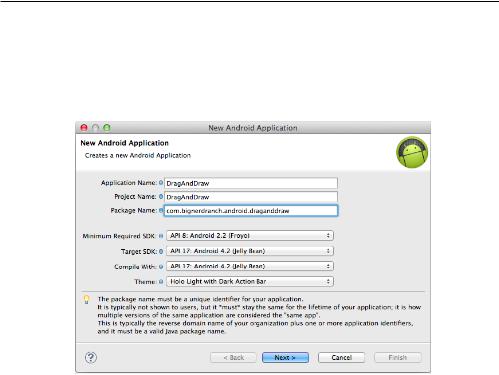
**Пользовательские представления и события касания**

В этой главе мы займемся обработкой событий касания. Для этого мы создадим субкласс View с именемBoxDrawingView. На этом представлении пользователь рисует прямоугольники, прикасаясь к экрану и перемещая палец.

**Рис. 32.2.**Создание проекта DragAndDraw



**Создание проекта DragAndDraw**

Класс BoxDrawingView занимает центральное место в новом проекте DragAndDraw. Выполните командуNew Android Application Project. Задайте параметры проекта, как показано на рис. 32.2, и создайте пустую активность с именемDragAndDrawActivity.

**Создание класса DragAndDrawActivity**

Класс DragAndDrawActivity представляет собой субклассSingleFragmentActivity, который заполняет обычный макет с одним фрагментом. На панелиPackage

Explorer скопируйте файл SingleFragmentActivity.javaв пакет com.bignerdranch.android. draganddraw. Затем скопируйте файл activity\_fragment.xmlв каталог res/layoutпроекта DragAndDraw.

В файле DragAndDrawActivity.java объявитеDragAndDrawActivityсубклассомSingleFragmentActivity. Этот класс должен создавать экземплярDragAndDrawFragment

(класс, который будет создан следующим).

**Листинг 32.1.**Изменение активности (DragAndDrawActivity.java)

public class DragAndDrawActivity extends **~~Activity~~SingleFragmentActivity**{

**~~@Override~~**

**~~public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_drag\_and\_draw);~~**

**~~}~~**

**~~@Override~~**

**~~public boolean onCreateOptionsMenu(Menu menu) { getMenuInflater().inflate(R.menu.activity\_drag\_and\_draw, menu); return true;~~**

**~~}~~**



activity\_drag\_

|  |  |
| --- | --- |
|  |  |
|  |  |

**@Override**

**public Fragment createFragment() { return new DragAndDrawFragment();**

**}**

**}**

**Создание класса DragAndDrawFragment**

Чтобы подготовить макет DragAndDrawFragment, переименуйте файлand\_draw.xmlв fragment\_drag\_and\_draw.xml.

Макет DragAndDrawFragment в конечном итоге будет состоять изBoxDrawingView — пользовательского представления, которое мы собираемся написать. Весь графический вывод и обработка событий касания будут реализованы вBoxDrawingView.

Создайте новый класс с именем DragAndDrawFragment и назначьте его суперклассом

android.support.v4.app.ListFragment. Переопределите метод onCreateView(…), чтобы он заполнял макет fragment\_drag\_and\_draw.xml.

**Листинг 32.2.**Создание фрагмента (DragAndDrawFragment.java)

public class DragAndDrawFragment extends Fragment {

**@Override**

**public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle savedInstanceState) {**

**View v = inflater.inflate(R.layout.fragment\_drag\_and\_draw, parent, false); return v;**

**}**

}

Запустите приложение DragAndDraw и убедитесь в том, что настройка была выполнена правильно.

**Создание нестандартного представления**

Android предоставляет много превосходных стандартных представлений и виджетов, но иногда требуется создать нестандартное представление с визуальным оформлением, полностью уникальным для вашего приложения.

Все многообразие нестандартных представлений можно условно разделить на две общие категории:

*простые представления*— простое представление может быть устроено достаточно сложно; «простым» оно называется только потому, что не имеет дочерних представлений. Простое представление почти всегда также выполняет нестандартную прорисовку;

*составные представления*— состоят из других объектов представлений. Составные представления обычно управляют дочерними представлениями, но не занимаются своей прорисовкой. Вместо этого каждому дочернему представлению делегируется своя часть работы по прорисовке.

Создание нестандартного представления состоит из трех шагов:

Выбор суперкласса. Для простого нестандартного представления Viewпредоставляет пустой «холст» для рисования, поэтому этот выбор является наиболее распространенным. Для составных нестандартных представлений выберите подходящий класс макета.

Субклассируйте выбранный класс и переопределите как минимум один конструктор из суперкласса или создайте собственный конструктор, вызывающий один из конструкторов суперкласса.

Переопределите другие ключевые методы для настройки поведения.

**Создание класса BoxDrawingView**

Класс BoxDrawingView относится к категории простых представлений и является прямым субклассомView.

Создайте новый класс с именем BoxDrawingView и назначьтеView его суперклассом. Добавьте в файлBoxDrawingView.java два конструктора.

**Листинг 32.3.**Исходная реализация BoxDrawingView (BoxDrawingView.java)public class BoxDrawingView extends View {

**//Используется при создании представления в коде public BoxDrawingView(Context context) {**

**this(context, null);**

**}**

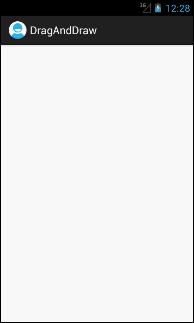
**//Используется при заполнении представления по разметке XML public BoxDrawingView(Context context, AttributeSet attrs) {**

**super(context, attrs);**

**}**

}

Два конструктора нужны потому, что экземпляр представления может создаваться как в коде, так и по файлу разметки. Представления, созданные на базе файла макета,



|  |  |
| --- | --- |
|  |  |
|  |  |

получают экземпляр AttributeSet с атрибутами XML, заданными в XML. Даже если вы не собираетесь использовать оба конструктора, их рекомендуется включить.

Затем обновите файл макета fragment\_drag\_and\_draw.xml, чтобы в нем использовалось новое представление.

**Листинг 32.4.**Включение Add BoxDrawingView в макет (fragment\_drag\_and\_draw.xml)

**~~<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:tools="http://schemas.android.com/tools" android:layout\_width="match\_parent"~~**

**~~android:layout\_height="match\_parent"~~**

**~~>~~**

**~~<TextView android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" android:layout\_centerHorizontal="true" android:layout\_centerVertical="true" android:text="@string/hello\_world" />~~**

**~~</RelativeLayout>~~**

**<com.bignerdranch.android.draganddraw.BoxDrawingView**

**xmlns:android="http://schemas.android.com/apk/res/android" android:layout\_width="match\_parent" android:layout\_height="match\_parent"**

**/>**

**Рис. 32.4.**BoxDrawingView без прямоугольников

Чтобы заполнитель макетов нашел класс BoxDrawingView, вы должны использовать полностью уточненное имя. Заполнитель просматривает файл макета, создавая экземплярыView. Если имя элемента будет неполным, то заполнитель ищет класс с указанным именем в пакетахandroid.view иandroid.widget. Если класс находится в другом месте, заполнитель его не найдет, и в приложении произойдет сбой. По этой причине для классов, не входящих вandroid.view иandroid.widget, необходимо всегда задавать полностью уточненное имя.

Запустите приложение DragAndDraw и убедитесь в том, что настройка была выполнена правильно. Правда, пока на экране нет ничего, кроме пустого представления.

На следующем этапе мы научим BoxDrawingView прослушивать события касания и использовать содержащуюся в них информацию для рисования прямоугольников на экране.

**Обработка событий касания**

Для прослушивания событий касания можно назначить слушателя события при помощи следующего метода класса View:

public void setOnTouchListener(View.OnTouchListener l)

Этот метод работает почти так же, как setOnClickListener(View.OnClickListener). Вы предоставляете реализациюView.OnTouchListener, а слушатель вызывается каждый раз, когда происходит событие касания.

Но поскольку мы субклассируем View, можно пойти по сокращенному пути и переопределить следующий метод классаView:

public boolean onTouchEvent(MotionEvent event)

Этот метод получает экземпляр MotionEvent — класса, описывающего событие касания, включая его позицию и действие. Действие описывает стадию события.

|  |  |
| --- | --- |
| **Константы действий** | **Описание** |
|  |  |
| ACTION\_DOWN | Пользователь прикоснулся к экрану |
|  |  |
| ACTION\_MOVE | Пользователь перемещает палец по экрану |
|  |  |
| ACTION\_UP | Пользователь отводит палец от экрана |
|  |  |
| ACTION\_CANCEL | Родительское представление перехватило событие касания |
|  |  |

В своей реализацияи onTouchEvent(…) для проверки действия можно воспользоваться следующим методом классаMotionEvent:

public final int getAction()

Добавьте в файл BoxDrawingView.java тег для журнала и реализациюonTouch(…), которая регистрирует в журнале информацию о каждом из четырех разных действий.

http://www.studfiles.ru/html/2706/1137/html_K2MGkhC3f0.fGLx/htmlconvd-G_3cW2525x1.jpg

Обработка событий касания    **525**

**Листинг 32.5.**Реализация BoxDrawingView (BoxDrawingView.java)

public class BoxDrawingView extends View {

**public static final String TAG = "BoxDrawingView";**

...

public boolean onTouchEvent(MotionEvent event) {

**PointF curr = new PointF(event.getX(), event.getY());**

**Log.i(TAG, "Received event at x=" + curr.x + ", y=" + curr.y + ":");**

**switch (event.getAction()) {**

**case MotionEvent.ACTION\_DOWN: Log.i(TAG, " ACTION\_DOWN"); break;**

**case MotionEvent.ACTION\_MOVE: Log.i(TAG, " ACTION\_MOVE"); break;**

**case MotionEvent.ACTION\_UP: Log.i(TAG, " ACTION\_UP"); break;**

**case MotionEvent.ACTION\_CANCEL: Log.i(TAG, " ACTION\_CANCEL"); break;**

**}**

return true;

}

}

Обратите внимание: координаты X и Y упаковываются в объекте PointF. В оставшейся части этой главы эти два значения обычно будут передаваться вместе.PointF — предоставленный Androidкласс-контейнер,который решает эту задачу за вас.

Запустите приложение DragAndDraw и откройте LogCat. Прикоснитесь к экрану и проведите пальцем. Вы увидите в журнале сообщения с координатами X и Y каждого действия касания, полученного BoxDrawingView.

**Отслеживание перемещений между событиями**

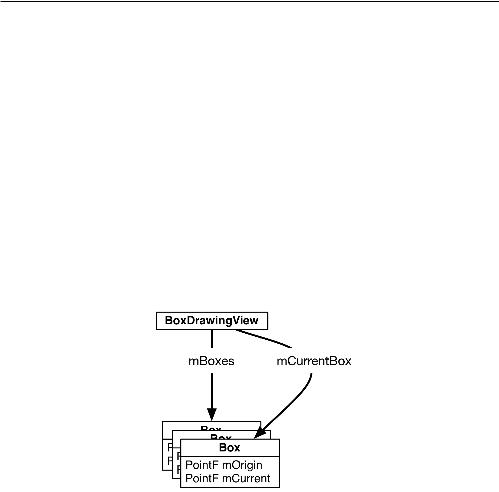
Класс BoxDrawingView должен рисовать прямоугольники, а не регистрировать координаты. Для этого необходимо решить ряд задач.

Прежде всего для определения прямоугольника нам понадобятся:базовая точка (в которой было сделано исходное касание);текущая точка (в которой находится палец).

Следовательно, для определения прямоугольника необходимо отслеживать данные от нескольких событий MotionEvent. Данные будут храниться в объектеBox.

Создайте класс с именем Box для хранения данных, определяющих прямоугольник.

**Рис. 32.5.**Объекты в DragAndDraw



**Листинг 32.6.**Класс Box (Box.java)

public class Box {

**private PointF mOrigin; private PointF mCurrent;**

**public Box(PointF origin) { mOrigin = mCurrent = origin;**

**}**

**public void setCurrent(PointF current) { mCurrent = current;**

**}**

**public PointF getOrigin() { return mOrigin;**

**}**

}

Когда пользователь прикасается к BoxDrawingView, новый объектBox создается и включается в массив существующих прямоугольников (рис. 32.5).

Добавьте в BoxDrawingView код, использующий новый объектBox для отслеживания текущего состояния рисования.

**Листинг 32.7.**Добавление методов жизненного цикла событий касания

(BoxDrawingView.java)

public class BoxDrawingView extends View {

public static final String TAG = "BoxDrawingView";

**private Box mCurrentBox;**

**private ArrayList<Box> mBoxes = new ArrayList<Box>();**

...

public boolean onTouchEvent(MotionEvent event) {

PointF curr = new PointF(event.getX(), event.getY());

http://www.studfiles.ru/html/2706/1137/html_K2MGkhC3f0.fGLx/htmlconvd-G_3cW2527x1.jpg

Обработка событий касания    **527**

switch (event.getAction()) {

case MotionEvent.ACTION\_DOWN: Log.i(TAG, " ACTION\_DOWN");

**// Reset drawing state mCurrentBox = new Box(curr); mBoxes.add(mCurrentBox);**break;

case MotionEvent.ACTION\_MOVE: Log.i(TAG, " ACTION\_MOVE"); **if (mCurrentBox != null) {**

**mCurrentBox.setCurrent(curr);**

**invalidate();**

**}**

break;

case MotionEvent.ACTION\_UP: Log.i(TAG, " ACTION\_UP");**mCurrentBox = null;**break;

case MotionEvent.ACTION\_CANCEL: Log.i(TAG, " ACTION\_CANCEL");**mCurrentBox = null;**

break;

}

return true;

}

}

При каждом получении события ACTION\_DOWN в полеmCurrentBox сохраняется новый объектBox с базовой точкой, соответствующей позиции события. Этот объектBoxдобавляется в массив прямоугольников (в следующем разделе, когда мы займемся прорисовкой,BoxDrawingView будет выводить каждый объектBox из массива).

В процессе перемещения пальца по экрану приложение обновляет mCurrentBox. mCurrent. Затем, когда касание отменяется или палец отходит от экрана, полеmCurrentBoxобнуляется для завершения операции. ОбъектBox завершен; он сохранен в массиве и уже не будет обновляться событиями перемещения.

Обратите внимание на вызов invalidate() в случаеACTION\_MOVE. Он заставляетBoxDrawingView перерисовать себя, чтобы пользователь видел прямоугольник в процессе перетаскивания. Мы подошли к следующему шагу: рисованию прямоугольников на экране.

**Рисование внутри onDraw(…)**

При запуске приложения все его представления недействительны (invalid). Это означает, что они ничего не вывели на экран. Для исправления ситуации Android вызывает метод draw() объектаView верхнего уровня. В результате представление перерисовывает себя, что заставляет его потомков перерисовать себя. Затем потомки этих потомков перерисовывают себя и так далее вниз по иерархии. Когда все

http://www.studfiles.ru/html/2706/1137/html_K2MGkhC3f0.fGLx/htmlconvd-G_3cW2528x1.jpg

представления в иерархии перерисуют себя, объект View верхнего уровня перестает быть недействительным.

Чтобы вмешаться в процесс прорисовки, следует переопределить следующий метод

View:

protected void onDraw(Canvas canvas)

Вызов invalidate(), выполняемый в ответ на действиеACTION\_MOVE вonTouchEvent(…), снова делает объектBoxDrawingView недействительным. Это заставляет его перерисовать себя и приводит к повторному вызовуonDraw(…).

Обратите внимание на параметр Canvas.Canvas иPaint — два главных класса, используемых при рисовании в Android.

Класс Canvas содержит все выполняемые операции графического вывода. Методы, вызываемые для объектаCanvas, определяют, где и что выводится — линия, круг, слово или прямоугольник.

Класс Paint определяет, как будут выполняться эти операции. Методы, вызываемые для объектаPaint, определяют характеристики вывода — должны ли фигуры заполняться, каким шрифтом должен выводиться текст, каким цветом должны выводиться линии и т. д.

В файле BoxDrawingView.java создайте два объектаPaint в конструктореBoxDrawingViewдля XML.

**Листинг 32.8.**Создание объектов Paint (BoxDrawingView.java)

public class BoxDrawingView extends View {

private static final String TAG = "BoxDrawingView";

private ArrayList<Box> mBoxen = new ArrayList<Box>(); private Box mCurrentBox;

**private Paint mBoxPaint; private Paint mBackgroundPaint;**

...

// Используется при заполнении представления по разметке XMLpublic BoxDrawingView(Context context, AttributeSet attrs) {

super(context, attrs);

**//Прямоугольники рисуются полупрозрачным красным цветом (ARGB) mBoxPaint = new Paint();**

**mBoxPaint.setColor(0x22ff0000);**

**//Фон закрашивается серовато-белымцветом**

**mBackgroundPaint = new Paint(); mBackgroundPaint.setColor(0xfff8efe0);**

}

}

После создания объектов Paint можно переходить к рисованию прямоугольников на экране.