    Modifier.padding(10.dp)) {
                    Text("Start", fontSize = 22.sp)
                }

Box(Modifier.padding(top=offset).size(boxHeight.dp).background(Color.DarkGray))
            }
        }
    }
}
```

Здесь мы анимируем движение компонента Box по вертикали сверху вниз и обратно:



Для целей анимации определяем ряд переменных, как начальная и конечная позиция круга и его диаметр (высота)

```
val boxHeight = 150          // высота Box
val startOffset = 10         // начальный отступ
val endOffset = LocalConfiguration.current.screenHeightDp - boxHeight   //
конечный отступ
```

Далее определем состояние, которое будет хранить текущий текущий отступ с верху:

```
var boxState by remember { mutableStateOf(endOffset)}
```

Затем создаем анимацию с помощью функции animateDpAsState(), которая будет определять отступ:

```
val offset: Dp by animateDpAsState(
    targetValue = boxState.dp,
    animationSpec =
        keyframes {
            durationMillis = 1000
            if (boxState==endOffset) {
                100.dp.at(100)
                110.dp.at(500)
                200.dp.at(800)
            }
        }
)
```

Здесь прежде всего определяем targetValue - значение, к которому надо перейти в процессе анимации. Поскольку при нажати на кнопку изменяется boxState, то именно эту переменная и будет хранить следующее состояние, к которому надо выполнить анимационный переход. Соответственно просто передаем параметру targetValue значение этой переменной, преобразуя его в объект Dp.

Параметру animationSpec передаем результат функции keyframes(). В ней устанавливаем общее время анимации - 1000 миллисекунд. И если надо перейти к конечному отступу (в boxState хранится endOffset), то есть box находится вверху, то задаем три ключевых кадра. Если Box находится внизу, то ключевые кадры отсутствуют, а Box будет двигаться на начальную позицию равномерно.

Для запуска анимации определяем кнопку, которая меняет значение boxState и тем самым вызывает анимацию:

```
Button({boxState = if (boxState==startOffset) endOffset else startOffset },
```

И в конце идет собственно компонент Box, у которого анимируется отступ сверху:

```
Box(Modifier.padding(top=offset).size(boxHeight.dp).background(Color.DarkGray))
```

Его значение отступа с верху привязано к переменной offset, которая генерируется функцией animateDpAsState()

Стоит отметить, что в качестве альтернативы для установки ключевых кадров в функции keyframes() мы могли бы использовать другой синтаксис, где функция at() применялась бы аналогично операторам:

```
animationSpec = keyframes {

    durationMillis = 1000
    if (boxState==endOffset) {
        100.dp at 100
        110.dp at 500
        200.dp at 800
    }
}
```

Анимация цвета. animateColorAsState

Функция animateColorAsState() анимирует значение типа Color, то есть цвет. Она имеет следующие параметры:

```
@Composable
public fun animateColorAsState(
```

```
    targetValue: Color,
    animationSpec: AnimationSpec<Color> = colorDefaultSpring,
    label: String = "ColorAnimation",
    finishedListener: ((Color) -> Unit)? = null
): State<Color>
```

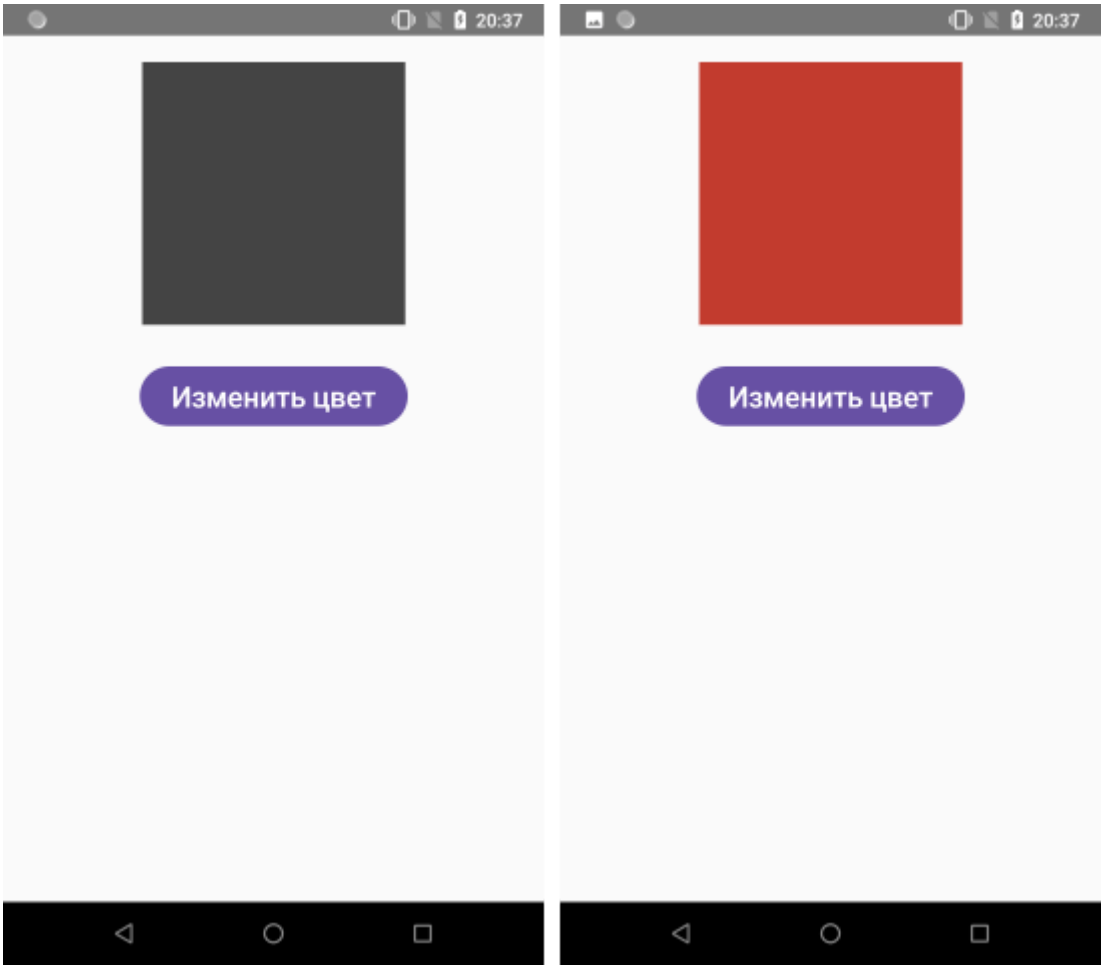
- targetColor: целевой цвет, к которому надо выполнить переход
- animationSpec: применяемая анимация в виде объекта AnimationSpec
- label: название анимации
- finishedListener: функция, которая выполняется при завершении анимации

Рассмотрим следующий пример:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.animateColorAsState
import androidx.compose.animation.core.tween
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var colorState by remember { mutableStateOf(Color.DarkGray) }
            val animatedColor: Color by animateColorAsState(
                targetValue = colorState,
                animationSpec = tween(5000)
            )
```

```
            Column(Modifier.fillMaxWidth(), horizontalAlignment =
Alignment.CenterHorizontally) {

Box(Modifier.padding(20.dp).size(200.dp).background(animatedColor))
                Button(
                    {colorState = if (colorState == Color.Red) {Color.DarkGray}
else {Color.Red}},
                    Modifier.padding(10.dp)) {
                    Text("Изменить цвет", fontSize = 22.sp)
                }
            }
        }
    }
}
```

Здесь происходит анимация цвета компонента Box от красного к темно-серому и наборот.



Для этого вначале определяем состояние, которое будет хранить текущий цвет:

```
var colorState by remember { mutableStateOf(Color.DarkGray) }
```

По умолчанию это темно-сервый цвет. Далее определяем саму анимацию цвета:

```
val animatedColor: Color by animateColorAsState(
    targetValue = colorState,
    animationSpec = tween(5000)
)
```

Параметр targetValue указывает на значение, к которому надо перейти. И здесь мы просто передаем значение colorState, то есть текущий цвет. Когда произойдет нажатие на кнопку, и изменится значение colorState, то targetValue получит новое значение. И функция выполнит переход к новому целевому цвету.

С помощью параметра animationSpec определяем настройки анимации. В данном случае с помощью функции tween() задаем время выполнения анимации - 5000 миллисекунд. То есть переход от одного цвета к другому будет выполняться 5 секунд.

Функция animateColorAsState() возвращает объект типа `State<Color>`, а с помощью оператора by получаем из него сам объект Color. Для демонстрации определен компонент Box, фон которого привязан к цвету, полученному из функции animateColorAsState

```
Box(Modifier.padding(20.dp).size(200.dp).background(animatedColor))
```

А с помощью кнопки переключаем цвет в colorState:

```
Button(
    {colorState = if (colorState == Color.Red) {Color.DarkGray} else {Color.Red}},
    Modifier.padding(10.dp)) {
    Text("Изменить цвет", fontSize = 22.sp)
}
```

В примере выше специафикация анимации устанавливалась с помощью функции tween(), но также можно использовать и другие функции, которые возвращают объект AnimationSpec. Например, применим функцию keyframes() для анимации по ключевым кадрам:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.animateColorAsState
import androidx.compose.animation.core.keyframes
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
```

```
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.compose.ui.Alignment


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var colorState by remember { mutableStateOf(Color.DarkGray) }
            val animatedColor: Color by animateColorAsState(
                targetValue = colorState,
                animationSpec = keyframes {
                    durationMillis = 3000
                    Color.Blue at 500
                    Color.Green at 1500
                    Color.Yellow at 2500
                }
            )

            Column(Modifier.fillMaxWidth(), horizontalAlignment =
Alignment.CenterHorizontally) {

Box(Modifier.padding(20.dp).size(200.dp).background(animatedColor))
                Button(
                    {colorState = if (colorState == Color.Red) {Color.DarkGray}
else {Color.Red}},
                    Modifier.padding(10.dp)) {
                    Text("Изменить цвет", fontSize = 22.sp)
                }
            }
        }
    }
}
```

В данном случае цвет компонента Box опять же изменяется с темно-серого на красный, однако в течение всего времени анимации (3 секунды) он также получает промежуточные цвета, которые описываются ключевыми кадрами анимации:

```
animationSpec = keyframes {
    durationMillis = 3000
    Color.Blue at 500
```

```
    Color.Green at 1500
    Color.Yellow at 2500
}
```

Таким образом, через 500 миллисекунд после начала анимации компонент Box получит цвет Color.Blue, через 1500 миллисекунд - Color.Green, и через 2500 миллисекунд после начала анимации - цвет Color.Yellow.

Анимация числовых значений и animateFloatAsState

Функция animateFloatAsState() применяется для анимации значений типа Float. Она имеет следующие параметры:

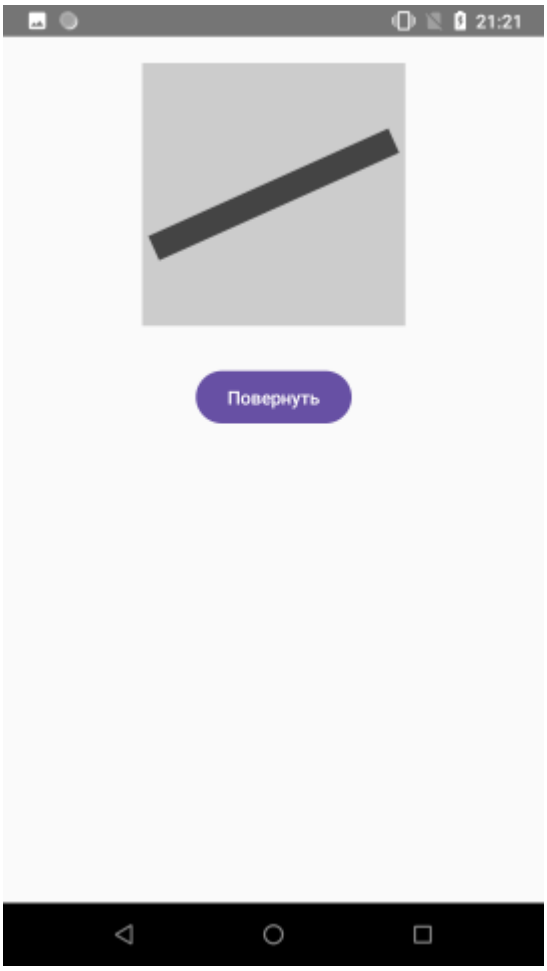
```
@Composable
public fun animateFloatAsState(
    targetValue: Float,
    animationSpec: AnimationSpec<Float> = defaultAnimation,
    visibilityThreshold: Float = 0.01f,
    label: String = "FloatAnimation",
    finishedListener: ((Float) -> Unit)? = null
): State<Float>
```

Здесь фактически те же самые параметры, что и в animateColorAsState() или animateDpAsState(), только вместо типа Color/Dp применяется тип Float. Например, параметр targetValue указывает на число, к которому надо выполнить переход. Так, возьмем следующий код:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.animateFloatAsState
import androidx.compose.animation.core.tween
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
```

```
import androidx.compose.ui.draw.rotate
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var rotated by remember { mutableStateOf(false) }
            val angle by animateFloatAsState(
                targetValue = if (rotated) 360f else 0f,
                animationSpec = tween(4000)
            )
            Column(Modifier.fillMaxWidth(), horizontalAlignment =
Alignment.CenterHorizontally) {

Box(Modifier.padding(20.dp).size(200.dp).background(Color.LightGray).rotate(angle)
, Alignment.Center){

Box(Modifier.size(width=200.dp,height=20.dp).background(Color.DarkGray))
                }
                Button({rotated = !rotated},
                    Modifier.padding(10.dp)) {
                    Text(text = "Повернуть")
                }
            }
        }
    }
}
```

В данном случае суть приложения заключается в повороте компонента с помощью анимации Float:



Для поворота вначале определяем состояние:

```
var rotated by remember { mutableStateOf(false) }
```

Это значение указывает, в какую сторону вращать компонент. Далее на основе этого состояния определяем анимацию:

```
val angle by animateFloatAsState(
    targetValue = if (rotated) 360f else 0f,
    animationSpec = tween(4000)
)
```

Если rotated равно true, то целевое значение анимации устанавливается на 360 градусов, в противном случае оно устанавливается на 0. Если rotated равно true, то производится вращение на 360 градусов. Результат анимации получаем в переменную angle.

Затем передаем значение angle в модификатор rotate(), который принимает угол поворота:

```
Box(Modifier.padding(20.dp).size(200.dp).background(Color.LightGray).rotate(angle)
, Alignment.Center){
    Box(Modifier.size(width=200.dp,height=20.dp).background(Color.DarkGray))
}
```

И для запуска анимации определяем кнопку, по нажатию на которую переключается значение rotated, что, в свою очередь, приведет к изменению угла вращения и анимации:

```
Button({rotated = !rotated},
```

Объединение анимаций

Иногда возникает необходимость запустить не одну, а сразу несколько анимаций. Для этой цели в Jetpack Compose применяется Transition, к которому можно добавить несколько дочерних анимаций и запустить их одновременно.

Для создания объекта

который по Несколько анимаций могут запускаться параллельно на основе одного целевого состояния с помощью функции updateTransition(). Эта функция передает целевое состояние и возвращает объект Transition, к которому можно добавить несколько дочерних анимаций. Функция updateTransition() двольно проста:

```
@Composable
public fun <T> updateTransition(
    targetState: T,
    label: String? = null
): Transition<T>
```

В качестве обязательного параметра targetState функции надо передать целевое состояние. Когда целевое состояние изменится, объект Transition запустит все дочерние анимации одновременно. Также может быть передан необязательный параметр label, который можно использовать для идентификации перехода в Animation Inspector в Android Studio.

Класс Transition включает набор функций, которые используются для добавления анимаций. Эти функции называются по шаблону animate[Тип], где [Тип] представляет тип анимируемого значения, например, animateFloat(), animateDp() и animateColor(). Синтаксис этих функций следующий:

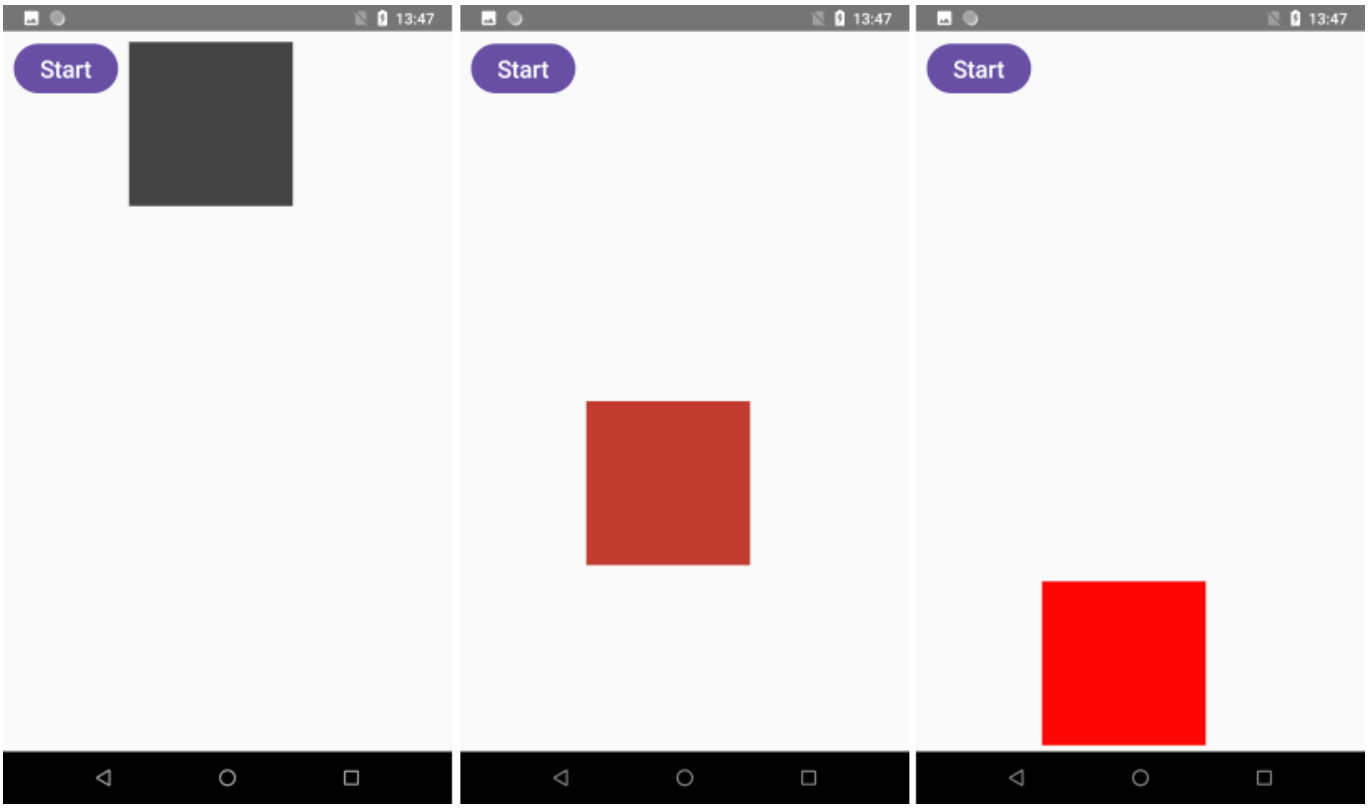
```
val myAnimation by transition.animate[Тип](

    transitionSpec = {

        // определение анимации (с помощью функций tween, spring и т.д.)
    }
) { state ->

    // код вычисления состояния, к которому надо перейти
}
```

Рассмотрим простейший пример применения переходов Transition:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.animateColor
import androidx.compose.animation.core.animateDp
import androidx.compose.animation.core.tween
import androidx.compose.animation.core.updateTransition
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalConfiguration
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val boxHeight = 150
            val startOffset = 10
            val endOffset = LocalConfiguration.current.screenHeightDp - boxHeight
            var boxState by remember { mutableStateOf(startOffset)}
            val transition = updateTransition(targetState = boxState)

            // определяем анимацию цвета
            val animatedColor: Color by transition.animateColor(
                transitionSpec = { tween(4000)} // длительность анимации - 4
секунды
            ) {
                state -> if(state == endOffset) Color.Red else Color.DarkGray
            }

            // определяем анимацию отступов
            val animatedOffset: Dp by transition.animateDp(
```

```
                transitionSpec = {tween(4000)}
            ) { state -> state.dp  }

            Row(Modifier.fillMaxSize()) {
                Button({boxState = if (boxState==startOffset) endOffset else
startOffset },
                    Modifier.padding(10.dp)) {
                    Text("Start", fontSize = 22.sp)
                }

Box(Modifier.padding(top=animatedOffset).size(boxHeight.dp).background(animatedCol
or))
            }
        }
    }
}
```

Здесь мы объединяем две анимации - анимацию цвета и анимацию по Dp (анимируем отступ сверху компонента Box)



Вначале определяем переменные для вычисления состояния компонента Box:

```
val boxHeight = 150
val startOffset = 10
val endOffset = LocalConfiguration.current.screenHeightDp - boxHeight
```

Переменная boxHeight определяет высоту компонента Box. Переменная startOffset представляет начальное смещение сверху. endOffset представляет конечное смещение сверху, которое зависит от

свойства LocalConfiguration.current.screenHeightDp. То есть фактически это самый низ экрана.

Далее определяется состояние компонента Box, которое будет отслеживаться для запуска анимации:

```
var boxState by remember { mutableStateOf(var boxState by remember {
mutableStateOf(startOffset)})}
```

По умолчанию это состояние равно startOffset.

Дальше идет создание объекта Transition:

```
val transition = updateTransition(targetState = boxState)
```

Объект Transtion будет отслеживать изменение состояния в boxState. И затем для этого объекта определяем анимации. Сначала определяется анимация цвета с помощью метода animateColor объекта Transition:

```
val animatedColor: Color by transition.animateColor(
    transitionSpec = { tween(4000)} // длительность анимации - 4 секунды
) {
    state -> if(state == endOffset) Color.Red else Color.DarkGray
}
```

Прежде всего через параметр transitionSpec задаем настройки анимации. Здесь с помощью вызова функции tween() устанавливаем время анимации - 4 секунды.

В фигурных скобках в лямбда-выражение через параметр state передается текущее состояние. state - это то состояние, которое отслеживает объект Transition и в данном случае это текущее значение boxState. В лямбда-выражении возвращаем значение, которое будет применяться в качестве нового состояния. И в итоге для перехода от текущего к новому состоянию и будет выполняться анимация. В данном случае пусть, если компонент Box прошел около половину пути, то есть соблюдается условие state == endOffset, то Box получает красный цвет, иначе получает темно-серый цвет.

Затем определяется анимация отступов:

```
val animatedOffset: Dp by transition.animateDp(
    transitionSpec = {tween(4000)}
) { state -> state.dp  }
```

В нашем случае boxState хранит одно из значение - startOffset или endOffset, который представляют отступ. В данном случае просто преобразуем значение в объект типа Dp и возвращаем его.

Для запуска анимации определена кнопка, которая переключает значение в boxState со startOffset на endOffset и обратно:

```
Button({boxState = if (boxState==startOffset) endOffset else startOffset },
```

И в конце определен собственно тот компонент Box, к которому и применяются анимации:

```
Box(Modifier.padding(top=animatedOffset).size(boxHeight.dp).background(animatedCol
or))
```

Его модификатору padding() для параметра top передается результат функции transition.animateDp(), а в модификатор background() - результат функции transition.animateColor

AnimatedVisibility. Управление видимостью компонента

Компонент AnimatedVisibility управляет видимостью вложенного содержимого и позволяет анимировать его появление и исчезновение, сделать переход к видимой или невидимой форме более плавным. AnimatedVisibility имеет следующие параметры:

```
@Composable
fun AnimatedVisibility(
    visible: Boolean,
    modifier: Modifier = Modifier,
    enter: EnterTransition = fadeIn() + expandIn(),
    exit: ExitTransition = shrinkOut() + fadeOut(),
    label: String = "AnimatedVisibility",
    content: @Composable AnimatedVisibilityScope.() -> Unit
): Unit
```

- visible: определяет, видимо ли вложенное содержимое (если true. то видимое)
- modifier: применяемые к компоненту функции-модификаторы
- enter: настройки перехода при появлении содержимого на экране
- exit: настройки перехода при исчезновении содержимого с экрана
- content: функция типа AnimatedVisibilityScope.() -> Unit, которая устанавливает вложенное содержимое

Этот компонент также имеет еще одну версию:

```
@Composable
fun AnimatedVisibility(
    visibleState: MutableTransitionState<Boolean>,
    modifier: Modifier = Modifier,
```

```
    enter: EnterTransition = fadeIn() + expandIn(),
    exit: ExitTransition = fadeOut() + shrinkOut(),
    label: String = "AnimatedVisibility",
    content: @Composable AnimatedVisibilityScope.() -> Unit
): Unit
```

Единственное отличие от предыдущей версии состоит в том, что здесь для управления видимостью применяется параметр visibleState, который представляет значение типа MutableTransitionState

Итак, компонент AnimatedVisibility управляет видимостью вложенного содержимого. Рассмотрим простейший пример:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            val boxVisible = remember { mutableStateOf(true) }
            Column(Modifier.padding(20.dp), horizontalAlignment =
Alignment.CenterHorizontally) {

                AnimatedVisibility(visible = boxVisible.value,
Modifier.padding(40.dp)) {
                    Box(Modifier.size(200.dp).background(Color.DarkGray))
                }
                Row(Modifier.fillMaxWidth(),horizontalArrangement =
Arrangement.SpaceEvenly) {
```

```
                    Button({boxVisible.value = true}, enabled = !boxVisible.value)
{ Text("Show", fontSize = 28.sp)}
                    Button({boxVisible.value = false}, enabled = boxVisible.value)
{ Text("Hide", fontSize = 28.sp)}
                }
            }
        }
    }
}
```

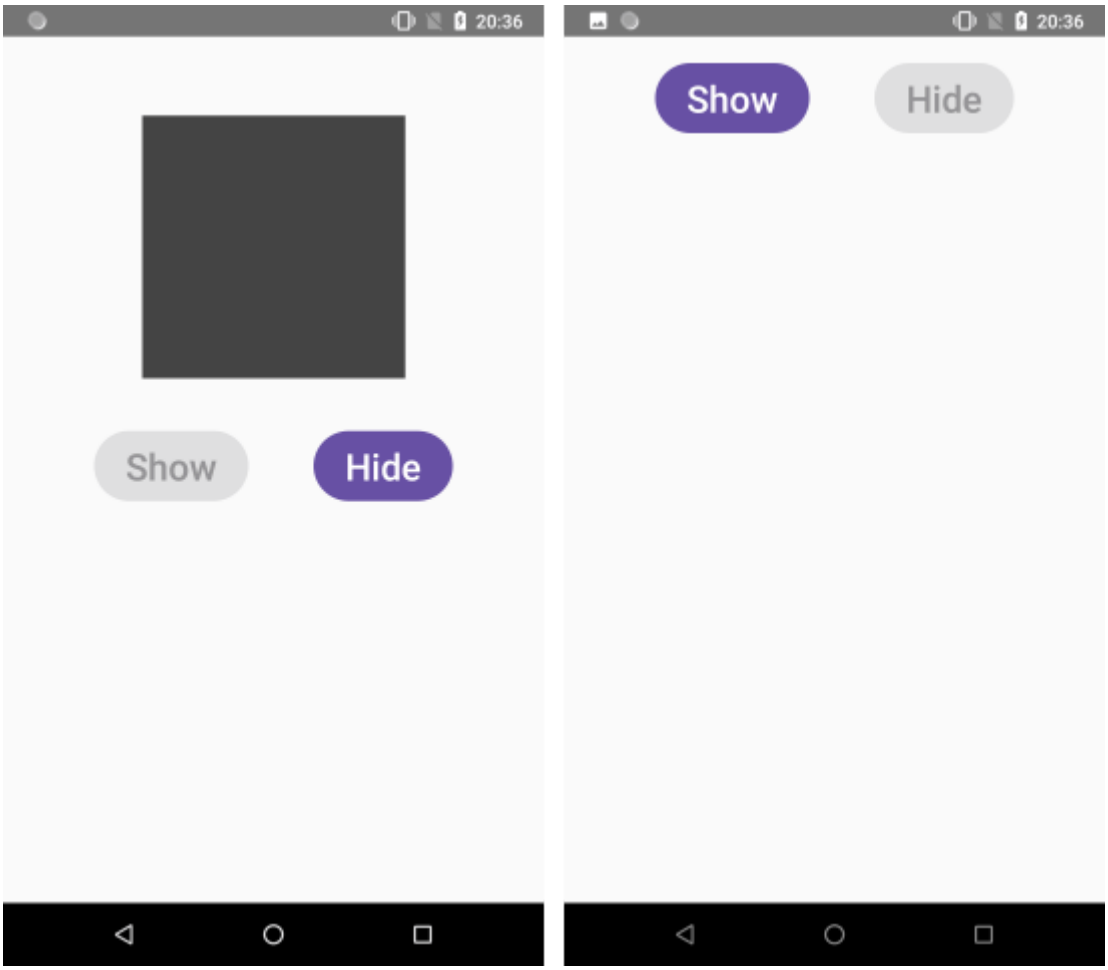
Здесь прежде всего для управления видимостью определяем состояние boxVisible:

```
val boxVisible = remember { mutableStateOf(true) }
```

AnimatedVisibility содержит компонент Box и соответственно управляет видимостью этого компонента. Для управления видимостью параметр visible привязан к состоянию boxVisible

```
AnimatedVisibility(visible = boxVisible.value, Modifier.padding(40.dp)) {
    Box(Modifier.size(200.dp).background(Color.DarkGray))
}
```

С помощью кнопок переключаем видимость компонента:



При этом появление и исчезновение комопнента Box будет происходить не многовенно, а плавно. И именно в это плавности и суть компонента, потому что само переключение видимости (мгновенное) мы могли бы сделать и без AnimatedVisibility, например, следующим образом:

```
val boxVisible = remember { mutableStateOf(true) }
Column(Modifier.padding(20.dp), horizontalAlignment =
Alignment.CenterHorizontally, verticalArrangement = Arrangement.SpaceBetween) {

    if(boxVisible.value) {
        Box(Modifier.size(200.dp).padding(40.dp).background(Color.DarkGray))
    }
    Row(Modifier.fillMaxWidth(),horizontalArrangement = Arrangement.SpaceEvenly) {
        Button({boxVisible.value = true}, enabled = !boxVisible.value){
Text("Show", fontSize = 28.sp)}
        Button({boxVisible.value = false}, enabled = boxVisible.value){
Text("Hide", fontSize = 28.sp)}
    }
}
```

Настройка анимации в AnimatedVisibility

Для управления анимацией при появлении и исчезновении вложенных компонентов в AnimatedVisibility применяются параметры enter и exit. Этим параметрам передается одна из функций, которые создают эффект анимации. В частности, это следующие функции:

- expandHorizontally(): содержимое отображается с использованием метода горизонтального отсечения. Позволяет указать, какая часть контента изначально отображается до начала анимации.
- expandVertically(): содержимое отображается с использованием техники вертикального отсечения. Позволяет указать, какая часть контента изначально отображается до начала анимации.
- expandIn(): содержимое отображается с использованием методов горизонтального и вертикального отсечения. Позволяет указать, какая часть контента изначально отображается до начала анимации.
- fadeIn(): постепенно делает прозрачное содержимое непрозрачным. Начальная прозрачность может быть объявлена с использованием значения с плавающей запятой от 0 до 1.0. Значение по умолчанию - 0.
- fadeOut(): постепенно делает непрозрачное содержимое прозрачным (невидимым). Целевая прозрачность перед исчезновением содержимого может быть объявлена с использованием значения с плавающей запятой от 0 до 1.0. Значение по умолчанию - 0.
- scaleIn(): содержимое постепенно увеличивается. По умолчанию содержимое начинается с нулевого размера и расширяется до полного размера, хотя это значение по умолчанию можно изменить, указав начальное значение масштаба как плавающее значение от 0 до 1.0.

- scaleOut(): содержимое постепенно уменьшается и в конце полностью исчезает. По умолчанию целевой масштаб равен 0, но его можно настроить с использованием плавающего значения от 0 до 1.0.
- shrinkHorizontally(): компонент постепенно сживается по горизонтали, пока польностью не исчезнет.
- shrinkVertically(): компонент постепенно сживается по вертикали, пока польностью не исчезнет.
- shrinkOut(): ккомпонент постепенно сживается по горизонтали и вертикали, пока польностью не исчезнет.
- slideInHorizontally(): содержимое перемещается в область обзора вдоль горизонтальной оси
- slideInVertically(): содержимое перемещается в область обзора вдоль вертикальной оси.
- slideIn(): содержимое появляется в области обзора под настраиваемым углом, определяемым через начальное значение смещения.
- slideOut(): содержимое исчезает из поля зрения под настраиваемым углом, определяемым через начальное значение смещения.
- slideOutHorizontally(): содержимое выходит из области обзора вдоль горизонтальной оси.
- slideOutVertically(): содержимое выходит из поля зрения вдоль вертикальной оси

Например, применм пару из них:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.fadeIn
import androidx.compose.animation.slideOutVertically
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
```

```
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            val boxVisible = remember { mutableStateOf(true) }
            Column(Modifier.padding(20.dp), horizontalAlignment =
Alignment.CenterHorizontally) {
                AnimatedVisibility(
                    visible = boxVisible.value,
                    Modifier.padding(40.dp),
                    enter = fadeIn(),
                    exit = slideOutVertically()) {
                        Box(Modifier.size(200.dp).background(Color.DarkGray))
                }
                Row(Modifier.fillMaxWidth(),horizontalArrangement =
Arrangement.SpaceEvenly) {
                    Button({boxVisible.value = true}, enabled = !boxVisible.value)
{ Text("Show", fontSize = 28.sp)}
                    Button({boxVisible.value = false}, enabled = boxVisible.value)
{ Text("Hide", fontSize = 28.sp)}
                }
            }
        }
    }
}
```

В данном случае при появлении компонент Box будет постепенно отображаться, пока не станет полностью непрозрачным (видимым). А при исчезновении он будет склользить по вертикали вверх, как бы уходя из поля зрения:

```
enter = fadeIn(),
exit = slideOutVertically()
```

Стоит отметить, что с помощью операции сложения мы можем комбинировать несколько функций:

```
AnimatedVisibility(
    visible = boxVisible,
    enter = fadeIn() + expandHorizontally(),
    exit = slideOutVertically()
) {
```

Настройка времени и плавности анимации

Для настройки анимации вышеуказанные функции имеют параметр animationSpec, который представляет интерфейс AnimationSpec. Так, для установки времени и сглаживания анимации в

AnimatedVisibility может использоваться рассмотренная в одной из прошлых тем функция tween(). Например:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.core.tween
import androidx.compose.animation.fadeIn
import androidx.compose.animation.slideOutVertically
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            val boxVisible = remember { mutableStateOf(true) }
            Column(Modifier.padding(20.dp), horizontalAlignment =
Alignment.CenterHorizontally) {
                AnimatedVisibility(
                    visible = boxVisible.value,
                    Modifier.padding(40.dp),
                    enter = fadeIn(animationSpec = tween(durationMillis = 5000)),
                    exit = slideOutVertically()) {
                        Box(Modifier.size(200.dp).background(Color.DarkGray))
                }
                Row(Modifier.fillMaxWidth(),horizontalArrangement =
Arrangement.SpaceEvenly) {
                    Button({boxVisible.value = true}, enabled = !boxVisible.value)
{ Text("Show", fontSize = 28.sp)}
                    Button({boxVisible.value = false}, enabled = boxVisible.value)
{ Text("Hide", fontSize = 28.sp)}
                }
```

```
            }
        }
    }
}
```

В данном случае для анимации появления на экране применяется функция fadeIn()

```
enter = fadeIn(animationSpec = tween(durationMillis = 5000)),
```

То есть появление компонента на экране будет занимать 5000 миллисекунд.

Аналогичным образом можно настраивать и другие функции, например:

```
import androidx.compose.animation.core.LinearOutSlowInEasing
import androidx.compose.animation.slideInHorizontally

.................................
enter = slideInHorizontally(tween(durationMillis = 5000, easing =
LinearOutSlowInEasing)),
```

В данном случае компонент будет появляется области обзора, скользя вдоль горизонтальной оси в течение 5 секунд.

Или

```
import androidx.compose.animation.shrinkVertically
import androidx.compose.animation.slideInVertically
.......................................

AnimatedVisibility(
    visible = boxVisible.value,
    Modifier.padding(40.dp),
    enter =slideInVertically(tween(durationMillis = 5000, easing =
LinearOutSlowInEasing)),
    exit = shrinkVertically(tween(durationMillis = 5000, easing =
LinearOutSlowInEasing))) {
        Box(Modifier.size(200.dp).background(Color.DarkGray))
}
```

В данном случае компонент будет постепенно появляться в течение 5 секунд, выдвигаясь из-за вертикальной границы. И будет ужиматься вдоль вертикальной оси в течение 5 секунд.

Также для установки специафикации анимации можно использовать и другие функции, которые возвращают объект AnimationSpec. Например, используем функцию repeatable(), чтобы повторить анимацию 10 раз:

```
import androidx.compose.animation.core.RepeatMode
import androidx.compose.animation.core.repeatable
.....................................

AnimatedVisibility(
    visible = boxVisible.value,
    Modifier.padding(40.dp),
    enter =fadeIn(animationSpec =
        repeatable(10,
            animation = tween(durationMillis = 2000),
            repeatMode = RepeatMode.Reverse)
    ),
    exit = fadeOut()) {
        Box(Modifier.size(200.dp).background(Color.DarkGray))
}
```

Модификатор animateEnterExit()

Когда к компоненту AnimatedVisibility применяются анимации, они применяются ко всем вложенным компонентам. Однако специальная функция-модификатор animateEnterExit() позволяет определить анимацию на уровне отдельных компонентов. Как и в случае с AnimatedVisibility, этот модификатор позволяет объявлять анимацию появления и исчезновения компонента на экране. Но важно отметить, что этот модификатор применяется не сам по себе в любом месте пользовательского интерфейса, а только в рамках AnimatedVisibility.

Модификатор animateEnterExit() имеет следующее определение:

```
@ExperimentalAnimationApi
public open fun Modifier.animateEnterExit(
    enter: EnterTransition = fadeIn() + expandIn(),
    exit: ExitTransition = fadeOut() + shrinkOut(),
    label: String = "animateEnterExit"
): Modifier
```

Как и AnimatedVisibility, модификатор использует параметры enter и exit для определения анимации появления и исчезновения компонента на экране. Кроме того с помощью параметра label можно назначить текстовую метку. Поскольку данный модификатор является экспериментальным то при его использовании к компоненту следует применять аннотацию @ExperimentalAnimationApi

Пример применения модификатора animateEnterExit:

```
package com.example.helloapp

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

```
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.EnterTransition
import androidx.compose.animation.ExitTransition
import androidx.compose.animation.ExperimentalAnimationApi
import androidx.compose.animation.core.tween
import androidx.compose.animation.slideInVertically
import androidx.compose.animation.slideOutVertically
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent @ExperimentalAnimationApi{
            val boxVisible = remember { mutableStateOf(true) }
            Column(Modifier.padding(20.dp), horizontalAlignment =
Alignment.CenterHorizontally) {
                AnimatedVisibility(
                    visible = boxVisible.value,
                    Modifier.padding(40.dp),
                    enter = EnterTransition.None,
                    exit = ExitTransition.None) {
                    Box(Modifier.animateEnterExit(
                        enter = slideInVertically(tween(durationMillis = 5000)),
                        exit = slideOutVertically(tween(durationMillis = 5000))
                    ).size(200.dp).background(Color.DarkGray))
                }
                Row(Modifier.fillMaxWidth(),horizontalArrangement =
Arrangement.SpaceEvenly) {
                    Button({boxVisible.value = true}, enabled = !boxVisible.value)
{ Text("Show", fontSize = 28.sp)}
                    Button({boxVisible.value = false}, enabled = boxVisible.value)
{ Text("Hide", fontSize = 28.sp)}
                }
            }
        }
```

```
    }
}
```

Здесь модификатор применяется к компоненту Box, который расположен внутри AnimatedVisibility:

```
AnimatedVisibility(
    visible = boxVisible.value,
    Modifier.padding(40.dp),
    enter = EnterTransition.None,
    exit = ExitTransition.None) {
        Box(Modifier.animateEnterExit(
            enter = slideInVertically(tween(durationMillis = 5000)),
            exit = slideOutVertically(tween(durationMillis = 5000))
        ).size(200.dp).background(Color.DarkGray))
}
```

Видимость компонента AnimatedVisibility привязана к переменной-состоянию boxVisible, значение которой переключается кнопками.

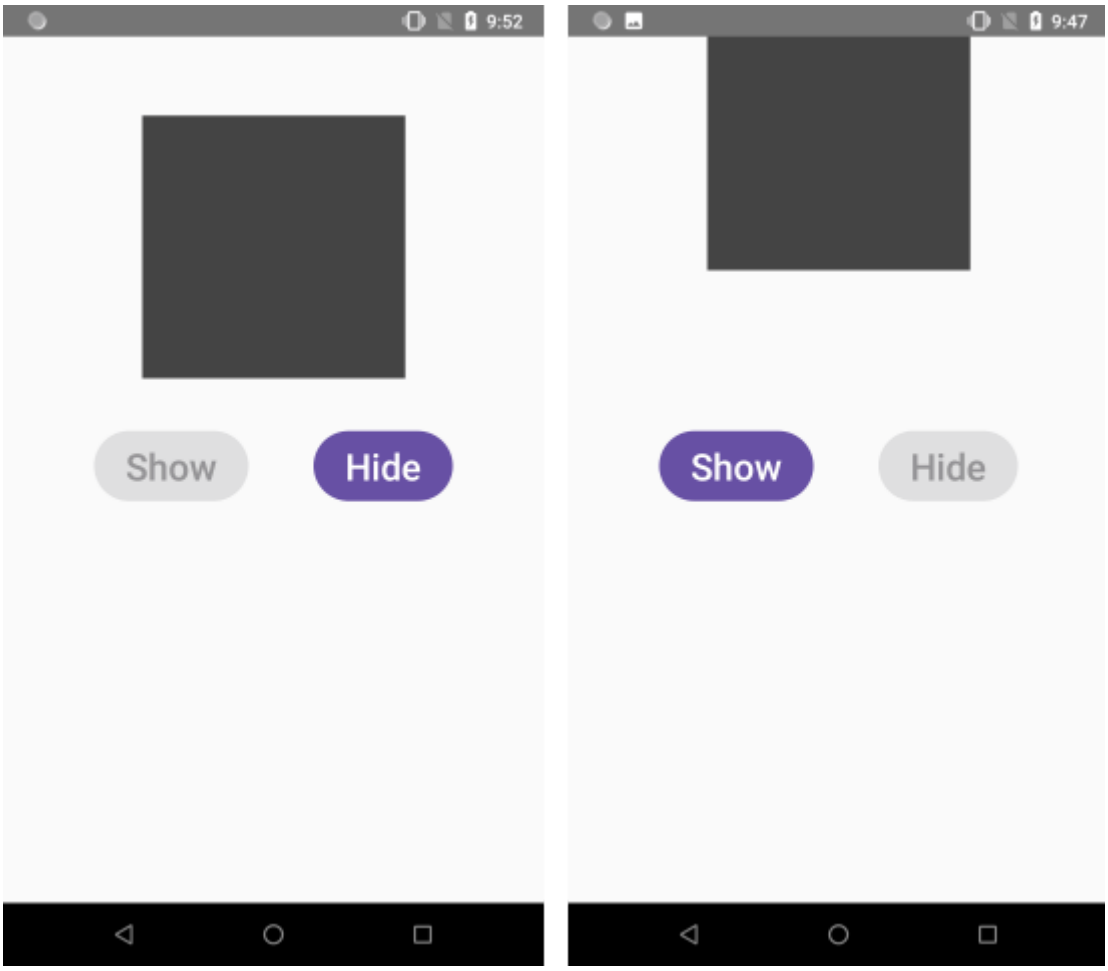
Поскольку для управления анимацией всецело используется модификатор animateEnterExit на компоненте Box, то для AnimatedVisibility наоборот отключаем анимацию:

```
enter = EnterTransition.None,
exit = ExitTransition.None
```

У компонента Box для анимации применяем функции slideInVertically и slideOutVertically

```
Box(Modifier.animateEnterExit(
    enter = slideInVertically(tween(durationMillis = 5000)),
    exit = slideOutVertically(tween(durationMillis = 5000))
)
```

Таким образом при появлении компонент Box начнет медленно скользить сверху вниз в течение 5 секунд, а при изчезновении, наоборот, подниматься снизу вверх, пока не уйдет из области обзора.



Аналогично можно было бы определить внутри AnimatedVisibility несколько компонентов с разными типами анимаций:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.EnterTransition
import androidx.compose.animation.ExitTransition
import androidx.compose.animation.ExperimentalAnimationApi
import androidx.compose.animation.core.tween
import androidx.compose.animation.fadeIn
import androidx.compose.animation.fadeOut
import androidx.compose.animation.slideInVertically
import androidx.compose.animation.slideOutVertically
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
```

```
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent @ExperimentalAnimationApi{
            val boxVisible = remember { mutableStateOf(true) }
            Column(Modifier.padding(20.dp), horizontalAlignment =
Alignment.CenterHorizontally) {
                AnimatedVisibility(
                    visible = boxVisible.value,
                    Modifier.padding(20.dp),
                    enter = EnterTransition.None,
                    exit = ExitTransition.None) {
                    Row{

                        Box(Modifier.animateEnterExit(
                            enter = slideInVertically(tween(durationMillis =
5000)),
                            exit = slideOutVertically(tween(durationMillis =
5000))
                        ).size(150.dp).background(Color.DarkGray))

                        Spacer(modifier = Modifier.width(20.dp))

                        Box(Modifier.animateEnterExit(
                            enter = fadeIn(tween(durationMillis = 5000)),
                            exit = fadeOut(tween(durationMillis = 5000))
                        ).size(150.dp).background(Color.DarkGray))
                    }
                }
                Row(Modifier.fillMaxWidth(),horizontalArrangement =
Arrangement.SpaceEvenly) {
                    Button({boxVisible.value = true}, enabled = !boxVisible.value)
{ Text("Show", fontSize = 28.sp)}
                    Button({boxVisible.value = false}, enabled = boxVisible.value)
{ Text("Hide", fontSize = 28.sp)}
                }
            }
        }
    }
}
```

В данном случае в AnimatedVisibility два компонента Box с разным набором анимаций. Первый Box для создания анимации использует функции slideInVertically() и slideOutVertically(), тогда как второй Box - функции fadeIn() и fadeOut()

Crossfade

Компонент Crossfade позволяет анимировать замену одного компонента на другой. Функция компонента Crossfade принимает следующие параметры:

```
@Composable
public fun <T> Crossfade(
    targetState: T,
    modifier: Modifier = Modifier,
    animationSpec: FiniteAnimationSpec<Float> = tween(),
    label: String = "Crossfade",
    content: @Composable (T) -> Unit
): Unit
```

- targetState: целевое состояние анимации. При каждом его изменении запускается анимация
- modifier: функции модификаторов, которые применяются к компоненту
- animationSpec: определение анимации
- label: текстовая метка анимации
- content: задает содержимое компонента

Рассмотрим небольшой пример:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.Crossfade
import androidx.compose.animation.ExperimentalAnimationApi
import androidx.compose.animation.core.tween
import androidx.compose.animation.fadeIn
import androidx.compose.animation.fadeOut
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
```

```
import androidx.compose.material3.Text
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent @ExperimentalAnimationApi{
            val boxVisible = remember { mutableStateOf(true) }
            Column(Modifier.fillMaxSize(), horizontalAlignment =
Alignment.CenterHorizontally) {

                AnimatedVisibility(
                    visible = boxVisible.value,
                    Modifier.padding(20.dp),
                    enter = fadeIn(tween(5000)),
                    exit = fadeOut(tween(5000))) {

                    Box(Modifier.size(150.dp).background(Color.DarkGray))
                }
                Crossfade(targetState = boxVisible, animationSpec = tween(5000)) {
visible ->
                    when (visible.value) {
                        true -> Button({boxVisible.value = false }){ Text("Hide",
fontSize = 28.sp)}
                        false -> Button({boxVisible.value = true }){ Text("Show",
fontSize = 28.sp)}
                    }
                }
            }
        }
    }
}
```

Здесь состояние для управления анимацией определено в виде переменной boxVisible:

```
val boxVisible = remember { mutableStateOf(true) }
```

Компонент AnimatedVisibility управляет видимостью вложенного компонента Box и для анимации использует функции fadeIn() и fadeOut():

```
AnimatedVisibility(
    visible = boxVisible.value,
```

```
    Modifier.padding(20.dp),
    enter = fadeIn(tween(5000)),
    exit = fadeOut(tween(5000))) {

    Box(Modifier.size(150.dp).background(Color.DarkGray))
}
```

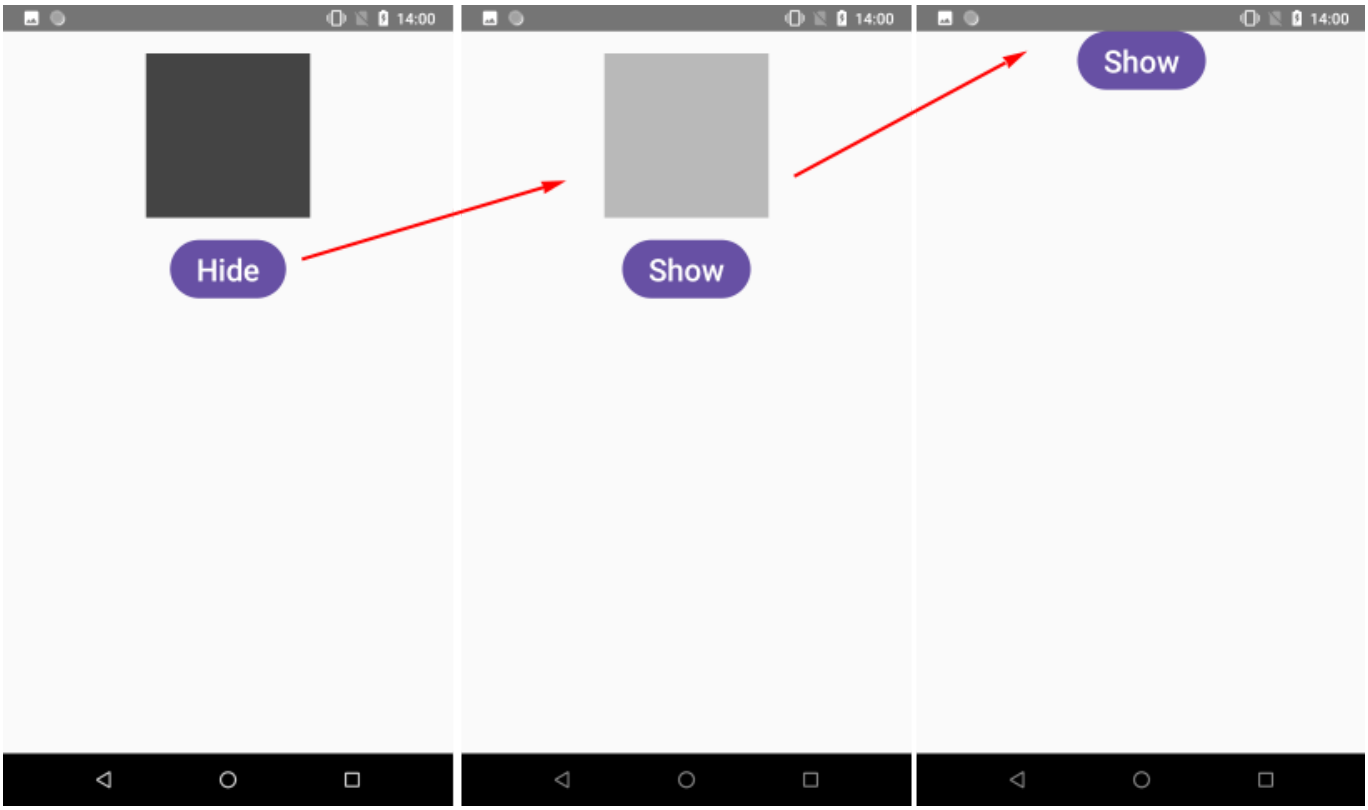
Затем идет вызов компонента Crossfade

```
Crossfade(targetState = boxVisible, animationSpec = tween(5000)) { visible ->
    when (visible.value) {
        true -> Button({boxVisible.value = false }){ Text("Hide", fontSize =
28.sp)}
        false -> Button({boxVisible.value = true }){ Text("Show", fontSize =
28.sp)}
    }
}
```

Здесь Crossfade привязан к значению переменной-состояния boxVisible и при ее изменении запускает анимацию. Параметр animationSpec в данном случае задает анимацию продолжительностью в 5000 миллисекунд.

Функция установки содержимого компонента Crossfade принимает один параметр - текущее значение состояния. В данном случае это параметр visible, но в реальности через этоот параметр передается значение boxVisible. А с помощью конструкции when мы решаем, при каком значении состояния отобразить тот или иной компонент. То есть если состояние равно true, то отображается кнопка Hide с кодом переключения состояния в false. А если состояние равно false, то отображается кнопка Show с кодом переключения состояния в true

В итоге при нажатии на кнопку произойдет изменение состояния boxVisible. Поскольку Crossfade привязан к boxVisible, то произойдет перерисовка Crossfade, и соответственно мы увидим новую кнопку.


14:00
Hide
14:00
Show
14:00
Show