Навигация

Введение в навигацию

Очень немногие мобильные приложения состоят только из одного экрана. На самом деле большинство приложений состоят из нескольких экранов, по которым пользователь перемещается с помощью экранных жестов, щелчков кнопок и выбора пунктов меню. Соответственно навигация является одним из ключевых моментов при построении приложения под Android.

Каждый экран обычно представляет собой отдельный компонент или объект Activity. Подобные экраны Google еще называет термином destination, то пункт назначения, к которому может перейти пользователь. Каждое приложение имеет главный экран (home screen), который появляется после запуска приложения и после появления любого экрана-заставки. На этом главном экране пользователь обычно выполняет задачи, которые приводят к появлению других экранов или пунктов назначения. Эти экраны обычно принимают форму других компонентов проекта. Например, приложение для обмена сообщениями может иметь главный экран со списком текущих сообщений, с которого пользователь может перейти на другой экран для доступа к списку контактов или экрану настроек. Экран списка контактов, в свою очередь, может позволить пользователю переходить к другим экранам, где можно добавлять новых пользователей или обновлять существующие контакты.

Для отслеживания переходов по различным экранам Android использует стек навигации. При первом запуске приложения начальный экран помещается в стек. Когда пользователь переходит к другому экрану, то этот экран помещается на верхушку стека. Когда пользователь переходит к другим экранам, они также помещаются в стек. Когда пользователь перемещается обратно по экранам с помощью системной кнопки "Назад", каждый компонент извлекается из стека до тех пор, пока главный экран не окажется единственным элементом в стеке. Извлекать элементы из стеа можно также программно.

Подключение функционала навигации в проект

По умолчанию проект не содержит функционала навигации, и нам надо добавить все необходимые зависимости. Для этого изменим файл libs.version.toml - в секцию [versions] версию подключаемой зависимости, а в секцию [libraries] имя подключемой библиотеки:

```
[versions]
navigationCompose = "2.7.7"
.................................


[libraries]
androidx-navigation-compose = { module = "androidx.navigation:navigation-compose",
version.ref = "navigationCompose" }
```

Затем в файл build.gradle.kts (Module :app) в секцию dependencies добавим следующую директиву:

```
dependencies {
    implementation(libs.androidx.navigation.compose)
.................................
```

Для синхронизации и обновления проекта нажмем на кнопку Sync Now

Создание хоста навигации

Для управления стеком навигации и самой навигацией применяется контроллер навигации, представленным классом NavHostController. И первым шагом при добавлении навигации в проект приложения является создание экземпляра NavHostController:

```
val navController = rememberNavController()
```

Для создания объекта NavHostController применяется функция rememberNavController(). Данная функция позволяет сохранить целостность стека навигации в процессе рекомпозиции компонентов пользовательского интерфейса.

После создания контроллера навигации его необходимо назначить хосту навигации - объекту NavHost. Хост навигации (NavHost) — это специальный компонент, который добавляется в макет пользовательского интерфейса действия и служит заполнителем для страниц, по которым будет перемещаться пользователь.

При вызове в NavHost передается объект NavHostController, компонент, который будет служить начальным экраном/начальным пунктом назначения, и граф навигации (navigation graph):

```
val navController = rememberNavController()
NavHost(navController = navController, startDestination = начальный_маршрут) {

    // Пункты назначения навигационного графа
}
```

Для передачи графа навигации применяется третий параметр - концевая лямбда. Граф навигации состоит из всех компонентов, которые должны быть доступны в качестве пунктов назначения навигации в контексте контроллера навигации. Пункты назначения добавляются в граф навигации путем вызова функции composable(), в который передается маршрут (route) и пункт назначения. Маршрут представляет обычную строку, которая однозначно идентифицирует пункт назначения в контексте текущего контроллера навигации. Пункт назначения — это компонент, который будет вызываться при выполнении навигации. Например:

```
NavHost(navController = navController, startDestination = "home") {

    composable("home") {
        Home()
```

```
    }


    composable("contact") {
        Contacts()
    }

    composable("about") {
        About()
    }
}
```

Здесь NavHost включает граф навигации, состоящий из трех пунктов назначения, где маршрут "home" настроен как начальный пункт назначения (начальный экран).

В качестве альтернативы жесткому кодированию строк маршрута в вызовах функции composable() можно определять маршруты в закрытом классе:

```
sealed class Routes(val route: String) {

    object Home : Routes("home")
    object Contacts : Routes("contact")
    object About : Routes("about")
}

NavHost(navController = navController, startDestination = Routes.Home.route) {

    composable(Routes.Home.route) { Home() }
    composable(Routes.Contacts.route) { Contacts()  }
    composable(Routes.About.route) { About() }
}
```

Использование закрытого класса позволяет определить единую точку для управления маршрутами.

Переход к точкам назначения

Для перехода между точками назначения/экранами применяется метод navigation() контроллера навигации. Данный метод определяет маршрут для пункта назначения. Например, в следующем коде по нажатию на кнопку происходит перехода к экрану "Contacts":

```
Button(onClick = {

    navController.navigate(Routes.Contacts.route)
}) {

    Text(text = "Contacts")
}
```

Метод navigation() также с помощью концевой лямбды позволяет задать параметры навигации с помощью различных функций. Так, функция popUpTo() позволяет очистить стек навигации вплоть до определенного элемента. Это упрощает навигацию, когда пользователь хочет вернуться не просто назад к экрану, который хранится на верхушке стека навигации, а в какое-то определенное место, например, на начальный экран. В этом случае мы могли бы сделать следующим образом:

```
Button(onClick = {

    navController.navigate(Routes.Contacts.route) {

        popUpTo(Routes.Home.route)
    }

}) {

    Text(text = "To Contact Page")
}
```

Теперь, когда пользователь нажимает кнопку "To Contact Page", в стек очищается вплоть до главного экрана Home. И после посещения экрана Contacts при возвращении назад пользователь перейдет к экрану Home.

Для определения начального пункта назначения также можно использовать метод navController.graph.findStartDestination(), а очистка стека вплоть до этого пункта будет представлять следующий вызов popUpTo:

```
popUpTo(navController.graph.findStartDestination().id)
```

Функция popUpTo() также может принимать дополнительные параметры. Например, параметр inclusive указывает, надо ли извлечь из стека навигации также и тот пункт назначения, который передан в функцию. Так, значение inclusive = true в следующем коде также извлекает из стека навигации пункт назначения Home перед выполнением навигации:

```
Button(onClick = {

    navController.navigate(Routes.Contacts.route) {

        popUpTo(Routes.Home.route) {

            inclusive = true
        }
    }
}) {

    Text(text = "To Contact Page")
}
```

По умолчанию попытка перехода от текущего пункта назначения к самому себе помещает в стек навигации дополнительный экземпляр пункта назначения. В большинстве ситуаций такое поведение вряд ли будет желательным. Чтобы этого предотвратить, для параметра launchSingleTop устанавливается значение true:

```
Button(onClick = {

    navController.navigate(Routes.Contacts.route) {

        launchSingleTop = true
    }
}) {

    Text(text = "To Contact Page")
}
```

Если для параметров saveState и restoreState установлено значение true, они будут автоматически сохранять и восстанавливать состояние записей стека, когда пользователь повторно выбирает пункт назначения, который был выбран ранее.

Пример навигации

Для рассмотрения навигации определим следующее простейшее приложение:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.clickable
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.material3.Text
import androidx.compose.runtime.Composable
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.navigation.NavController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController


class MainActivity : ComponentActivity() {
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Main()
        }
    }
}
@Composable
fun Main() {
    val navController = rememberNavController()
    Column(Modifier.padding(8.dp)) {
        NavBar(navController = navController)
        NavHost(navController, startDestination = NavRoutes.Home.route) {
            composable(NavRoutes.Home.route) { Home() }
            composable(NavRoutes.Contacts.route) { Contacts()  }
            composable(NavRoutes.About.route) { About() }
        }
    }
}
@Composable
fun NavBar(navController: NavController){
    Row(
        Modifier.fillMaxWidth().padding(bottom = 8.dp)){
        Text("Home",
            Modifier
                .weight(0.33f)
                .clickable { navController.navigate(NavRoutes.Home.route) },
fontSize = 22.sp, color= Color(0xFF6650a4))
        Text("Contacts",
            Modifier
                .weight(0.33f)
                .clickable { navController.navigate(NavRoutes.Contacts.route) },
fontSize = 22.sp, color= Color(0xFF6650a4))
        Text("About",
            Modifier
                .weight(0.33f)
                .clickable { navController.navigate(NavRoutes.About.route) },
fontSize = 22.sp, color= Color(0xFF6650a4))
    }
}

@Composable
fun Home(){
    Text("Home Page", fontSize = 30.sp)
}
@Composable
fun Contacts(){
    Text("Contact Page", fontSize = 30.sp)
}
@Composable
fun About(){
    Text("About Page", fontSize = 30.sp)
}
```

```
sealed class NavRoutes(val route: String) {
    object Home : NavRoutes("home")
    object Contacts : NavRoutes("contact")
    object About : NavRoutes("about")
}
```

Итак, здесь для описания маршрутов применяется класс NavRoutes, который через параметр получает маршрут. В этом классе определены три возможных маршрута - Home, Contacts и About.

Для каждого из этих маршуртов определены соответствующие одноименные компоненты:

```
@Composable
fun Home(){
    Text("Home Page", fontSize = 30.sp)
}
@Composable
fun Contacts(){
    Text("Contact Page", fontSize = 30.sp)
}
@Composable
fun About(){
    Text("About Page", fontSize = 30.sp)
}
```

Для простоты каждый компонент просто выводит некоторый текст.

ЧТобы в приложении можно было программно переходить по компонентам определена своего рода навигационная панель - компонент NavBar с тремя условными кнопками:

```
@Composable
fun NavBar(navController: NavController){
    Row(
        Modifier.fillMaxWidth().padding(bottom = 8.dp)){
        Text("Home",
            Modifier
                .weight(0.33f)
                .clickable { navController.navigate(NavRoutes.Home.route) },
fontSize = 22.sp, color= Color(0xFF6650a4))
        Text("Contacts",
            Modifier
                .weight(0.33f)
                .clickable { navController.navigate(NavRoutes.Contacts.route) },
fontSize = 22.sp, color= Color(0xFF6650a4))
        Text("About",
            Modifier
                .weight(0.33f)
                .clickable { navController.navigate(NavRoutes.About.route) },
fontSize = 22.sp, color= Color(0xFF6650a4))
```

```
    }
}
```

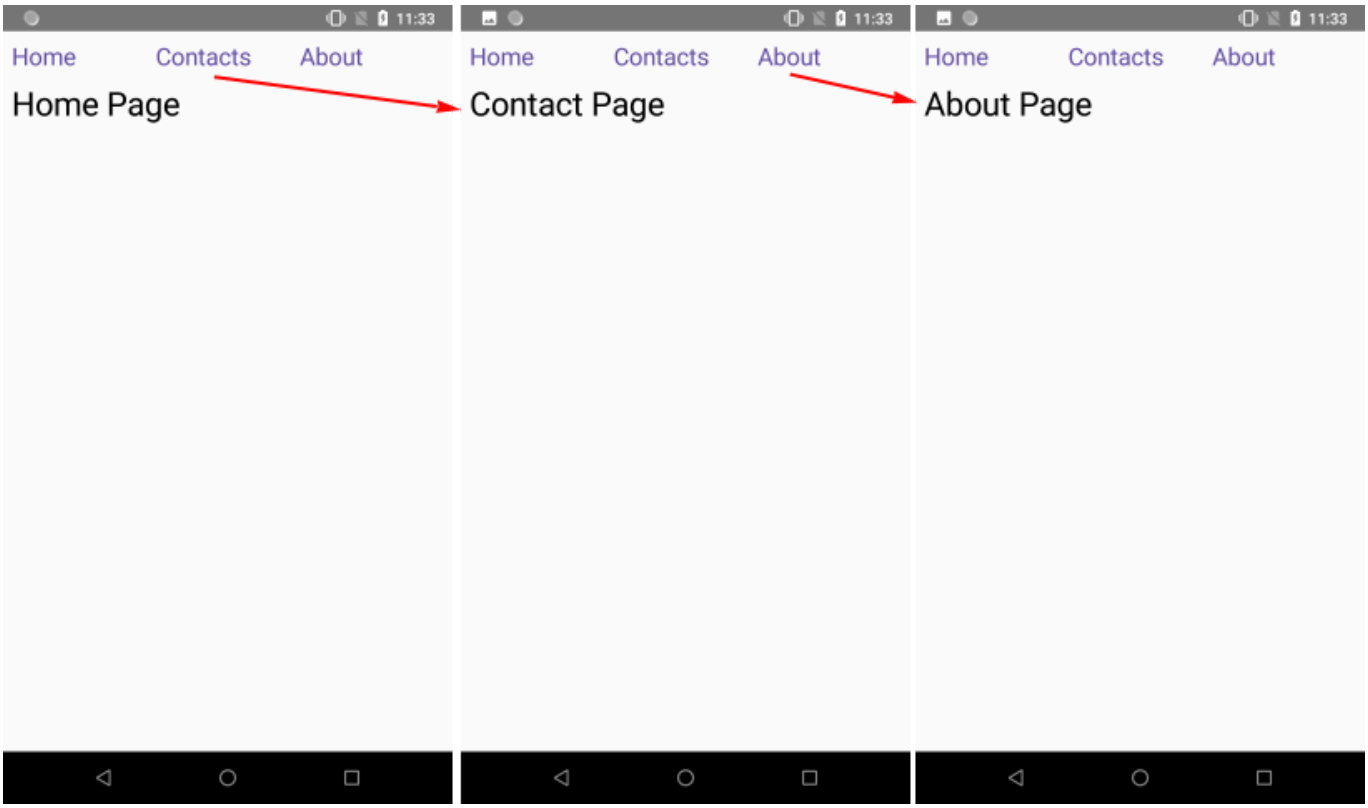
Этот компонент извне принимает объект NavController, с помощью метода navigate которого можно перейти по определенному маршруту.

И чтобы все это объединить определен компонент Main:

```
@Composable
fun Main() {
    val navController = rememberNavController()
    Column(Modifier.padding(8.dp)) {
        NavBar(navController = navController)
        NavHost(navController, startDestination = NavRoutes.Home.route) {
            composable(NavRoutes.Home.route) { Home() }
            composable(NavRoutes.Contacts.route) { Contacts()  }
            composable(NavRoutes.About.route) { About() }
        }
    }
}
```

Вначале в нем создаем с помощью функции rememberNavController() объект NavController. Затем создаем вертикальную компоновку, где в верхней части располагается навигационная панель, а за ней идет компонент NavHost, внутри которого и разворачиваются навигационные компоненты. По умолчанию NavHost отображает компонент Home.

Запустим приложение и мы сможем переходить с помощью ссылок по различным компонентам в приложении:



Параметры навигации

Иногда бывает необходимо при переходе от одного экрана к другому требуется передать аргумент в пункт назначения. Compose поддерживает передачу аргументов самых различных типов. Для этого сначала надо добавить именя аргумента к маршруту назначения:

```
NavHost(navController = navController, startDestination = Routes.Home.route) {

    composable(Routes.User.route + "/{userId}") { stackEntry ->

        val userId = stackEntry.arguments?.getString("userId")

        User(userId)
    }
}
```

В данном случае в маршрут User передается аргумент "userId". Когда приложение запускает переход к месту назначения, значение аргумента сохраняется в соответствующей записи стека навигации. Запись стека передается в качестве параметра в концевую лямбду функции composable(), откуда ее можно извлечь и передать компоненту User.

По умолчанию предполагается, что аргумент навигации имеет тип String, соответственно для его извлечения из записи стека применяется метод getString(). Чтобы передать аргументы разных типов, тип необходимо указать с помощью перечисления NavType через параметр arguments функции composable(). В следующем примере тип параметра объявлен как тип Int, соответственно для его извлечения используется метод getInt():

```
composable(
    Routes.User.route + "/{userId}",
    arguments = listOf(navArgument("userId") { type = NavType.IntType })
){

        navBackStack -> User(navBackStack.arguments?.getInt("userId"))
}
```

На своей стороне компонент User должен ожидать получения аргумент userId через одноименный параметр:

```
@Composable
fun User(userId: String?) {

.......................
}
```

И на последнем шаге надо передать значение аргумента при вызове метода navigation(), например, добавив значение аргумента в конец маршрута назначения:

```
val someId = 22   // некоторое значение, которое передается аргументу навигации

Button(onClick = {

    navController.navigate(Routes.User.route + "/$someId")
}) {

    Text(text = "Show User")
}
```

В итоге при нажатии на кнопку произойдет следующая последовательность событий:

1. Для текущего пункта назначения создается запись для стека
2. Текущее значение someId сохраняется в эту запись
3. Запись помещается в стек навигации
4. Вызывается функция composable() в объявлении NavHost
5. Концевая лямбда функции composable() извлекает значение аргумента из записи стека и передает его в компонент User

Пример использования параметров

Рассмотрим простейший пример применения параметров при навигации:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.clickable
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.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
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.navigation.NavController
import androidx.navigation.NavType
import androidx.navigation.compose.NavHost
```

```
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import androidx.navigation.navArgument


class MainActivity : ComponentActivity() {

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

sealed class UserRoutes(val route: String) {
    object Users : UserRoutes("users")
    object User : UserRoutes("user")
}
@Composable
fun Main() {
    val navController = rememberNavController()
    val employees = listOf(
        Employee(1, "Tom", 39),
        Employee(2, "Bob", 43),
        Employee(3, "Sam", 28)
    )
    NavHost(navController, startDestination = UserRoutes.Users.route) {
        composable(UserRoutes.Users.route) { Users(employees, navController) }
        composable(UserRoutes.User.route + "/{userId}",
            arguments = listOf(navArgument("userId") { type = NavType.IntType }))
{
            stackEntry ->
                val userId = stackEntry.arguments?.getInt("userId")
                User(userId, employees)
        }
    }
}
@Composable
fun Users(data: List<Employee>, navController: NavController){
    LazyColumn {
        items(data){
            u->
                Row(Modifier.fillMaxWidth()){
                    Text(u.name,
                        Modifier.padding(8.dp).clickable {
navController.navigate("user/${u.id}") },
                        fontSize = 28.sp)
                }
        }
    }
}
@Composable
fun User(userId:Int?, data: List<Employee>){
```

```
    val user = data.find { it.id==userId }
    if(user!=null) {
        Column {
            Text("Id: ${user.id}", Modifier.padding(8.dp), fontSize = 22.sp)
            Text("Name: ${user.name}", Modifier.padding(8.dp), fontSize = 22.sp)
            Text("Age: ${user.age}", Modifier.padding(8.dp), fontSize = 22.sp)
        }
    }
    else{
        Text("User Not Found")
    }
}

data class Employee(val id:Int, val name:String, val age:Int)
```

Здесь для описания данных используется класс Employee с тремя свойствами, из которых свойство id - идентификатор, уникально идентифицирует пользователя.

Компонент User применяется для отображения одного объекта Employee. Через параметры он получает идентификатор пользователя и список пользователей:

```
@Composable
fun User(userId:Int?, data: List<Employee>){
    val user = data.find { it.id==userId }
.........................................
```

Получив идентификатор, компонент извлекает из переданного списка нужного пользователя и выводит его данные на экран.

Компонент Users отображает список пользователей

```
@Composable
fun Users(data: List<Employee>, navController: NavController){
    LazyColumn {
        items(data){
            u->
                Text(
                    u.name,
                    Modifier.padding(8.dp).clickable {
navController.navigate("user/${u.id}") },
                    fontSize = 28.sp
                )
        }
    }
}
```

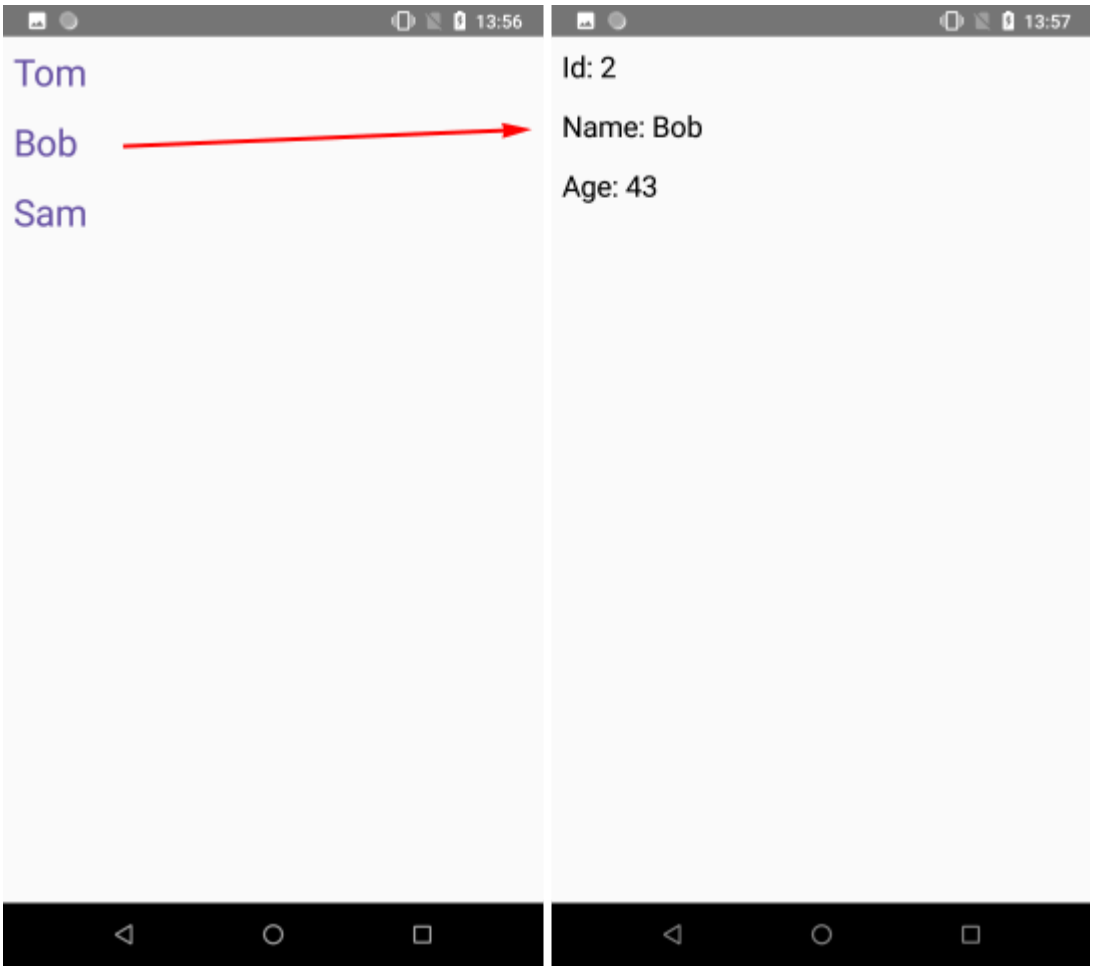
По нажатию на каждый элемент списка будет происходить переход к отображению соответствующего пользователя. И при переходе передается в качестве аргумента навигации идентификатор пользователя

В компоненте Main создаем список пользователей для отображения и разворачиваем NavHost для его отображения:

```
fun Main() {
    val navController = rememberNavController()
    val employees = listOf(
        Employee(1, "Tom", 39),
        Employee(2, "Bob", 43),
        Employee(3, "Sam", 28)
    )
    NavHost(navController, startDestination = UserRoutes.Users.route) {
        composable(UserRoutes.Users.route) { Users(employees, navController) }
        composable(UserRoutes.User.route + "/{userId}",
            arguments = listOf(navArgument("userId") { type = NavType.IntType }))
{
            stackEntry ->
                val userId = stackEntry.arguments?.getInt("userId")
                User(userId, employees)
        }
    }
}
```

В данном случае параметр userId определяется как значение типа Int, и для его получения применяется метод getInt()

Таким образом, если мы запустим приложение, мы увидим список пользователей. При нажатии на одного из пользователей в списке произойдет переход к компоненту User, который отобразит информацию по данному пользователю:



Панель навигации

Для управления навигацией мы можем использовать стандартные встроенные компоненты типа кнопок, определять свои компоненты, однако специально для целей навигации Compose также предоставляет специальный компонент - NavigationBar, который представляет навигационную панель. Каждый отдельный элемент этой панели представляет компонент NavigationBarItem.

Реализация NavigationBar обычно включает содержит цикл forEach, который проходит по списку и для каждого его элемента создает отдельный компонент BNavigationBarItem. Для NavigationBarItem настраивается иконку и текст, а также обработчик onClick для перехода к соответствующему пункту назначения.

Рассмотрим небольшой примерчик:

package com.example.helloapp

import android.os.Bundle import androidx.activity.ComponentActivity import androidx.activity.compose.setContent import androidx.compose.foundation.layout.Column import androidx.compose.foundation.layout.padding import androidx.compose.material.icons.Icons import androidx.compose.material.icons.filled.Face import androidx.compose.material.icons.filled.Home import androidx.compose.material.icons.filled.Info import androidx.compose.material3.Icon import androidx.compose.material3.NavigationBar import androidx.compose.material3.NavigationBarItem import androidx.compose.material3.Text

import androidx.compose.runtime.Composable import androidx.compose.runtime.getValue import androidx.compose.ui.Modifier import androidx.compose.ui.graphics.vector.ImageVector import androidx.compose.ui.tooling.preview.Preview import androidx.compose.ui.unit.dp import androidx.compose.ui.unit.sp import androidx.navigation.NavController import androidx.navigation.NavGraph.Companion.findStartDestination import androidx.navigation.compose.NavHost import androidx.navigation.compose.composable import androidx.navigation.compose.currentBackStackEntryAsState import androidx.navigation.compose.rememberNavController

class MainActivity : ComponentActivity() {

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

} @Composable fun Main() { val navController = rememberNavController() Column(Modifier.padding(8.dp)) { NavHost(navController, startDestination = NavRoutes.Home.route, modifier = Modifier.weight(1f)) { composable(NavRoutes.Home.route) { Home() } composable(NavRoutes.Contacts.route) { Contacts() } composable(NavRoutes.About.route) { About() } } BottomNavigationBar(navController = navController) } }

@Composable fun BottomNavigationBar(navController: NavController) { NavigationBar { val backStackEntry by navController.currentBackStackEntryAsState() val currentRoute = backStackEntry?.destination?.route

```
    NavBarItems.BarItems.forEach { navItem ->
        NavigationBarItem(
            selected = currentRoute == navItem.route,
            onClick = {
                navController.navigate(navItem.route) {
                    popUpTo(navController.graph.findStartDestination().id)
{saveState = true}
                    launchSingleTop = true
                    restoreState = true
                }
            },
            icon = {
                Icon(imageVector = navItem.image,
                    contentDescription = navItem.title)
            },
            label = {
                Text(text = navItem.title)
            }
        )
    }
}
```

}

object NavBarItems { val BarItems = listOf(BarItem(title = "Home", image = Icons.Filled.Home, route = "home"), BarItem(title = "Contacts", image = Icons.Filled.Face, route = "contacts"), BarItem(title = "About", image = Icons.Filled.Info, route = "about")) }

data class BarItem(val title: String, val image: ImageVector, val route: String)

@Composable fun Home(){ Text("Home Page", fontSize = 30.sp) } @Composable fun Contacts(){ Text("Contact Page", fontSize = 30.sp) } @Composable fun About(){ Text("About Page", fontSize = 30.sp) }

sealed class NavRoutes(val route: String) { object Home : NavRoutes("home") object Contacts : NavRoutes("contacts") object About : NavRoutes("about") }

Рассмотрим основные моменты. Прежде всего для представления отдельного элемента панели навигации определяем класс BarItem:

```
data class BarItem(
    val title: String,
    val image: ImageVector,
    val route: String
)
```

Данный класс будет хранить текст ссылки навигации, иконку и маршрут для перехода к другому экрану.

Для представления набора ссылок навигации определяем объект NavBarItems, который содержит список из трех навигационных ссылок:

```
object NavBarItems {
    val BarItems = listOf(
        BarItem(
            title = "Home",
            image = Icons.Filled.Home,
            route = "home"
        ),
        BarItem(
            title = "Contacts",
            image = Icons.Filled.Face,
            route = "contacts"
        ),
        BarItem(
            title = "About",
            image = Icons.Filled.Info,
            route = "about"
        )
    )
}
```

Для создания самой навигационной панели определяем компонент BottomNavigationBar:

```
@Composable
fun BottomNavigationBar(navController: NavController) {
    NavigationBar {
        val backStackEntry by navController.currentBackStackEntryAsState()
        val currentRoute = backStackEntry?.destination?.route

        NavBarItems.BarItems.forEach { navItem ->
            NavigationBarItem(
                selected = currentRoute == navItem.route,
                onClick = {
                    navController.navigate(navItem.route) {
                        popUpTo(navController.graph.findStartDestination().id)
{saveState = true}
                        launchSingleTop = true
                        restoreState = true
                    }
                },
                icon = {
                    Icon(imageVector = navItem.image,
                        contentDescription = navItem.title)
                },
                label = {
                    Text(text = navItem.title)
                }
            )
        }
    }
}
```

Фактически этот компонент является оберткой над NavigationBar, в который передает объект NavController для выполнения навигации.

Внутри BottomNavigationBar мы можем идентифицировать текущий маршрут с помощью метода currentBackStackEntryAsState() контроллера навигации. Это позволит провести некоторую стилизацию элементов панели на основе текущего маршрута:

```
val backStackEntry by navController.currentBackStackEntryAsState()
val currentRoute = backStackEntry?.destination?.route
```

Далее проходим по всем элементам в списке и выводим каждый элемент с помощью компонента NavigationBarItem:

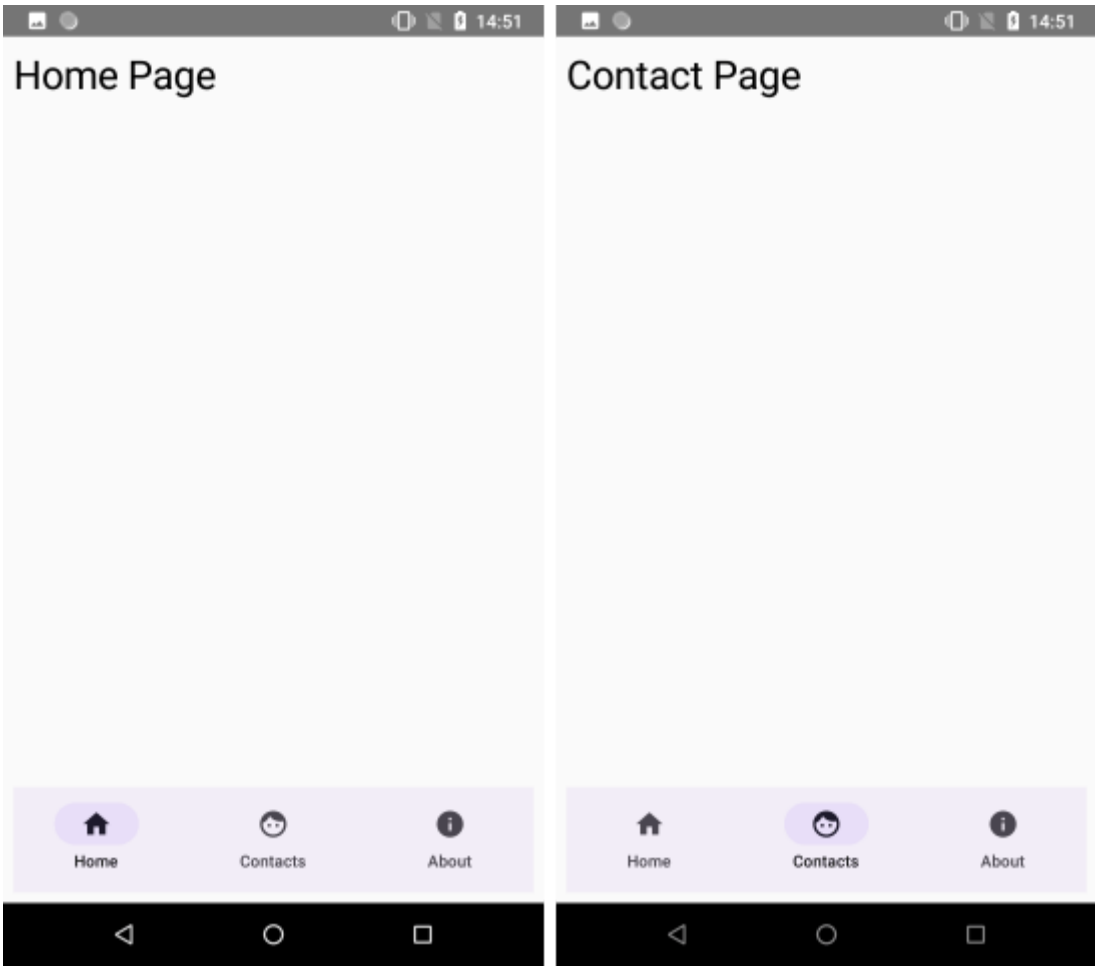
```
NavBarItems.BarItems.forEach { navItem ->
    NavigationBarItem(
        selected = currentRoute == navItem.route,
        onClick = {
```

```
            navController.navigate(navItem.route) {
                popUpTo(navController.graph.findStartDestination().id) {saveState
= true}
                launchSingleTop = true
                restoreState = true
            }
        },
        icon = {
            Icon(imageVector = navItem.image,
            contentDescription = navItem.title)
        },
        label = {
            Text(text = navItem.title)
        }
    )
}
```

Обратите внимание, что при нажатии зедсь вызывается функция popUpTo(), чтобы гарантировать, что если пользователь нажмет кнопку "Назад", навигация вернется к начальному пункту назначения. Мы можем определить начальный пункт назначения, вызвав метод findStartDestination() в графе навигации:

```
onClick = {
    navController.navigate(navItem.route) {
        popUpTo(navController.graph.findStartDestination().id) {saveState = true}
        launchSingleTop = true
        restoreState = true
    }
},
```

Запустим приложение, и нашему взору предстанет очаровательная панель с иконками, нажав на которые мы сможем переходить к различным компонентами:



Обработка жестов

Жесты нажатия

Jetpack Compose предоставляет функционал для работы с распространенными жестами (касание, двойное касание, длительное нажатие и перетаскивание, масштабирование, вращение, смахивание и т.д.). Причем в некоторых случаях Compose предоставляет два способа обнаружения жестов. Один из подходов предполагает использование модификаторов обнаружения жестов. Другой подход - применение функций интерфейса PointerInputScope, которые требуют дополнительного написания кода, но предоставляют более продвинутые возможности работы с жестами.

Первый подход уже был ранее частично рассмотрен. В частности, с помощью модификатора clickable можно обнаружить касание на компоненте. Этот модификатор принимает функцию (лямбду), которая выполняется при касании/нажатии на компоненте, к которому применяется этот модификатор. Например:

```
Text(
    "Click",
    Modifier.clickable {
        // здесь обработка нажатия
    }
)
```

Проблема данного модификатора состоит в том, что он не может различать различные типы жестов - касания, нажатия, длинные нажатия и двойные нажатия. Для этого уровня точности нам нужно использовать функцию detectTapGestures() класса PointerInputScope. Эта функция применяется к компоненту через модификатор pointerInput(), который дает нам доступ к PointerInputScope следующим образом:

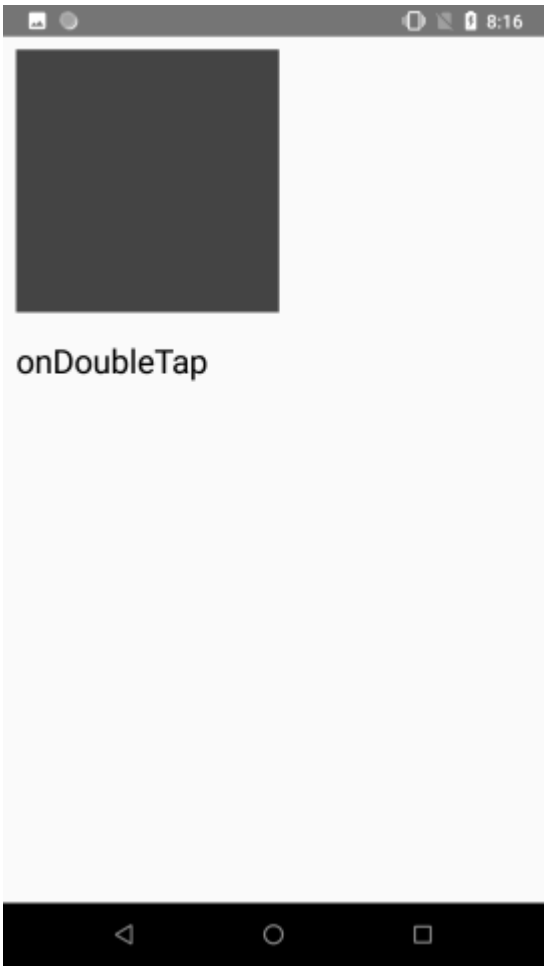
```
Text(
    "Click",
    Modifier
        .pointerInput(Unit) {
            detectTapGestures(
                onPress = { /* обработка нажатия */ },
                onDoubleTap = { /* обработка двойного нажатия */ },
                onLongPress = { /* обработка долгого нажатия */ },
                onTap = { /* обработка простого касания */ }
            )
        }
)
```

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.gestures.detectTapGestures
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.Text

import androidx.compose.runtime.Composable
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.input.pointer.pointerInput
import androidx.compose.ui.tooling.preview.Preview
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 tapType by remember { mutableStateOf("Undefined") }

            Column(Modifier.fillMaxSize()) {
                Box(
                    Modifier
                        .padding(10.dp)
                        .background(Color.DarkGray)
                        .size(200.dp)
                        .pointerInput(Unit) {
                            detectTapGestures(
                                onPress = { tapType = "onPress" },
                                onDoubleTap = { tapType = "onDoubleTap" },
                                onLongPress = { tapType = "onLongPress" },
                                onTap = { tapType = "onTap" }
                            )
                        }
                )
                Text(tapType, Modifier.padding(10.dp), fontSize = 25.sp)
            }
        }
    }
}
```

В данном случае по нажатию на темно-сервый квадрат в модиифкаторе pointerInput перехватываем тип нажатия и выводим его в текстовое поле.



Петаскивание

Модификатор draggable() позволяет определить жесты перетаскивания на компоненте. Этот модификатор сохраняет смещение (или дельту) движения перетаскивания от исходной точки по мере его возникновения и сохраняет его в состоянии, которое создается с помощью функции rememberDraggableState(). Это состояние затем можно использовать, например, для перемещения перетаскиваемого компонента в соответствии с жестом. Определение функции модификатора:

```
Modifier.draggable(
    state: DraggableState,
    orientation: Orientation,
    enabled: Boolean = true,
    interactionSource: MutableInteractionSource? = null,
    startDragImmediately: Boolean = false,
    onDragStarted: suspend CoroutineScope.(startedPosition: Offset) -> Unit = {},
    onDragStopped: suspend CoroutineScope.(velocity: Float) -> Unit = {},
    reverseDirection: Boolean = false
)
```

Параметры модификатора:

- state: состояние типа DraggableState, которое хранит информацию об операции перетаскивания.

- orientation: направоение пертаскивания. Может быть горизонтальным (значение Orientation.Horizontal), либо вертикальным (Orientation.Vertical)
- enabled: доступна ли операция перетаскивания
- interactionSource: объект MutableInteractionSource, который будет использоваться для генерации DragInteraction.Start при начале перетаскивания.
- startDragImmediately: если установлено значение true, то перетаскивание начнется немедленно. Предназначено для того, чтобы конечные пользователи могли "поймать" анимируемый компонент, нажав на него.
- onDragStarted: suspend-функция, которая вызывает при начале перетаскивания
- onDragStopped: suspend-функция, которая вызывает после завершения перетаскивания.
- reverseDirection: изменяет направление прокрутки на обратное (прокрутка сверху вниз будет вести себя как снизу вверх, а слева направо — как справа налево).

При вызове модификатора draggable() необходимо указать как минимум два параиметра - состояние перемещения и его тип - по горизонтали или по вертикали.

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.gestures.Orientation
import androidx.compose.foundation.gestures.draggable
import androidx.compose.foundation.gestures.rememberDraggableState
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.offset
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Text

import androidx.compose.runtime.Composable
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.tooling.preview.Preview
import androidx.compose.ui.unit.IntOffset
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
```

```
import kotlin.math.roundToInt


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var xOffset by remember { mutableStateOf(0f) }
            Column(Modifier.fillMaxSize()) {
                Box(
                    Modifier
                        .offset { IntOffset(xOffset.roundToInt(), 20) }
                        .background(Color.DarkGray)
                        .size(150.dp)
                        .draggable(
                            orientation = Orientation.Horizontal,
                            state = rememberDraggableState { distance ->
                                xOffset += distance
                            }
                        )
                )
                Text("xOffset: $xOffset", Modifier.padding(10.dp), fontSize=22.sp)
            }
        }
    }
}
```

В данном случае применяется перетаскивание по горизонтали. Для его отслеживания определяем сначала состояние xOffset:

```
var xOffset by remember { mutableStateOf(0f) }
```

Для перетаскивания определяем компонент Box, x-координата которого будет привязана к состоянию xOffset:

```
Box(
    Modifier
        .offset { IntOffset(xOffset.roundToInt(), 20) }
```

Далее к этому компоненту Box применяется модификатор draggable, который использует горизонтальную ориентацию. Параметр состояния устанавливается путем вызова функции rememberDraggableState(), в которой концевая лямбда-выражение используется для получения текущего значения дельты перемещения для получения обновленного состояния xOffset. Это, в свою очередь, приводит к перемещению поля в направлении жеста перетаскивания:

```
.draggable(
    orientation = Orientation.Horizontal,
    state = rememberDraggableState { distance ->
        xOffset += distance
    }
)
```

И для большей наглядности определен компонент Text, который выводит значение xOffset:

Перетаскивание с помощью PointerInputScope

В прошлой теме был рассмотрен модификатор draggable(), который позволяет нам реализовать перетаскивание компонента. Однако этот модификатор имеет ограничение: одновременно он может использовать либо перетаскивание по горизонтали, либо по вертикали. Функция detectDragGestures() типа PointerInputScope решает эту проблему и позволяет нам одновременно поддерживать операции горизонтального и вертикального перетаскивания. Эта функция имеет следующие параметры:

```
suspend fun PointerInputScope.detectDragGestures(
    onDragStart: (Offset) -> Unit = { },
    onDragEnd: () -> Unit = { },
    onDragCancel: () -> Unit = { },
```

```
    onDrag: (change: PointerInputChange, dragAmount: Offset) -> Unit
): Unit
```

- onDragStart: функция, которая вызывается при начале перетаскивания
- onDragEnd: функция, которая вызывается после завершения перетаскивания
- onDragCancel: функция, которая вызывается при отмене перетаскивания
- onDrag: функция, которая вызывается при перетаскивания

Нам достаточно задать последний параметр для управления перетаскиваемым компонентом. И обработка перетаскивания в общем случае будет выглядеть примерно следующим образом:

```
Modifier.pointerInput(Unit) {
    detectDragGestures { _, distance ->
        xOffset += distance.x
        yOffset += distance.y
    }
}
```

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.gestures.Orientation
import androidx.compose.foundation.gestures.detectDragGestures
import androidx.compose.foundation.gestures.draggable
import androidx.compose.foundation.gestures.rememberDraggableState
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.offset
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Text

import androidx.compose.runtime.Composable
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.input.pointer.pointerInput
import androidx.compose.ui.unit.IntOffset
```

```
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import kotlin.math.roundToInt


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var xOffset by remember { mutableStateOf(0f) }
            var yOffset by remember { mutableStateOf(0f) }

            Column(Modifier.fillMaxSize()) {
                Text("xOffset: $xOffset ; yOffset: $yOffset",
Modifier.padding(10.dp), fontSize=22.sp)
                Box(
                    Modifier
                        .offset { IntOffset(xOffset.roundToInt(),
yOffset.roundToInt()) }
                        .background(Color.DarkGray)
                        .size(150.dp)
                        .pointerInput(Unit) {
                            detectDragGestures { _, distance ->
                                xOffset += distance.x
                                yOffset += distance.y
                            }
                        }
                )
            }
        }
    }
}
```

Для отслеживания перетаскивания как по горизонтали, так и по вертикали, определяются две переменных состояния для хранения смещений по оси X и Y:

```
var xOffset by remember { mutableStateOf(0f) }
var yOffset by remember { mutableStateOf(0f) }
```

К их значениям привязаны координаты компонента Box:

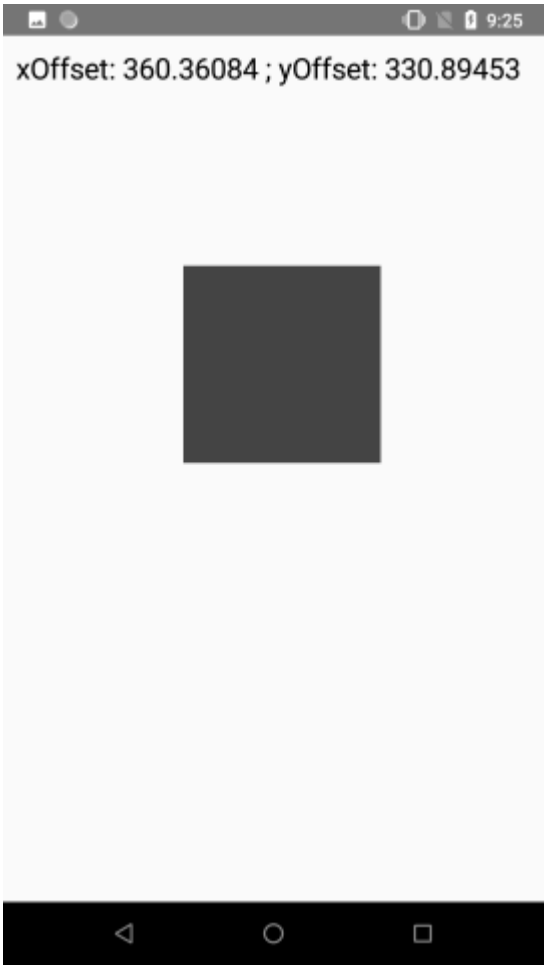
```
Box(
    Modifier.offset { IntOffset(xOffset.roundToInt(), yOffset.roundToInt()) }
```

В лямбда-выражение в функции detectDragGestures передается параметр distance, который представляет объект Offset и из которого мы можем получить последние значения смещения

перетаскивания по осям x и y. Они добавляются к состояниям xOffset и yOffset соответственно, в результате чего компонент Box следует за движением перетаскивания по экрану:

```
.pointerInput(Unit) {
    detectDragGestures { _, distance ->
        xOffset += distance.x
        yOffset += distance.y
    }
}
```

И для наглядности координаты перемещения выводятся на экран ыв компоненте Text:



Перетаскивание по опорным точкам

Кроме обычного возможности обычного перетаскивания Jetpack Compose также позволяет перетаскивать компоненты вдоль некоторой фиксированной линии, которая основана на двух или более опорных точках (anchor points). Каждая опорная точка имеет фиксированные положения на экране, соответственно вся ось перетаскивания также фиксирована.

Точка между двумя опорными точками называется порогом (threshold). Перетаскиваемый компонент вернется к начальной опорной точке, если перетаскивание закончится до достижения порогового значения. Если, с другой стороны, перетаскивание заканчивается после прохождения точки перехода, компонент продолжит движение, пока не достигнет следующей опорной точки.

Для использования перетаскивания по опорным точкам нужно добавить в проект библиотеку foundation. Для этого откроем файл libs.versions.toml и внесем в нем изменения в две секции:

```
versions]
foundation = "1.6.4"

.......................

[libraries]
androidx-foundation = { module = "androidx.compose.foundation:foundation",
version.ref = "foundation" }
```

Затем в файл build.gradle.kts(Module: app) добавим зависимость библиотеки foundation:

```
.............
dependencies {
    implementation(libs.androidx.foundation)
.......................
```

И для применения изменений нажмем на кнопку Sync Now

Основные элементы перетаскивания вдоль фиксированной оси

Для обработчки перетаскивания вдоль фиксированной оси к компоненту, для которого надо обработать перетаскивание, применяется модификатор anchoredDraggable():

```
@ExperimentalFoundationApi
fun <T : Any?> Modifier.anchoredDraggable(
    state: AnchoredDraggableState<T>,
    orientation: Orientation,
    enabled: Boolean = true,
    reverseDirection: Boolean = false,
    interactionSource: MutableInteractionSource? = null,
    startDragImmediately: Boolean = state.isAnimationRunning
): Modifier
```

Его параметры:

- state: состояние типа AnchoredDraggableState, которое хранит информацию о перемещении
- orientation: направление перетаскивания
- enabled: доступно ли перетаскивание
- reverseDirection: в каком направлении идет перетаскивание

- interactionSource: объект MutableInteractionSource, который передается во внутренний модификатор Modifier.draggable
- startDragImmediately: если установлено значение true, то перетаскивание начнется немедленно. Предназначено для того, чтобы конечные пользователи могли "поймать" анимируемый компонент, нажав на него.

Перетаскиваемые опорные точки объявляются с помощью фабрики DraggableAnchors. Координаты этих точек определяются в пикселях в виде значений Float. Например, следующий код создает объект DraggableAnchors, который состоит из трех опорных точек, расположенных в начале, центре и конце пути перетаскивания:

```
enum class Anchors {
    Left,
    Center,
    Right
}

val anchors = DraggableAnchors {
    Anchors.Left at 0f
    Anchors.Center at widthPx / 2
    Anchors.Right at widthPx
}
```

Пороги объявляются как лямбда-выражения, которые возвращают пороговую позицию. При вызове лямбда-выражения в него передается значение, которое представляет расстояние между исходной и конечной опорными точками. Это расстояние можно использовать для расчета пороговой точки на пути перетаскивания. Например, следующий код объявляет порог в точке, составляющей 70% расстояния между двумя опорными точками:

```
{ distance: Float -> distance * 0.7f }
```

После объявления опорных точек и порога они используются для создания объекта AnchoredDraggableState, синтаксис которого следующий:

```
val state = remember {
    AnchoredDraggableState(
        initialValue = [позиция начальной опорной точки],
        anchors = DraggableAnchors {
            [определение всех опорных точек]
        },
        positionalThreshold = [вычисление порога],
        velocityThreshold = [пороговая скорость],
        animationSpec = [анимация]
    )
}
```

Конструктор AnchoredDraggableState принимает ряд параметров:

- intialValue: начальная опорная точка перетаскиваемого элемента, в которой перетаскиваемый элемент появится при первом отображении
- anchors: объект DraggableAnchors, который содержит опорные точки
- positionalThreshold: лямбда вычисления порога
- velocityThreshold: дополнительная настройка, определяющая скорость в dp в секунду, которую должна превысить скорость перетаскивания, чтобы перейти в следующее состояние
- animationSpec: применяет эффекты анимации к операции перетаскивания

При перетаскивании позиция перемещаемого компонента автоматически не обновляется. Нам ее надо обновлять вручную. Для этого мы можем использовать модификатор offset(), который устанавливает координаты компонента. Для установки позиции в этот модификатор можно передать текущее смещения, которое можно получить из состояния AnchoredDraggableState с помощью вызова метода requireOffset(). Результатом будет текущая позиция по оси X или Y, в зависимости от того, является ли направоление перетаскивания горизонтальным или вертикальным. Например, при горизонтальном перетаскивании установка позиции могла бы выглядеть так:

```
Box(Modifier.offset { IntOffset( x = state .requireOffset().roundToInt(), y = 0)
})
```

Пример перетаскивания

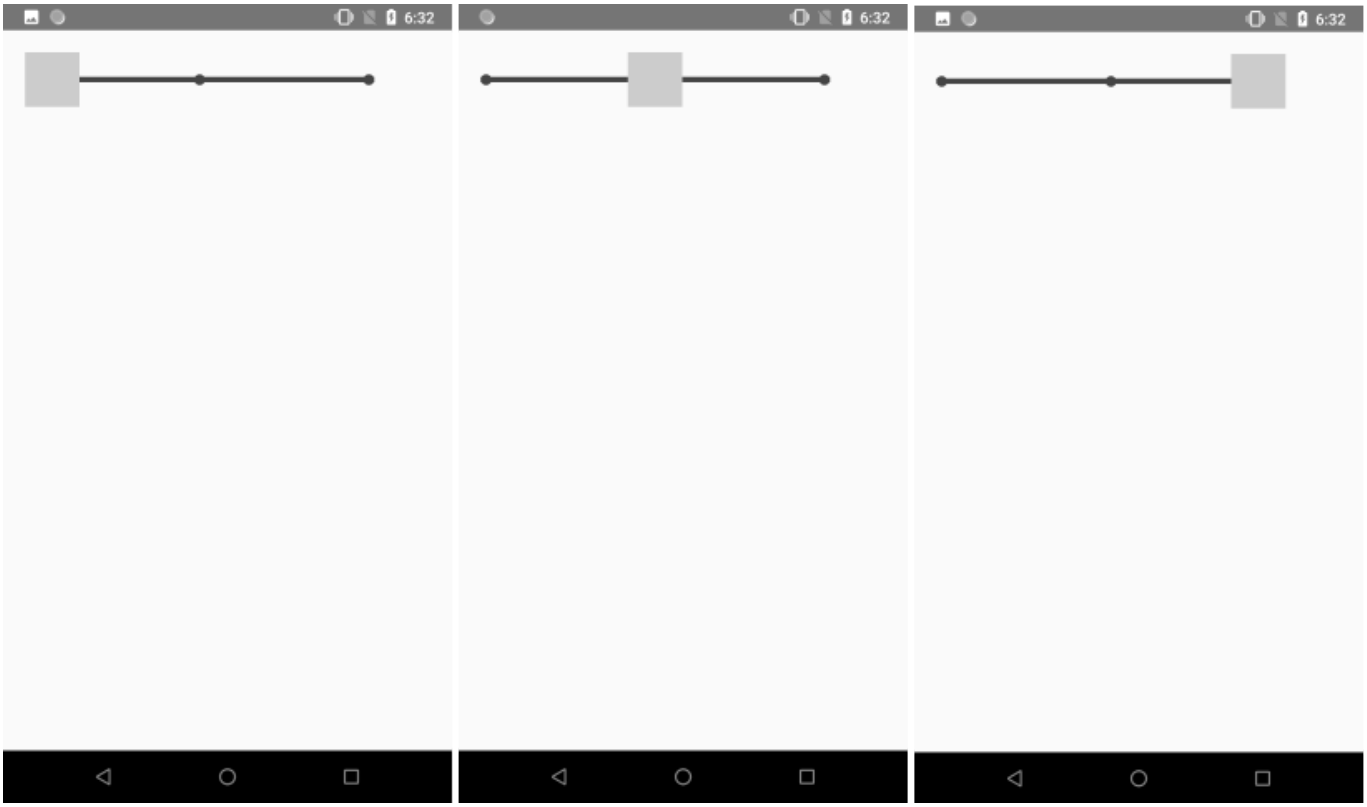
В качестве простейшего примера перетаскивания по опорным точкам рассмотрим следующее приложение:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.tween
import androidx.compose.foundation.ExperimentalFoundationApi
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.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalDensity
import androidx.compose.ui.tooling.preview.Preview
```

```
import androidx.compose.ui.unit.dp
import androidx.compose.foundation.gestures.AnchoredDraggableState
import androidx.compose.foundation.gestures.DraggableAnchors
import androidx.compose.foundation.gestures.Orientation
import androidx.compose.foundation.gestures.anchoredDraggable
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.offset
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.ui.unit.IntOffset
import kotlin.math.roundToInt

enum class Anchors {
    Start,
    Center,
    End
}

class MainActivity : ComponentActivity() {

    @OptIn(ExperimentalFoundationApi::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val density = LocalDensity.current
            val parentBoxWidth = 320.dp
            val boxSize = 50.dp
            val widthPx = with(density) {(parentBoxWidth - boxSize).toPx() }
            val state = remember {
                AnchoredDraggableState(
                    initialValue = Anchors.Start,
                    anchors = DraggableAnchors {
                        Anchors.Start at 0f
                        Anchors.Center at widthPx / 2
                        Anchors.End at widthPx
                    },

                    positionalThreshold = { distance: Float -> distance * 0.5f },
                    velocityThreshold = { with(density) { 100.dp.toPx() } },
                    animationSpec = tween()
                )
            }
            Box(Modifier.padding(20.dp).width(parentBoxWidth)){

Box(Modifier.width(parentBoxWidth).height(5.dp).background(Color.DarkGray).align(A
lignment.CenterStart))
                Box(Modifier.size(10.dp).background(Color.DarkGray,
CircleShape).align(Alignment.CenterStart))
                Box(Modifier.size(10.dp).background(Color.DarkGray,
CircleShape).align(Alignment.Center))
                Box(Modifier.size(10.dp).background(Color.DarkGray,
CircleShape).align(Alignment.CenterEnd))
                Box(
```

```
                    Modifier
                        .offset {
                            IntOffset(
                                x = state
                                    .requireOffset()
                                    .roundToInt(),
                                y = 0,
                            )
                        }
                        .anchoredDraggable(
                            state,
                            Orientation.Horizontal
                        )
                        .size(boxSize)
                        .background(Color.LightGray)
                )
            }
        }
    }
}
```

В итоге у нас получится ось с тремя точками, по которым мы сможем перемещать светло-серый компонент Box:



Итак, здесь мы определяем три опорных точки:

```
enum class Anchors {
    Start,
    Center,
```

```
    End
}
```

Сначала определяем несколько базовых переменных, которые будут применяться при рассчетах:

```
val density = LocalDensity.current
val parentBoxWidth = 320.dp
val boxSize = 50.dp
val widthPx = with(density) {(parentBoxWidth - boxSize).toPx() }
```

Переменная density представляет плотность текущего экрана и необходима для перевода из единиц dp в стандартные пиксели. Переменная parentBoxWidth хранит ширину контейнера, а также длину оси, по которой будет перемещаться компонент. Переменная boxSize хранит ширину и высоту компонента (ширина равна высоте). Наконец, ширина перетаскиваемой области в пикселях - переменная widthPx рассчитывается путем вычитания ширины перетаскиваемого компонента из ширины родительского контейнера.

Ширина перетаскиваемого компонента вычитается, чтобы учесть тот факт, что перетаскиваемый компонент будет центрирован по опорным точкам, оставляя отступ в половину ширины перетаскиваемого компонента на первой и последней опорных точках (эти две половины объединяются, и получается полная ширина перетаскиваемого компонента).

Далее с помошью вызова AnchoredDraggableState определяется состояния перетасиквания:

```
val state = remember {
    AnchoredDraggableState(
        initialValue = Anchors.Start,
        anchors = DraggableAnchors {
            Anchors.Start at 0f
            Anchors.Center at widthPx / 2
            Anchors.End at widthPx
        },
        positionalThreshold = { distance: Float -> distance * 0.5f },
        velocityThreshold = { with(density) { 100.dp.toPx() } },
        animationSpec = tween()
    )
}
```

В конструктор класса AnchoredDraggableState передается позиция начальной опорной точки, а также объект DraggableAnchors с тремя опорными точками. Через параметр positionalThreshold задаем порог, который расположен по середине между соседними опорными точками.

Стоит отметить, что на момент написания статьи класс AnchoredDraggableState представляет экспериментальную функциональность, поэтому к методу onCreate() в MainActivity применяется аннотация @ExperimentalFoundationApi:

```
class MainActivity : ComponentActivity() {

    @OptIn(ExperimentalFoundationApi::class)
    override fun onCreate(savedInstanceState: Bundle?) {
```

Компонент верхнего уровня, который содержит интерфейс, представляет компонент Box:

```
Box(Modifier.padding(20.dp).width(parentBoxWidth)){

Box(Modifier.width(parentBoxWidth).height(5.dp).background(Color.DarkGray).align(A
lignment.CenterStart))
    Box(Modifier.size(10.dp).background(Color.DarkGray,
CircleShape).align(Alignment.CenterStart))
    Box(Modifier.size(10.dp).background(Color.DarkGray,
CircleShape).align(Alignment.Center))
    Box(Modifier.size(10.dp).background(Color.DarkGray,
CircleShape).align(Alignment.CenterEnd))
```

Этот Box имеет ширину, равную parentBoxWidth. Внутри этого компонента с помощью дополнительных компонентов Box отрисована ось перемещения с тремя точками.

Затем определяется компонент Box, который собственно будет перемещаться вдоль оси:

```
Box(
    Modifier
        .offset {
            IntOffset(
                x = state
                    .requireOffset()
                    .roundToInt(),
                y = 0,
            )
        }
        .anchoredDraggable(
            state,
            Orientation.Horizontal,
        )
        .size(boxSize)
        .background(Color.LightGray)
)
```

Сначала к этому компоненту применяется модификатор offset() для управления положением. Для получения координаты x применяется метод requireOffset() состояния. Затем смещение используется для позиционирования Box вдоль оси X.

Затем Box становится перетаскиваемым путем применения модификатора anchoredDraggable(), в который передается состояние перетаскивания и направление (по горизонтали).

Прокрутка

Прокрутка или скроллинг представляет движение пальцем по вертикали или по горизонтали. Ранее мы уже рассматривали прокрутку в статье Создание прокрутки, в частности, мы рассматривали, как с помощью модификаторов horizontalScroll() и verticalScroll() можно определить прокрутку соответственно по горизонтали и вертикали. Благодаря этим модификаторам нам не надо думать о реализации логики прокрутки, все делеается автоматически. Тем не менее мы можем задать и свою собственную логику прокрутки, применяя модификатор scrollable().

Для управления прокруткой применяется состояние, которое создается с помощью функции rememberScrollableState(). В нее передается функция, которая дает нам доступ к расстоянию, пройденному жестом прокрутки. Данное расстояние можно использовать для регулировки смещения одного или нескольких компонентов на экране. 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.gestures.Orientation
import androidx.compose.foundation.gestures.rememberScrollableState
import androidx.compose.foundation.gestures.scrollable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.offset
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size

import androidx.compose.runtime.Composable
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.tooling.preview.Preview
import androidx.compose.ui.unit.IntOffset
import androidx.compose.ui.unit.dp
import kotlin.math.roundToInt


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var offset by remember { mutableStateOf(0f) }
            Box(Modifier
                .fillMaxSize()
```

```
                .scrollable(
                    orientation = Orientation.Vertical,
                    state = rememberScrollableState { distance ->
                        offset += distance
                        distance
                    }
                )
            ) {
                Box(Modifier
                    .size(150.dp)
                    .padding(10.dp)
                    .offset { IntOffset(0, offset.roundToInt()) }
                    .background(Color.DarkGray))
            }
        }
    }
}
```

В данном случае мы управляем прокруткой внутри элемента Box, который занимает всю поверхность экрана. Для отслеживания позиции прокрутки устанавливается переменная offset.

```
var offset by remember { mutableStateOf(0f) }
```

В компоненте Box применяем модификатор scrollable():

```
.scrollable(
    orientation = Orientation.Vertical,
    state = rememberScrollableState { distance ->
        offset += distance
        distance
    }
)
```

В rememberScrollableState передается функция, которая определяет параметр distanse. Этот параметр представляет объект Offset, из которого мы можем получить последние значения смещения при прокрутке. Затем это смещение добавляется к состоянияю offset.

Вложенный компонент Box, который представляет темно-серый квадрат размером 150x150, использует значение offset для установки положения по оси y

```
Box(Modifier
    .size(150.dp)
    .padding(10.dp)
    .offset { IntOffset(0, offset.roundToInt()) }
    .background(Color.DarkGray))
```

Таким образом, при прокрутке будет изменяться позиция вложенного компонента Box:

Масштабирование, вращение и перемещение

Мастабирование

С помощью двух пальцев можно выполнить масштабирование контента и создать эффекта увеличения (при разведении пальцев) или уменьшения масштаба (при сведении пальцев). Этот тип жестов обрабатывается с помощью модификатора transformable(), который принимает в качестве параметра состояние типа TransformableState:

```
fun Modifier.transformable(
    state: TransformableState,
    lockRotationOnZoomPan: Boolean = false,
    enabled: Boolean = true
): Modifier
```

Для создания объекта TransformableState применяется функция rememberTransformableState(), которая принимает функцию с тремя параметрами:

```
(zoomChange: Float, panChange: Offset, rotationChange: Float) -> Unit
```

- zoomChange: значение Float, которое обновляется при выполнении жестов масштабирования.
- panChange: значение Offset, которое содержит текущие значения смещения x и y. Это значение обновляется, когда с помощью жестов производится перемещение целевого компонента.
- rotationChange: значение Float, которое представляет текущее изменение угла с помощью жестов вращения.

При вызове функции rememberTransformableState() необходимо объявить все три параметра, даже если их не предполагается использовать. Типичное объявление TransformableState, которое отслеживает изменения масштаба, может выглядеть следующим образом:

```
var scale by remember { mutableStateOf(1f) }

val state = rememberTransformableState {
    scaleChange, offsetChange, rotationChange -> scale *= scaleChange
}
```

Затем созданное состояние можно передать в вызов модификатора transformable():

```
Composable(modifier = Modifier.transformable(state = state) {

    }
)
```

Например, рассмотрим следующее приложение:

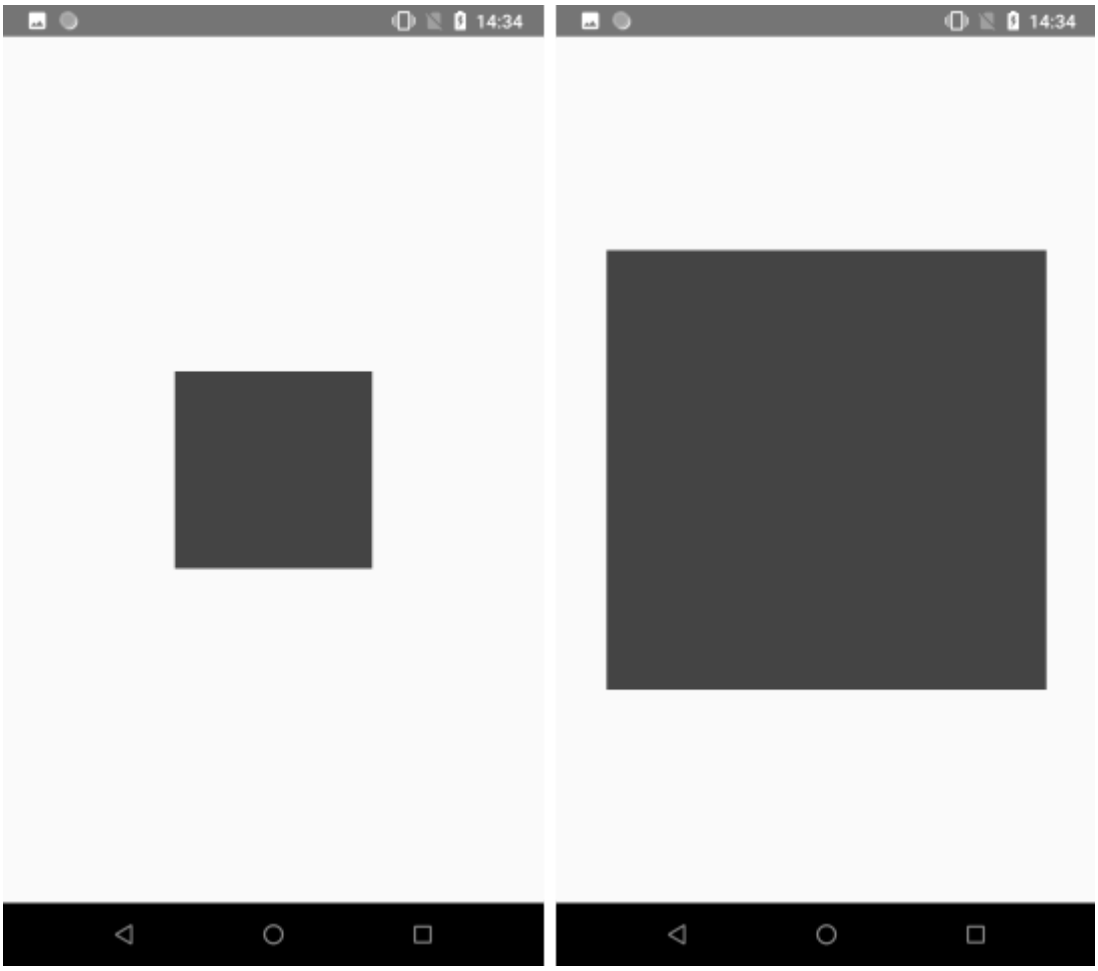
```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.gestures.rememberTransformableState
import androidx.compose.foundation.gestures.transformable
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.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.graphics.graphicsLayer
```

```
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var scale by remember { mutableStateOf(1f) }

            val state = rememberTransformableState {
                scaleChange, offsetChange, rotationChange -> scale *= scaleChange
            }

            Box(contentAlignment = Alignment.Center, modifier =
Modifier.fillMaxSize()) {
                Box(Modifier
                    .graphicsLayer(scaleX = scale, scaleY = scale)
                    .transformable(state = state)
                    .background(Color.DarkGray)
                    .size(150.dp)
                )
            }
        }
    }
}
```

Здесь внутренний компонент Box использует модификатор transformable() для отслеживания масштабирования. По мере выполнения жеста масштабирования состояние масштаба - переменная scale будет обновляться. Чтобы отразить эти изменения, мы обращаемся к графическому слою компонента, установив параметры scaleX и scaleY в текущее состояние масштаба:

```
Box(Modifier
    .graphicsLayer(scaleX = scale, scaleY = scale)
```

Таким образом, с помощью жеста масштабирования мы сможем изменять размер вложенного компонента Box:



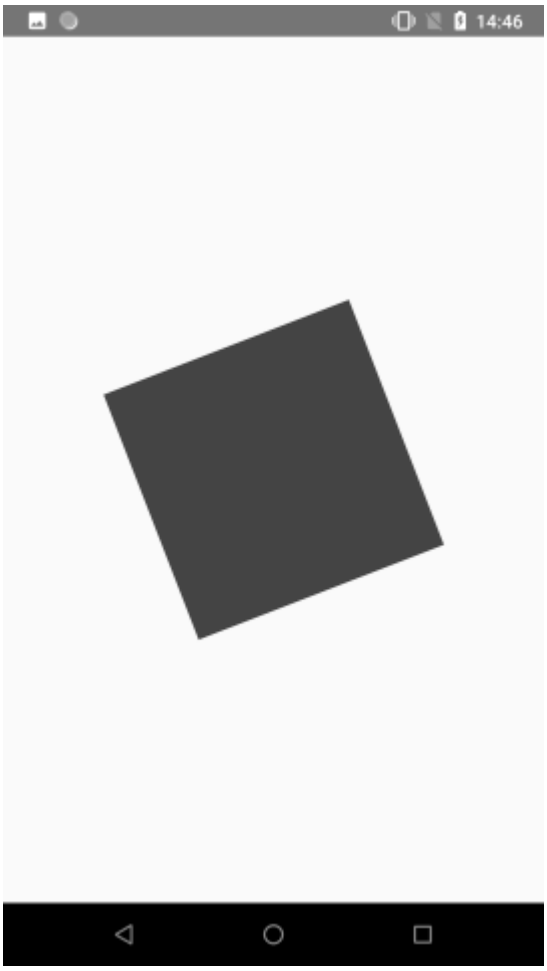
Вращение

Аналогично с помощью того же модификатора transformable() можно обрабатывать жесты вращения:

```
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.gestures.rememberTransformableState
import androidx.compose.foundation.gestures.transformable
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.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.graphics.graphicsLayer
import androidx.compose.ui.tooling.preview.Preview
```

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


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var angle by remember { mutableStateOf(0f) }
            val state = rememberTransformableState {
                scaleChange, offsetChange, rotationChange -> angle +=
rotationChange
            }

            Box(contentAlignment = Alignment.Center, modifier =
Modifier.fillMaxSize()) {
                Box(Modifier
                    .graphicsLayer(rotationZ = angle)
                    .transformable(state = state)
                    .background(Color.DarkGray)
                    .size(200.dp)
                )
            }
        }
    }
}
```

В данном случае для отслеживания угла поворота определяем переменную angle и затем изменяем ее, прибавляя к ней значение из rotationChange. А у компонента Box устанавливаем угол поворота с помощью параметра rotationZ модификатора graphicsLayer



Перемещение

И последний тип трансформаций - перемещение компонента жестами делается аналогично:

```
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.gestures.rememberTransformableState
import androidx.compose.foundation.gestures.transformable
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.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.geometry.Offset
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.graphicsLayer
```

```
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var offset by remember { mutableStateOf(Offset.Zero)}
            val state = rememberTransformableState {
                scaleChange, offsetChange, rotationChange -> offset +=
offsetChange
            }

            Box(contentAlignment = Alignment.Center, modifier =
Modifier.fillMaxSize()) {
                Box(Modifier
                    .graphicsLayer(
                        translationX = offset.x,
                        translationY = offset.y
                    )
                    .transformable(state = state)
                    .background(Color.DarkGray)
                    .size(200.dp)
                )
            }
        }
    }
}
```

Здесь для хранения данных о перемещении определяем переменную offset:

```
var offset by remember { mutableStateOf(Offset.Zero)}
```

Причем эта переменная представляет объект Offset и будет содержать смещение сразу по обоим осям - X и Y.

При обработке перемещения к этой переменной прибавляем значения перемещения из параметра offsetChange:

```
val state = rememberTransformableState {
    scaleChange, offsetChange, rotationChange -> offset += offsetChange
}
```

Для установки привязки компонента Box к значениям из переменной offset используем параметры translationX и translationY функции-модификатора graphicsLayer():

```
Box(Modifier
    .graphicsLayer(
        translationX = offset.x,
        translationY = offset.y
)
```

При необходимости можно объединить обработку нескольких типов жестов:

```
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.gestures.rememberTransformableState
import androidx.compose.foundation.gestures.transformable
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.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.geometry.Offset
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.graphicsLayer
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp


class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            var scale by remember { mutableStateOf(1f) }
            var angle by remember { mutableStateOf(0f) }
            var offset by remember { mutableStateOf(Offset.Zero)}
            val state = rememberTransformableState {
                scaleChange, offsetChange, rotationChange ->
                    scale *= scaleChange
                    angle += rotationChange
                    offset += offsetChange
            }

            Box(contentAlignment = Alignment.Center, modifier =
Modifier.fillMaxSize()) {
```

```
                Box(Modifier
                    .graphicsLayer(
                        scaleX = scale,
                        scaleY = scale,
                        rotationZ = angle,
                        translationX = offset.x,
                        translationY = offset.y
                    )
                    .transformable(state = state)
                    .background(Color.DarkGray)
                    .size(200.dp)
                )
            }
        }
    }
}
```