Лабораторная работа № 5 по теме:
«Обработка касаний и мультитач жестов»

Теоретическая часть

В Android существует несколько способов перехвата событий взаимодействия пользователя с приложением. Для обработки событий в пользовательском интерфейсе осуществляется захват событий из определенного объекта View, с которым взаимодействует пользователь.

В разных классах View содержатся различные открытые методы обратного вызова, которые вызываются платформой Android, когда над этими объектами происходят соответствующие действия. Например, при касании View для этого объекта вызывается метод onTouchEvent (). Чтобы перехватить это событие, нужно расширить класс и переопределить метод. Однако расширение каждого объекта View для обработки события нецелесообразно. Поэтому класс View также содержит набор вложенных интерфейсов с обратными вызовами, которые гораздо проще определить. Эти интерфейсы называются слушателями событий.

Слушатель событий — это интерфейс в классе View, который содержит единственный метод обратного вызова. Эти методы будут вызываться платформой Android, когда пользователь будет определённым образом взаимодействовать с элементом View, для которого зарегистрирован слушатель.

В интерфейсы слушателей событий включены методы обратного вызова, представленные в таблице 1.

Таблица 1. Методы в слушателях событий.

Метод	**Слушатель**	**Описание**
onClick ()	View.OnClickListener	Вызывается, когда пользователь либо касается элемента (в сенсорном режиме), либо фокусируется на элементе с помощью клавиш навигации или трекбола и нажимает клавишу «ввода» или нажимает на трекбол.
onLongClick ()	View.OnLongClickListener	Вызывается, когда пользователь либо касается элемента и удерживает его (в

		сенсорном режиме), либо фокусируется на элементе с помощью навигационных клавиш или трекбола, и нажимает и удерживает клавишу «ввода», либо нажимает и удерживает трекбол (на одну секунду).
onFocusChange ()	View.OnFocusChange Listener	Вызывается, когда пользователь переходит на элемент или от него, используя клавиши навигации или трекбол.
onKey ()	View.OnKeyListener	Вызывается, когда пользователь фокусируется на элементе и нажимает или отпускает аппаратную клавишу на устройстве.
onTouch ()	View.OnTouchListener	Вызывается, когда пользователь выполняет действие, квалифицируемое как событие касания, включая нажатие, отпускание или любой жест на экране (в границах элемента).
onCreateContext Menu ()	View.OnCreateContext MenuListener	Вызывается, когда создается контекстное меню (в результате длительного события Long Click).

Чтобы определить один из этих методов и обработать события, в активити реализуется вложенный интерфейс или определяется как анонимный класс (листинг 1). Затем передаётся экземпляр этой реализации в соответствующий метод View.set...Listener (). Например, вызывается setOnClickListener () и ему передаётся реализация OnClickListener. Также можно реализовать OnClickListener как часть активити (листинг 2). Это позволит избежать дополнительной загрузки классов и выделения объектов.

Листинг 1. Регистрация слушателя нажатия на кнопку.

```
private OnClickListener myListener = new OnClickListener() {
    public void onClick(View v) {
        // действия при нажатии на кнопку
    }
};
protected void onCreate(Bundle savedValues) {
...
    // Определение кнопки из компоновки
    Button button = (Button)findViewById(R.id.myButton);
    // Регистрация слушателя onClick
    button.setOnClickListener(myListener);
...
}
```

Листинг 2. Регистрация слушателя нажатия на кнопку.

```
public class MainActivity extends Activity implements OnClickListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        Button button = (Button)findViewById(R.id.myButton);
        button.setOnClickListener(this);
    }
    @Override
    public void onClick(View v) {
        // действия при нажатии на кнопку
    }
    ...
}
```

Обработка касаний экрана

«Сенсорный жест» возникает, когда пользователь касается сенсорного экрана одним или несколькими пальцами, и приложение интерпретирует эти касания как определенный жест. Существуют два этапа распознавания жестов:

- Сбор данных о событиях касания.
- Интерпретация данных для определения, соответствуют ли они критериям какого-либо жеста, поддерживаемого приложением.

Когда пользователь касается экрана одним или несколькими пальцами, это вызывает метод обратного вызова onTouchEvent () в представлении, которое получило события касания. Для каждой последовательности событий касания (с разными параметрами положения, давления, размера, добавления ещё одного пальца и т.д.), которая в итоге определяется как жест, метод onTouchEvent () вызывается несколько раз.

Жест начинается, когда пользователь впервые касается экрана, продолжается, пока система отслеживает положение пальца (пальцев) пользователя, и заканчивается фиксацией последнего события, когда пальцы пользователя покидают экран. Во время этого взаимодействия объект MotionEvent, переданный в onTouchEvent (), предоставляет подробную информацию о каждом взаимодействии. Ваше приложение

может использовать данные, предоставленные MotionEvent, чтобы определить, произошел ли жест, который его интересует.

Чтобы перехватить события касания в активити или представлении, следует переопределить обратный вызов onTouchEvent ().

В листинге 3 приведён код приложения, которое позволяет отобразить координаты одиночного касания: координаты нажатия, перемещения и отпускания.

Листинг 3. Обработка касания окна активити.

```
package com.example.motioneventsapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    TextView downX; TextView downY;
    TextView upX; TextView upY;
    TextView moveX; TextView moveY;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        downX = (TextView)findViewById(R.id.down_x);
        downY = (TextView)findViewById(R.id.down_y);
        upX = (TextView)findViewById(R.id.up_x);
        upY = (TextView)findViewById(R.id.up_y);
        moveX = (TextView)findViewById(R.id.move_x);
        moveY = (TextView)findViewById(R.id.move_y);
    }
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        String x; String y;
        int action = event.getAction();
        x =  Integer.toString((int)event.getX());
        y =  Integer.toString((int)event.getY());
        switch (action) {
            case (MotionEvent.ACTION_DOWN):
                downX.setText(x);
                downY.setText(y);
                return true;
            case (MotionEvent.ACTION_MOVE):
                moveX.setText(x);
                moveY.setText(y);
                upX.setText("");
                upY.setText("");
                return true;
            case (MotionEvent.ACTION_UP):
                upX.setText(x);
```

```
                upY.setText(y);
                moveX.setText("");
                moveY.setText("");
                return true;
            default:
                return super.onTouchEvent(event);
        }
    }
}
```

На рисунке 1 представлена работа данного активити.

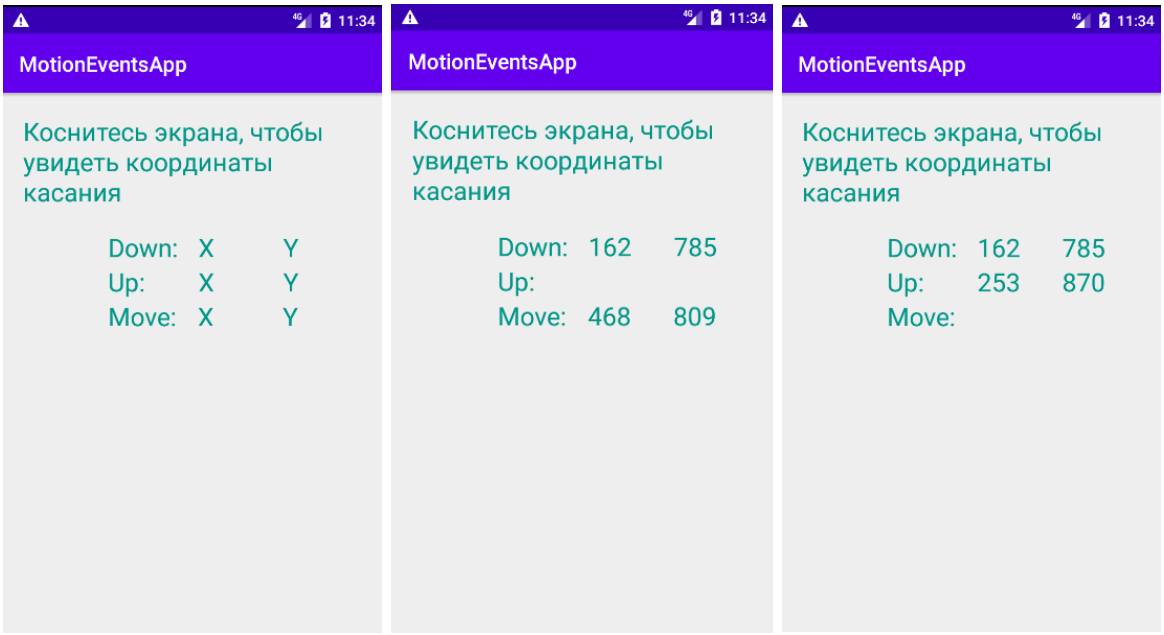

Рисунок 1. Обработка касания окна активити.

Метод getAction () используется для извлечения действия, выполненного пользователем, из параметра события. Это дает необработанные данные, необходимые для определения того, произошел ли жест, который вас интересует. Затем можно выполнить собственную обработку этих событий, чтобы определить, произошел ли жест. Однако если приложение использует стандартные жесты, такие как двойное касание, долгое нажатие и т.д., можно воспользоваться классом GestureDetector. GestureDetector позволяет легко обнаруживать распространенные жесты без самостоятельной обработки отдельных событий касания.

Некоторые устройства могут сообщать о нескольких путях движения одновременно. На мультисенсорных экранах для каждого пальца отображается по одному пути движения. Отдельные пальцы или другие объекты, образующие пути движения, называются указателями (англ. pointer). События движения содержат информацию обо всех указателях, которые в настоящее время активны, даже если некоторые из них не перемещались с момента доставки последнего события.

Каждый указатель имеет уникальный идентификатор, который присваивается при первом касании (ACTION_DOWN или ACTION_POINTER_DOWN). Идентификатор указателя остается действительным до тех пор, пока указатель не перестанет касаться экрана (ACTION_UP или ACTION_POINTER_UP) или, когда жест отменяется (обозначается ACTION_CANCEL).

Если указатель перемещается во время касания (между ACTION_DOWN и ACTION_UP), то отслеживается движение (ACTION_MOVE). Движение содержит самую последнюю точку, а также любые промежуточные точки с момента последнего события касания или движения.

Обработка касаний представления

В качестве альтернативы onTouchEvent () можно прикрепить слушатель View.OnTouchListener к любому объекту View с помощью метода setOnTouchListener () (листинг 4). Это позволяет отслеживать события касания без создания подкласса существующего представления.

Листинг 4. Обработка касания представления.

```
View myView = findViewById(R.id.my_view);
myView.setOnTouchListener(new OnTouchListener() {
    public boolean onTouch(View v, MotionEvent event) {
        //обработка касаний
        return true;
    }
});
```

Избегайте создания слушателя, который возвращает false для события ACTION_DOWN. Если создать такой слушатель, то он не будет вызываться для последующих событий ACTION_MOVE и ACTION_UP, так событие ACTION_DOWN является отправной точкой для всех событий касания.

Обработка жестов

Хотя ваше приложение не должно зависеть от сенсорных жестов для базового поведения (поскольку жесты могут быть доступны не всем пользователям), добавление сенсорного взаимодействия может значительно повысить полезность и привлекательность приложения.

Android предоставляет класс GestureDetector для обнаружения распространенных жестов. Поддерживаются жесты касания (onDown ()), длительного нажатия (onLongPress ()) и т.д. Также можно использовать GestureDetector вместе с методом onTouchEvent (), описанным выше.

Когда создаётся экземпляр GestureDetectorCompat, одним из параметров, которые он принимает, является класс, реализующий интерфейс GestureDetector.OnGestureListener — слушатель жестов. Этот класс уведомляет пользователей, когда произошло определенное событие касания. Чтобы объект GestureDetector мог получать события, переопределяется метод onTouchEvent () представления или активити, а затем экземпляру детектора передаются все наблюдаемые события (листинг 5).

Листинг 5. Обработка жестов.

```
public class MainActivity extends Activity implements
        GestureDetector.OnGestureListener,
        GestureDetector.OnDoubleTapListener{

    private GestureDetectorCompat mDetector;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mDetector = new GestureDetectorCompat(this,this);
        mDetector.setOnDoubleTapListener(this);
    }
```

```
    @Override
    public boolean onTouchEvent(MotionEvent event){
        if (this.mDetector.onTouchEvent(event)) {
            return true;
        }
        return super.onTouchEvent(event);
    }
    @Override
    public boolean onDown(MotionEvent event) {
        //действия при наступлении данного события
        return true;
    }
    @Override
    public boolean onFling(MotionEvent event1, MotionEvent event2,
                           float velocityX, float velocityY) {
        //действия при наступлении данного события
        return true;
    }
    @Override
    public void onLongPress(MotionEvent event) {
        //действия при наступлении данного события
    }
    @Override
    public boolean onScroll(MotionEvent event1, MotionEvent event2,
float distanceX, float distanceY) {
        //действия при наступлении данного события
        return true;
    }
    @Override
    public void onShowPress(MotionEvent event) {
        //действия при наступлении данного события
    }
    @Override
    public boolean onSingleTapUp(MotionEvent event) {
        //действия при наступлении данного события
        return true;
    }
    @Override
    public boolean onDoubleTap(MotionEvent event) {
        //действия при наступлении данного события
        return true;
    }
    @Override
    public boolean onDoubleTapEvent(MotionEvent event) {
        //действия при наступлении данного события
        return true;
    }
    @Override
    public boolean onSingleTapConfirmed(MotionEvent event) {
        //действия при наступлении данного события
        return true;
    }
}
```

В данном коде, возвращаемое значение true от отдельных методов событий касания, указывает, что вы обработали событие касания. Возвращаемое значение false передает событие вниз через стек представления, пока касание не будет успешно обработано.

В методе onCreate () активити также с помощью метода setOnDoubleTapListener () устанавливается слушатель событий, связанных с двойным касанием. Активити должен реализовать интерфейсы GestureDetector.OnGestureListener и GestureDetector.OnDoubleTapListener.

В таблицах 2 и 3 приведены методы слушателей событий жестов и двойного касания соответственно. Эти метода переопределяются в классе активити.

Таблица 2. Методы интерфейса GestureDetector.OnGestureListener.

Метод	**Описание**
onDown ()	Отслеживает появление касания экрана.
onFling ()	Отслеживает появление жеста смахивания.
onLongPress ()	Отслеживает удерживание пальца прижатым к экрану в течении длительного времени.
onScroll ()	Отслеживает появление жеста прокрутки (пролистывания).
onShowPress ()	Отслеживает, что произошло событие касания и больше никаких событий не происходит короткое время.
onSingleTapUp ()	Отслеживает появление жеста одиночного нажатия (клик).

Таблица 3. Методы интерфейса GestureDetector.OnDoubleTapListener.

Метод	**Описание**
onDoubleTap	Отслеживает появление жеста двойного нажатия ("двойной клик").
onDoubleTapEvent	Отслеживает появление события во время выполнения жеста двойного нажатия, включая касание, перемещение, подъем пальца.
onSingleTapConfirmed	Отслеживает появление жеста одиночного нажатия (клик).

Если требуется обработать только несколько жестов, то можно расширить GestureDetector.SimpleOnGestureListener вместо реализации интерфейса GestureDetector.OnGestureListener.

GestureDetector.SimpleOnGestureListener предоставляет реализацию для всех методов событий касания, возвращая false для всех из них. Таким образом, можно переопределить только те методы, которые нужны.

Независимо от того, используете вы GestureDetector.OnGestureListener или нет, лучше всего реализовать метод onDown (), который возвращает true, так как все жесты начинаются с сообщения onDown (). Если вы вернете false из onDown (), как это делает GestureDetector.SimpleOnGestureListener по умолчанию, система сочтёт, что вы хотите игнорировать остальную часть жеста, и другие методы GestureDetector.OnGestureListener не будут вызваны. Это может вызвать неожиданные проблемы в вашем приложении. Единственный случай, когда вы должны вернуть false из onDown (), — это если вы действительно хотите игнорировать весь жест.

Обработка жестов в представлении

В листинге 6 приведён код активити, выполняющего обработку жестов в представлении. Компоновка этого активити содержит две области. В верхней располагается прокручиваемая область ScrollView, в которой содержится текстовое поле. При касании нижней области в текстовом поле в верхней области появляется название произошедшего события касания.

Листинг 6. Обработка жестов в представлении.

```
package com.example.gestureapp;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.view.GestureDetectorCompat;
import android.os.Bundle;
import android.view.GestureDetector;
import android.view.MotionEvent;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    private GestureDetectorCompat myDetector;
    TextView myText;
    View myView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        myText = findViewById(R.id.myText);
```

```
        myView = findViewById(R.id.myView);
        GestureListener myGestures = new GestureListener();
        myDetector = new GestureDetectorCompat(this, myGestures);
        myView.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                return myDetector.onTouchEvent(event);
            }
        });
        myDetector.setOnDoubleTapListener(myGestures);
    }

    public class GestureListener implements
GestureDetector.OnGestureListener,
            GestureDetector.OnDoubleTapListener {
        @Override
        public boolean onDown(MotionEvent event) {
            myText.append("\n onDown");
            return true;
        }
        @Override
        public boolean onFling(MotionEvent event1, MotionEvent event2,
                               float velocityX, float velocityY) {
            myText.append("\n onFling");
            return true;
        }
        @Override
        public void onLongPress(MotionEvent event) {
            myText.append("\n onLongPress");
        }
        @Override
        public boolean onScroll(MotionEvent event1, MotionEvent event2,
float distanceX, float distanceY) {
            myText.append("\n onScroll");
            return true;
        }
        @Override
        public void onShowPress(MotionEvent event) {
            myText.append("\n onShowPress");
        }
        @Override
        public boolean onSingleTapUp(MotionEvent event) {
            myText.append("\n onSingleTapUp");
            return true;
        }
        @Override
        public boolean onDoubleTap(MotionEvent event) {
            myText.append("\n onDoubleTap");
            return true;
        }
        @Override
        public boolean onDoubleTapEvent(MotionEvent event) {
            myText.append("\n onDoubleTapEvent");
            return true;
        }
```

```
        @Override
        public boolean onSingleTapConfirmed(MotionEvent event) {
            myText.append("\n onSingleTapConfirmed");
            return true;
        }
    }
}
```

На рисунке 2 представлена работа данного активити.

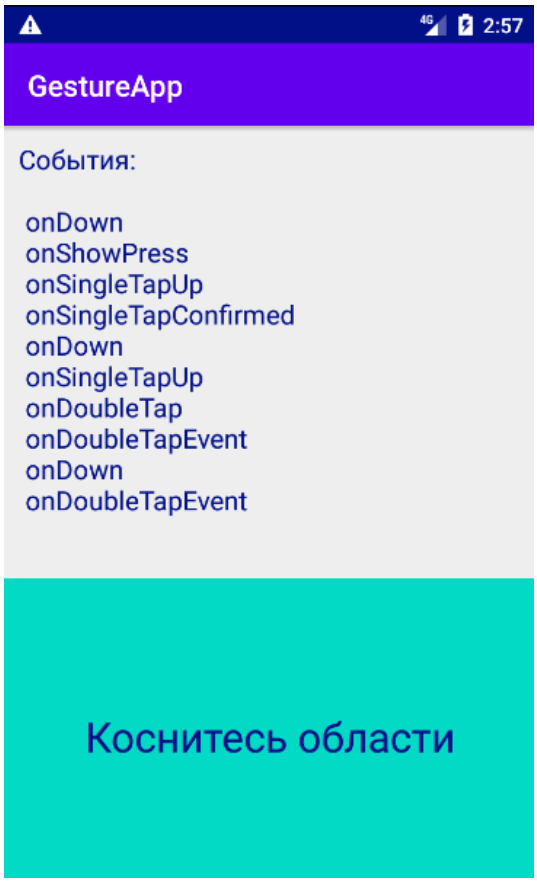

Рисунок 2. Обработка жестов в представлении.

Отслеживание движение в событиях касания

Новое событие onTouchEvent () запускается с событием ACTION_MOVE всякий раз, когда изменяется текущее положение касания, давление или размер. Все эти события записываются в параметре MotionEvent функции onTouchEvent ().

Поскольку прикосновение пальцами не всегда является самой точной формой взаимодействия, обнаружение событий прикосновения часто основывается больше на движении, чем на простом контакте. Чтобы помочь приложениям различать жесты, основанные на движении (например,

смахивание), и жесты, не связанные с движением (например, одиночное касание), Android включает понятие «допуск касания» (англ. touch slop). Это расстояние в пикселях, на которое пользователь может отклониться, прежде чем жест будет интерпретирован как жест, основанный на движении.

Есть несколько разных способов отслеживания движение с помощью жеста, которые могут использоваться в зависимости от потребностей приложения, например:

- По начальной и конечной позиции указателя.
- По направлению, в котором движется указатель.
- По истории перемещения. Можно узнать размер истории, вызвав метод MotionEvent getHistorySize (). Затем можно получить положение, размеры, время и давление каждого из событий, используя методы getHistorical<Value> события движения. История полезна при визуализации следа пальца пользователя, например, для рисования касанием.
- По скорости указателя при перемещении по сенсорному экрану.

Скорость часто является определяющим фактором при отслеживании характеристик жеста или даже при принятии решения о том, произошел ли жест. Чтобы упростить расчет скорости, Android предоставляет класс VelocityTracker. VelocityTracker помогает отслеживать скорость событий касания. Это полезно для жестов, в которых скорость является частью критерия жеста, например, смахивания.

В листинге 7 приведён пример, иллюстрирующий назначение методов VelocityTracker.

Листинг 7. Работа с VelocityTracker.

```
private VelocityTracker mVelocityTracker = null;

@Override
public boolean onTouchEvent(MotionEvent event) {
    int action = event.getActionMasked();
    int index = event.getActionIndex();
    int pointerId = event.getPointerId(index);
```

```
    switch(action) {
        case MotionEvent.ACTION_DOWN:
            if(mVelocityTracker == null) {
                // Получаем новый объект VelocityTracker
                mVelocityTracker = VelocityTracker.obtain();
            }
            else {
                // Сбрасываем VelocityTracker к начальному состоянию
                mVelocityTracker.clear();
            }
            // Добавляем движение в трекер
            mVelocityTracker.addMovement(event);
            break;
        case MotionEvent.ACTION_MOVE:
            mVelocityTracker.addMovement(event);
            mVelocityTracker.computeCurrentVelocity(1000);
velocityX.setText(Integer.toString((int)(mVelocityTracker.getXVelocity(
pointerId))));
velocityY.setText(Integer.toString((int)(mVelocityTracker.getYVelocity(
pointerId))));
            break;
        case MotionEvent.ACTION_UP:
            break;
        case MotionEvent.ACTION_CANCEL:
            // Возвращаем объект VelocityTracker для повторного
            mVelocityTracker.recycle();
            break;
    }
    return true;
}
```

Когда нужно определить скорость вызываем метод computeCurrentVelocity (int units). Значение units представляет количество миллисекунд в промежутке времени для измерения скорости. Значение 1 соответствует пикселям в миллисекунду, 1000 — пикселям в секунду и т.д.

Затем вызываем методы getXVelocity () и getYVelocity (), чтобы получить скорость перемещения по координатам X и Y для каждого указателя. Значение скорости, возвращаемое этими методами, будет положительным, если движение совершается вправо (для X) или вниз (для Y), и отрицательным — если влево (для X) или вверх (для Y).

Обработка мультитач-жестов

Мультитач-жест — это одновременное касание экрана несколькими указателями (пальцами).

Когда несколько указателей касаются экрана одновременно, система генерирует следующие события касания:

1. ACTION_DOWN — для первого указателя, касающегося экрана. Это событие запускает жест. Данные для этого указателя всегда имеют индекс 0 в MotionEvent.
2. ACTION_POINTER_DOWN — для дополнительных указателей, которые касаются экрана после первого. Данные для этого указателя находятся в индексе, возвращаемом getActionIndex ().
3. ACTION_MOVE — изменение произошло во время жеста нажатия.
4. ACTION_POINTER_UP — событие случается, когда экрана перестаёт касаться указатель с индексом не равным 0 (не первый указатель).
5. ACTION_UP — событие случается, когда последний указатель покидает экран.

Отдельные указатели в MotionEvent отслеживаются по индексу и идентификатору каждого указателя. MotionEvent эффективно хранит информацию о каждом указателе в массиве. Индекс указателя — это его позиция в данном массиве. Большинство методов MotionEvent, которые используются для взаимодействия с указателями, принимают в качестве параметра индекс, а не идентификатор указателя.

Каждый указатель также имеет собственный идентификатор, который не изменяется при событиях касания, что позволяет отслеживать отдельный указатель на протяжении всего жеста.

Порядок, в котором отдельные указатели появляются в событии движения, не определен. Таким образом, индекс указателя может изменяться от одного события к другому, но идентификатор указателя гарантированно останется постоянным, пока указатель остается активным.

Каждое новое касание получает минимальный свободный идентификатор, а индексы всегда перестраиваются так, чтобы идентификаторы шли по возрастанию. Идентификатор привязан к касанию

(пока оно длится идентификатор неизменен). А индексы — это просто номера касаний, но эти номера вовсе не означают порядок касаний. Индексы и идентификаторы могут принимать значения от 0 до 9.

Рассмотрим пример:

Когда первый палец касается экрана, он получает нулевые индекс (ix = 0) и идентификатор (id = 0).

Затем второй палец касается экрана и получает ix = 1 и id = 1.

Затем третий палец касается экрана и получает ix = 2 и id = 2.

Если оторвать второй палец от экрана, то третий палец получает ix = 1 и сохраняет id = 2, а первый сохраняет ix = 0 и id = 0.

Если затем оторвать первый палец от экрана, то третий палец получает ix = 0 и сохраняет id = 2.

Затем четвёртый палец касается экрана и получает ix = 0 и id = 0, а третий палец получает ix = 1 и сохраняет id = 2.

Первый палец касается экрана и получает ix = 1 и id = 1, а третий палец получает ix = 2 и сохраняет id = 2, четвёртый сохраняет ix = 0 и id = 0.

Второй палец касается экрана и получает ix = 3 и id = 3, а остальные сохраняют предыдущие значения индексов и идентификаторов:

- четвёртый сохраняет ix = 0 и id = 0,
- первый сохраняет ix = 1 и id = 1,
- третий сохраняет ix = 2 и id = 2.

Чтобы получить идентификатор указателя для его отслеживания во всех последующих событиях движения в жесте, используется метод getPointerId (). Затем для последовательных событий движения используется метод findPointerIndex (), чтобы получить индекс указателя для данного идентификатора указателя в этом событии движения.

Для получения событий MotionEvent следует использовать метод getActionMasked (). В отличие от более старого метода getAction (), getActionMasked () предназначен для работы с несколькими указателями.

Затем можно использовать getActionIndex () для получения индекса указателя, связанного с событием.

События ACTION_UP и ACTION_DOWN содержат в себе индекс касания. По этому индексу мы всегда можем получить идентификатора указателя. Событие ACTION_MOVE информации об индексах не дает. Оно только уведомляет, что происходит какое-то движение.

В листинге 8 приведён код активити, отслеживающего координаты мультитач касаний.

Листинг 8. Отслеживание координат мультитач касаний.

```
package com.example.multitouchapp;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity implements
OnTouchListener {
    int upIndex = 0;
    int downIndex = 0;
    boolean touchFlag = false;
    TextView[] idView = new TextView[10];
    TextView[] xView = new TextView[10];
    TextView[] yView = new TextView[10];
    TextView myText;
    View myView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        myView = findViewById(R.id.myView);
        myText = findViewById(R.id.myText);
        String myPackage = getPackageName();
        Resources myResources = getResources();
        for (int i = 0; i < 10; i++) {
idView[i] = findViewById(myResources.getIdentifier("id"+i, "id",
myPackage));
xView[i] = findViewById(myResources.getIdentifier("x"+i, "id",
myPackage));
yView[i] = findViewById(myResources.getIdentifier("y"+i, "id",
myPackage));
        }
        myView.setOnTouchListener(this);
    }
```

```
    @Override
    public boolean onTouch(View view, MotionEvent event) {
        int actionMask = event.getActionMasked();
        int pointerIndex = event.getActionIndex();
        int pointerCount = event.getPointerCount();

        switch (actionMask) {
            case MotionEvent.ACTION_DOWN:
                touchFlag = true;
            case MotionEvent.ACTION_POINTER_DOWN:
                downIndex = pointerIndex;
                break;
            case MotionEvent.ACTION_UP:
                touchFlag = false;
                for (int i = 0; i < 10; i++) {
                    idView[i].setText("");
                    xView[i].setText("");
                    yView[i].setText("");
                    myText.setText("Координаты касаний");
                }
            case MotionEvent.ACTION_POINTER_UP:
                upIndex = pointerIndex;
                break;
            case MotionEvent.ACTION_MOVE:
                for (int i = 0; i < 10; i++) {
                    idView[i].setText("");
                    xView[i].setText("");
                    yView[i].setText("");
                    if (i < pointerCount) {
idView[i].setText(Integer.toString(event.getPointerId(i)));
xView[i].setText(Integer.toString((int)event.getX(i)));
yView[i].setText(Integer.toString((int)event.getY(i)));
                    } else {
                        idView[i].setText("");
                        xView[i].setText("");
                        yView[i].setText("");
                    }
                }
                break;
        }
         if (touchFlag) {
             myText.setText("Количество касаний: " +
Integer.toString(pointerCount) + "\n" + "Индекс последнего касания: " +
downIndex + "\n" + "Индекс последнего отпускания: " + upIndex );
         }
        return true;
    }
}
```

Для получения событий MotionEvent используем метод getActionMasked (). Индекс касания определяем методом getActionIndex (), а количество текущих касаний методом getPointerCount ().

На рисунке 3 представлена работа данного активити.

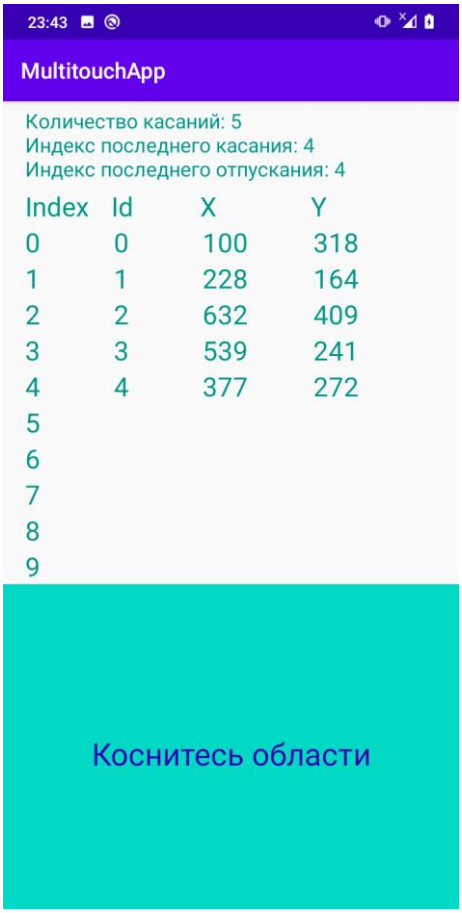

Рисунок 3. Обработка мультитач касаний.

При наступлении события ACTION_DOWN устанавливаем метку касания touchFlag = true. В данной ветке оператора выбора case не ставим break, так как следующая ветка (при ACTION_POINTER_DOWN) также выполнится при ACTION_DOWN.

Если произошло событие ACTION_POINTER_DOWN (или ACTION_DOWN), то в переменную downIndex помещаем индекс касания (последнего прикоснувшегося пальца).

Если произошло событие ACTION_UP, значит последнее касание было прервано и указатели не касаются экрана. Устанавливаем метку touchFlag = false, т.е. касания отсутствуют.

Если произошло событие ACTION_POINTER_UP (или ACTION_UP), то в переменную upIndex помещаем индекс касания (индекс последнего прерванного касания). Когда мы одно за другим прерываем касания, эта

переменная будет последовательно содержать индексы последнего из прерванных касаний.

Если произошло событие ACTION_MOVE, то перебираем все существующие индексы. С помощью pointerCount определяем, какие из них сейчас задействованы и содержат информацию о касаниях. Для них мы отображаем идентификатор, полученный методом getPointerId (), и координаты, полученный с помощью getX () и getY ().

Если в данный момент есть касания, то также отображаем количество касаний.

Задание

Разработайте приложение, позволяющее отслеживать одиночные и мультитач касания сенсорного экрана, а также обнаруживающее жесты с использованием класса GestureDetector.

Контрольные вопросы:

1. Какие методы есть в слушателях событий класса View?
2. Когда вызывается метод onTouch ()?
3. Какие этапы распознавания жестов существуют?
4. Какой метод следует переопределить, чтобы перехватить события касания в активити?
5. Для чего используется метод getAction ()?
6. Для чего используется метод getActionMasked ()?
7. Для чего служит класс MotionEvent?
8. Для чего служит класс GestureDetector?
9. Когда происходит событие ACTION_DOWN?
10. Когда происходит событие ACTION_MOVE?
11. Когда происходит событие ACTION_UP?
12. Когда происходит событие ACTION_POINTER_DOWN?
13. Когда происходит событие ACTION_ POINTER_UP?
14. Какие методы есть в интерфейсе GestureDetector.OnGestureListener?

15. Какие методы есть в интерфейсе GestureDetector.OnDoubleTapListener?
16. Какие способы отслеживания движения с помощью жеста существуют?
17. Для чего служит класс VelocityTracker?
18. Для чего используется метод getPointerId ()?
19. Для чего используется метод findPointerIndex ()?
20. Для чего используется метод getActionIndex ()?
21. Для чего используется метод getPointerCount ()?