

Android 6

Двумерная графика и обработка касаний

# Рассматриваемые вопросы

- Обнаружение событий, связанных с касанием экрана, перемещением пальца вдоль экрана и отведением пальца от экрана
- Обработка многоточечных касаний экрана и возможность рисования несколькими пальцами одновременно
- Использование объекта `SensorManager` и акселерометра для обнаружения событий перемещения
- Применение объектов `Paint` для определения цвета и ширины линии
- Использование объектов `Path` для хранения данных линий и объектов `Canvas` для рисования линий на `Bitmap`
- Создание меню и отображение команд на панели действий
- Использование инфраструктуры печати и класса `PrintHelper` из `Android Support Library` для вывода изображений на печать
- Использование новой модели разрешений `Android 6.0` для сохранения изображений во внешнем хранилище
- Добавление библиотек в приложение с помощью системы сборки `Gradle`

# Целевое приложение



## Функции

- Рисование одним или несколькими пальцами
- Изменение цвета и толщины линии
- Очистка экрана
- Сохранение рисунка на устройстве
- Вывод рисунка на печать
- Запрос разрешения на сохранение файла
- Динамическое формирование панели приложения в зависимости от размера экрана

# Используемые возможности

- методы жизненного цикла активности и фрагмента (**onResume, onPause**)
- пользовательские представления (расширение **View**)
- прослушивание событий акселерометра (**SensorManager**)
- пользовательские реализации **DialogFragment**
- предотвращение одновременного появления нескольких диалоговых окон (**onAttach, onDetach**)
- рисование с использованием **Canvas, Paint, Bitmap**
- обработка событий многоточечных касаний и хранение данных линий (объекты **Path, MotionEvent**, метод **onTouchEvent**)
- сохранение данных на устройстве (объекты **ContentResolver, MediaStore**)
- печать изображения (класс **PrintHelper**)
- разрешение на доступ к ресурсам
- добавление зависимостей проекта в Gradle

# Создание проекта

- Имя проекта: L4 Doodlz
- Android 6, API 23
- Шаблон: **Basic Activity**
- флажок **Use a Fragment**
- Добавить значок в проект
- Удалить в макете компонент TextView с текстом “Hello World!” и кнопку FloatingActionButton

# Добавление библиотеки поддержки

Для использования класса PrintHelper в приложении необходима библиотека поддержки Android Support Library.

The image shows the 'Project Structure' dialog in an IDE, specifically the 'Dependencies' tab. The 'Dependencies' list includes:

Dependency	Scope
{include=[*.jar], dir=libs}	Implementation
com.android.support:appcompat-v7:26.1.0	Implementation
com.android.support.constraint:constraint-layout:1.0.2	Implementation
com.android.support:design:26.1.0	Implementation
junit:junit:4.12	Implementation
com.android...	Unit Test imple...

The 'Choose Library Dependency' dialog is open, showing search results for 'com.android.support:support-v4:27.0.1'. The search results include:

- com.android.support:support-annotations (com.android.support:support-annotations:27.0.1)
- com.android.support:support-v4 (com.android.support:support-v4:27.0.1)**
- com.android.support:support-v13 (com.android.support:support-v13:27.0.1)
- com.android.support:appcompat-v7 (com.android.support:appcompat-v7:27.0.1)
- com.android.support:support-vector-drawable (com.android.support:support-vector-drawable:27.0.1)
- com.android.support:design (com.android.support:design:27.0.1)
- com.android.support:gridlayout-v7 (com.android.support:gridlayout-v7:27.0.1)
- com.android.support:mediarouter-v7 (com.android.support:mediarouter-v7:27.0.1)

См. <https://developer.android.com/topic/libraries/support-library/index.html>

# Определение строковых ресурсов

Ключ	Значение по умолчанию
button_erase	Erase Image
button_set_color	Set Color
button_set_line_width	Set Line Width
line_imageview_description	This displays the line thickness
label_alpha	Alpha
label_red	Red
label_green	Green
label_blue	Blue
menuitem_color	Color
menuitem_delete	Erase Drawing
menuitem_line_width	Line Width
menuitem_save	Save

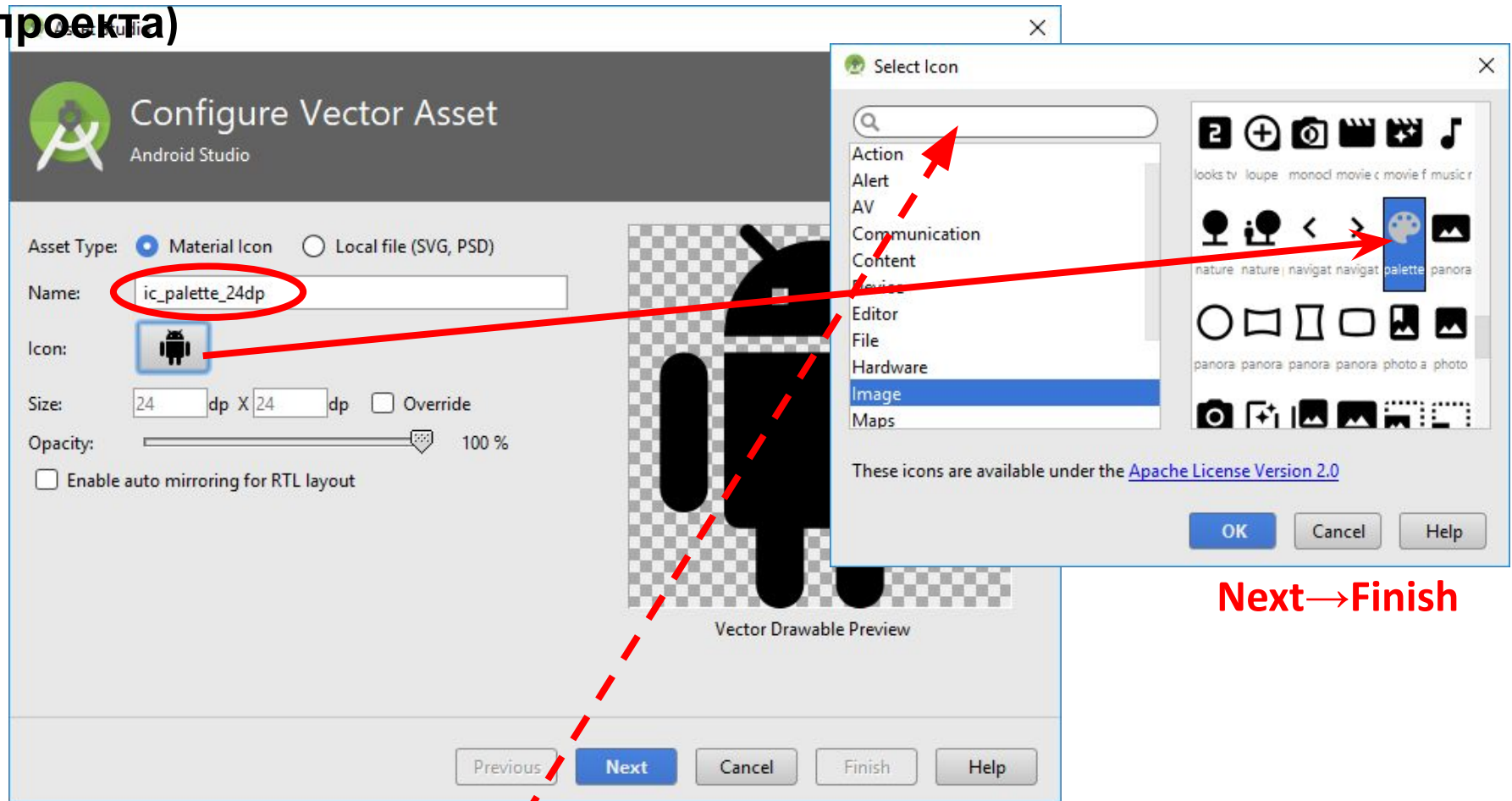
# Определение строковых ресурсов

Ключ	Значение по умолчанию
menuitem_print	Print
message_erase	Erase the drawing?
message_error_saving	There was an error saving the image
message_saved	Your saved painting can be viewed in the Photos app by selecting Device Folders from that app's menu [Примечание: \' — служебная последовательность для представления апострофа ('). Без префикса \ IDE выдает сообщение «Перед апострофом нет символа \».]
message_error_printing	Your device does not support printing
permission_explanation	To save an image, the app requires permission to write to external storage
title_color_dialog	Choose Color
title_line_width_dialog	Choose Line Width



# Импорт необходимых значков меню

**File**→**New**→**Vector Asset** (при активной корневой папке проекта)



**Next**→**Finish**

Аналогично добавить ресурсы: brush, delete, save, print

(найти с помощью поля поиска в окне Select Icon)  
В XML-файлах заменить значение fillColor на

@android:color/white

# Создание меню

- Удалить файл main.xml, а также методы onCreateOptionsMenu() и onOptionsItemSelected() из класса MainActivity
- Создать новый ресурс меню (res/menu → New → Menu Resource File) под именем doodle\_fragment\_menu

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
```

```
<item
  android:id="@+id/color"
  android:title="@string/menuitem_color"
  android:icon="@drawable/ic_palette_24dp"
  app:showAsAction="ifRoom">
</item>
```

Разместить на панели приложения только при наличии свободного места

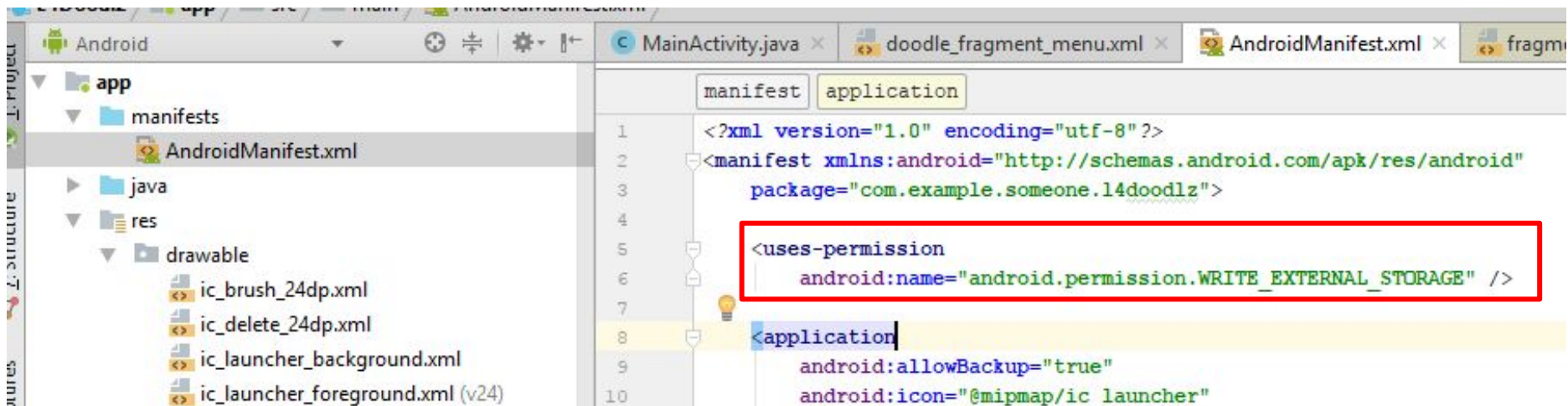
```
</menu>
```

- аналогично остальные элементы меню

Id	Title
@+id/line_width	@string/menuitem_line_width
@+id/delete_drawing	@string/menuitem_delete
@+id/save	@string/menuitem_save
@+id/print	@string/menuitem_print

# Добавление разрешения в манифест

- Каждое приложение должно указать все используемые разрешения в файле AndroidManifest.xml.

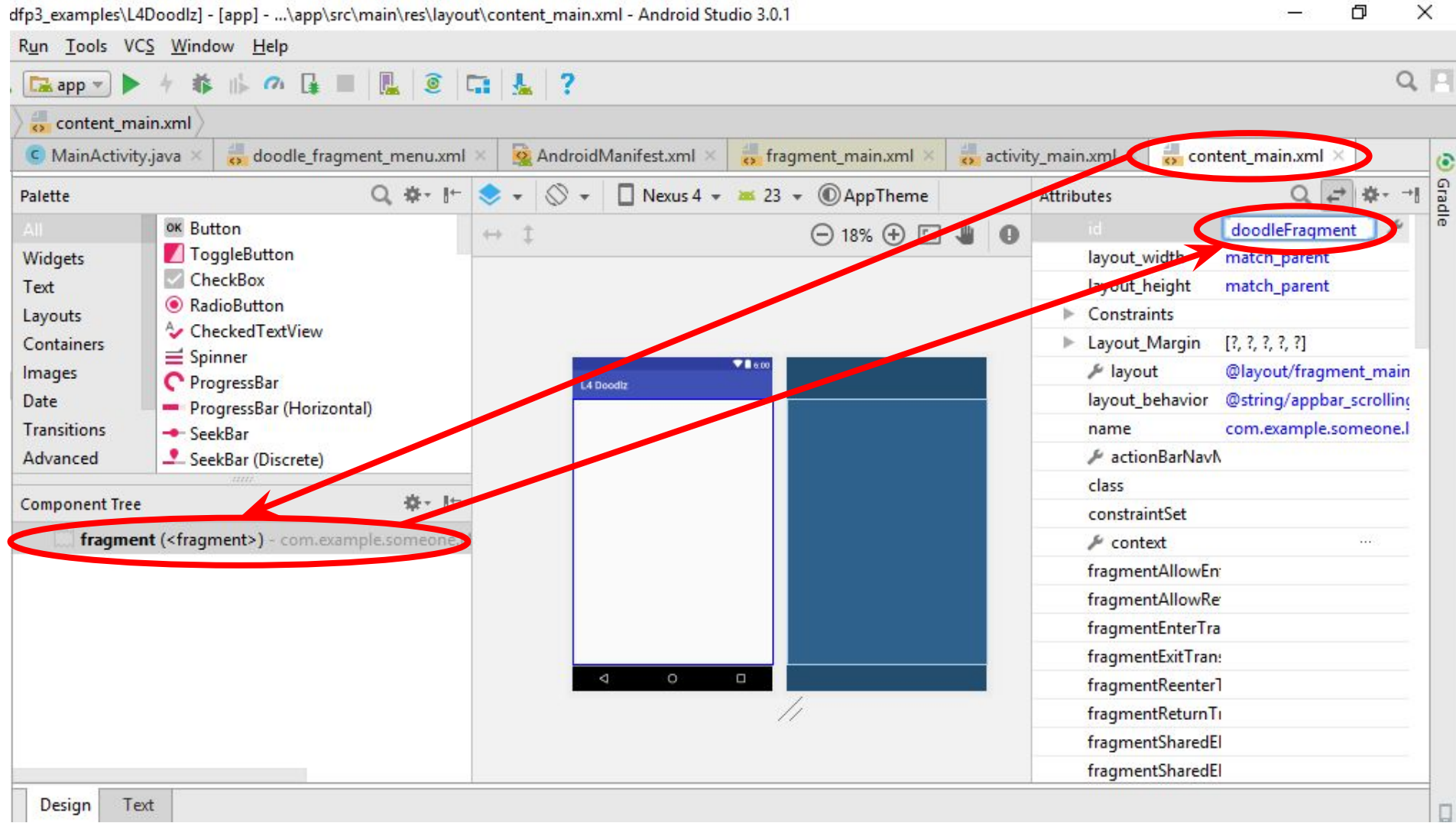


The screenshot shows an IDE window with the AndroidManifest.xml file open. The file content is as follows:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.example.someone.l4doodlz">
4
5     <uses-permission
6         android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
7
8     <application
9         android:allowBackup="true"
10        android:icon="@mipmap/ic_launcher"
```

The `<uses-permission>` tag on line 5 is highlighted with a red box, indicating the addition of the `android.permission.WRITE_EXTERNAL_STORAGE` permission.

# Построение графического интерфейса



# Построение графического интерфейса

- Создать новый java-

класс

Create New Class

Name: DoodleView

Kind: Class

Superclass: android.view.View

Interface(s):

Package:

Visibility:  Public  Package Private

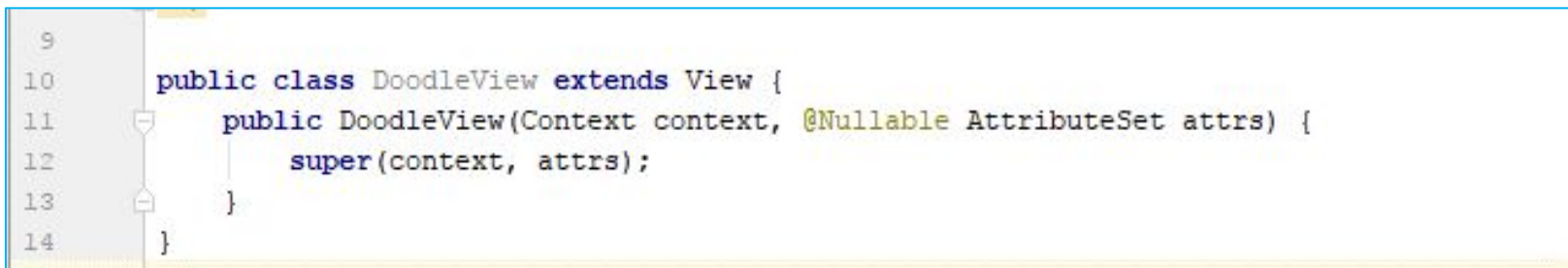
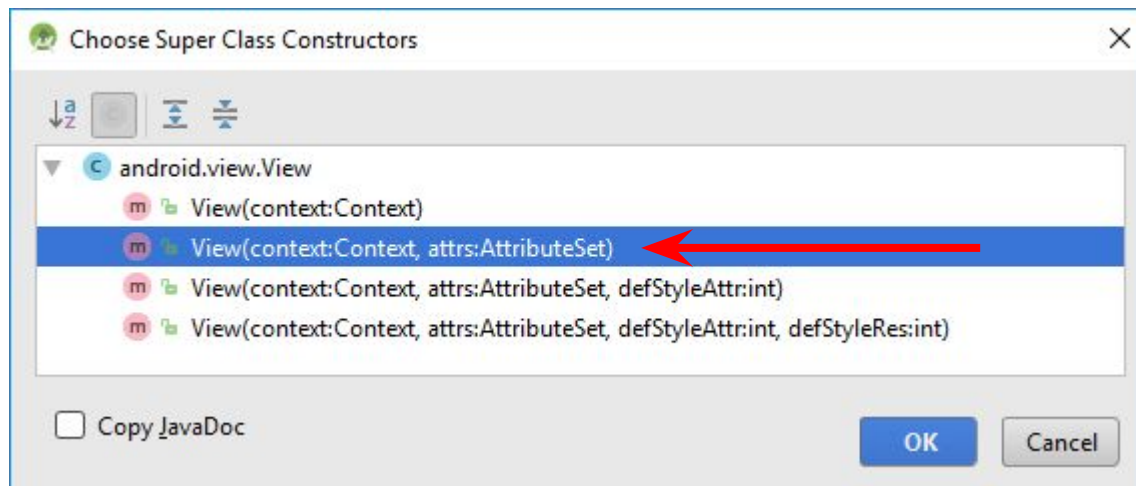
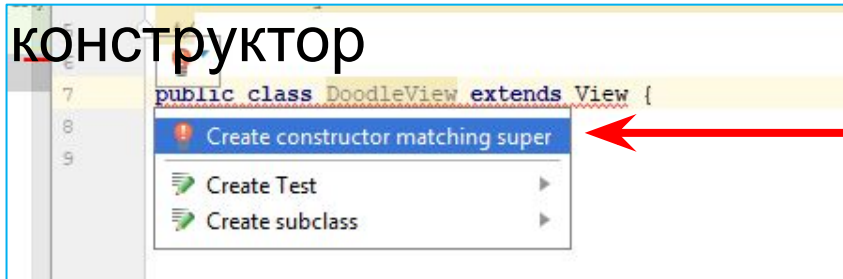
Modifiers:  None  Abstract  Final

Show Select Overrides Dialog

OK Cancel Help

# Построение графического интерфейса

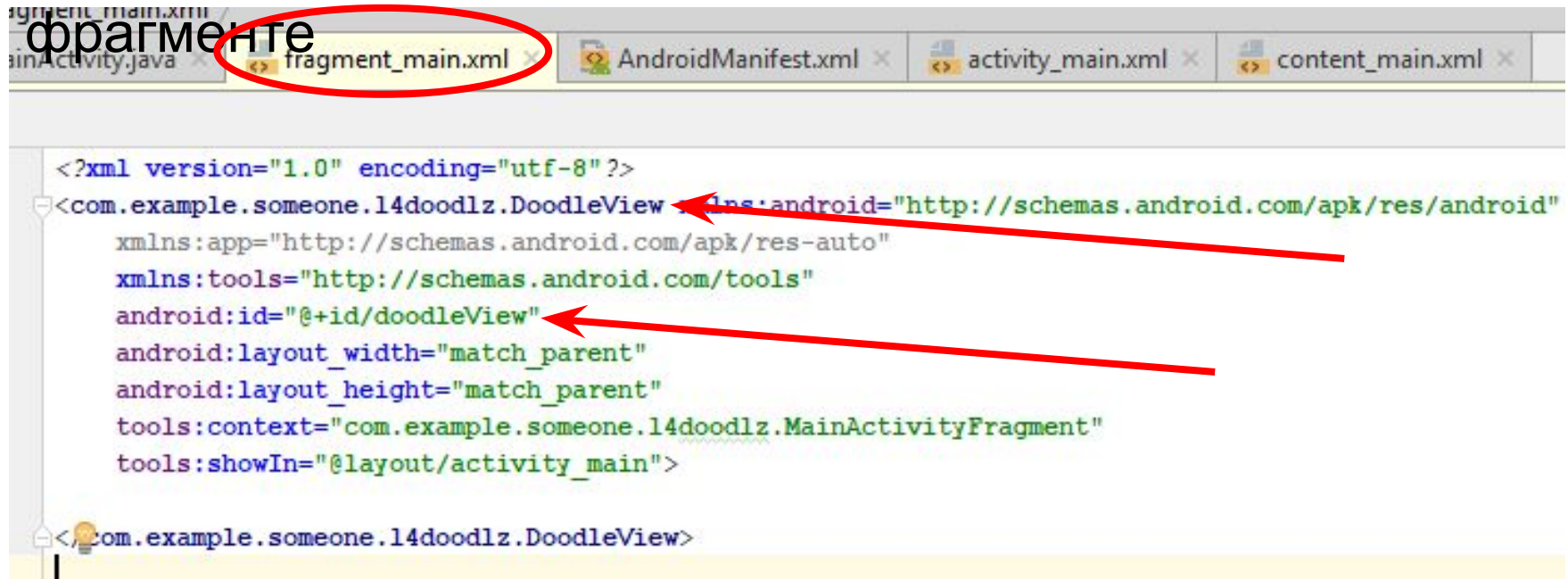
- Описать конструктор





# Построение графического интерфейса

- Использовать созданный класс во фрагменте



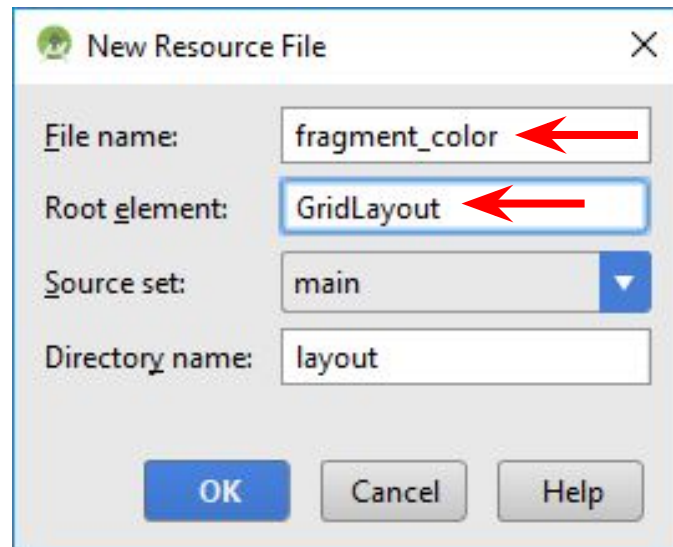
The screenshot shows an IDE window with several tabs: 'main.xml', 'MainActivity.java', 'fragment\_main.xml', 'AndroidManifest.xml', 'activity\_main.xml', and 'content\_main.xml'. The 'fragment\_main.xml' tab is active and contains the following XML code:

```
<?xml version="1.0" encoding="utf-8"?>
<com.example.someone.l4doodlz.DoodleView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/doodleView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.someone.l4doodlz.MainActivityFragment"
    tools:showIn="@layout/activity_main">
</com.example.someone.l4doodlz.DoodleView>
```

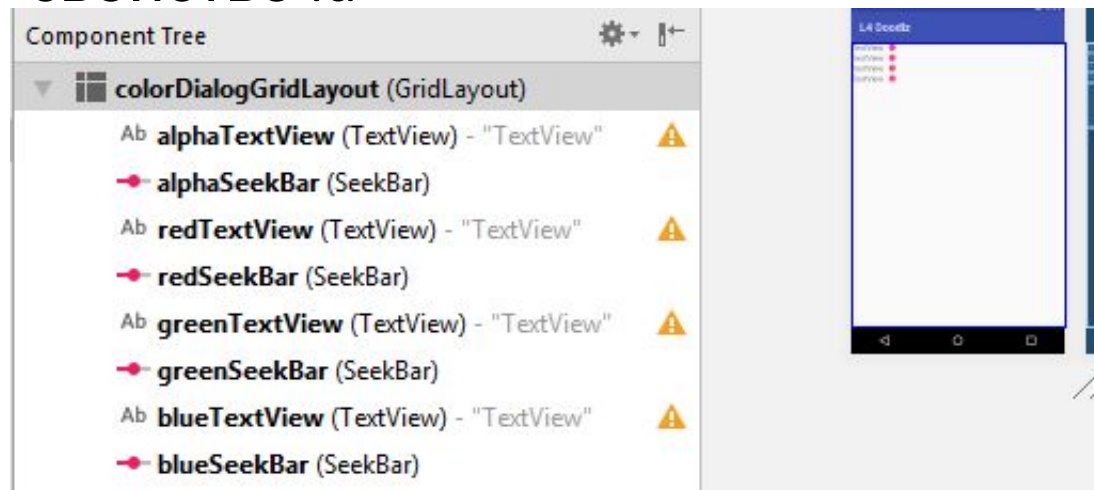
Two red arrows point to the class name 'com.example.someone.l4doodlz.DoodleView' in the opening and closing tags of the XML element. The word 'фрагменте' from the bullet point above is circled in red in the original image.

## Макет для выбора цвета

- Создать новый ресурс макета (res/layout → New → Layout Resource File) под именем fragment\_color
- GridLayout.id = colorDialogGridLayout
- colorDialogGridLayout.columnCount = 2



- Добавить компоненты TextView и SeekBar и задать им СВОЙСТВО id





# Построение графического интерфейса.

## Макет для выбора цвета

- Добавить colorView в макет fragmentColor

```
<View
```

```
    android:layout_width="wrap_content"  
    android:layout_height="@dimen/color_view_height"  
    android:id="@+id/colorView"  
    android:layout_column="0"  
    android:layout_columnSpan="2"  
    android:layout_gravity="fill_horizontal"/>
```

```
</GridLayout>
```

- Задать ресурс @dimen/color\_view\_height=80dp

## Макет для выбора цвета

- Задать свойства компонентов

Компонент	Свойство	Значение
colorDialogGridLayout	columnCount	2
	orientation	vertical
	useDefaultMargins	true
	padding top	@dimen/activity_vertical_margin
	padding bottom	@dimen/activity_vertical_margin
	padding left	@dimen/activity_horizontal_margin
	padding right	@dimen/activity_horizontal_margin
alphaTextView	Параметры layout	
	layout:column	0
	layout:gravity	right, center_vertical
	layout:row	0
	Другие свойства	
	text	@string/label_alpha

## Макет для выбора цвета

alphaSeekBar	Параметры layout	
	layout:column	1
	layout:gravity	fill_horizontal
	layout:row	0
	Другие свойства	
	max	255
redTextView	Параметры layout	
	layout:column	0
	layout:gravity	right, center_vertical
	layout:row	1
	Другие свойства	
	text	@string/label_red
redSeekBar	Параметры layout	
	layout:column	1
	layout:gravity	fill_horizontal
	layout:row	1
	Другие свойства	
	max	255

## Макет для выбора цвета

greenTextView	Параметры layout	
	layout:column	0
	layout:gravity	right, center_vertical
	layout:row	2
	Другие свойства	
	text	@string/label_red
greenSeekBar	Параметры layout	
	layout:column	1
	layout:gravity	fill_horizontal
	layout:row	2
	Другие свойства	
	max	255
blueTextView	Параметры layout	
	layout:column	0
	layout:gravity	right, center_vertical
	layout:row	3
	Другие свойства	
	text	@string/label_red

# Построение графического интерфейса.

## Макет для выбора цвета

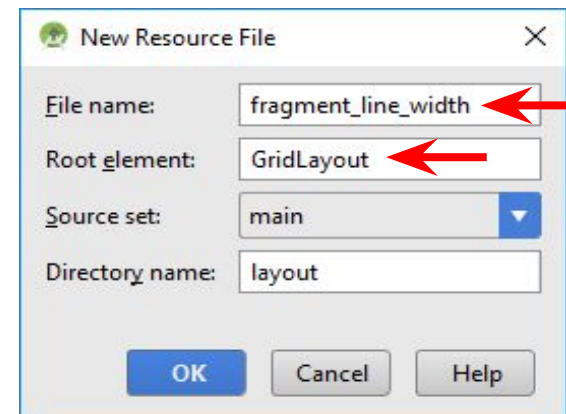
blueSeekBar	Параметры layout	
	layout:column	1
	layout:gravity	fill_horizontal
	layout:row	3
	Другие свойства	
	max	255
colorView	Параметры layout	
	layout:height	@dimen/color_view_height
	layout:column	0
	layout:columnSpan	2
	layout:gravity	fill_horizontal

Если ресурсы `@dimen/activity_horizontal_margin` и `@dimen/activity_horizontal_margin` неизвестны, то добавить их со значениями 16dp.

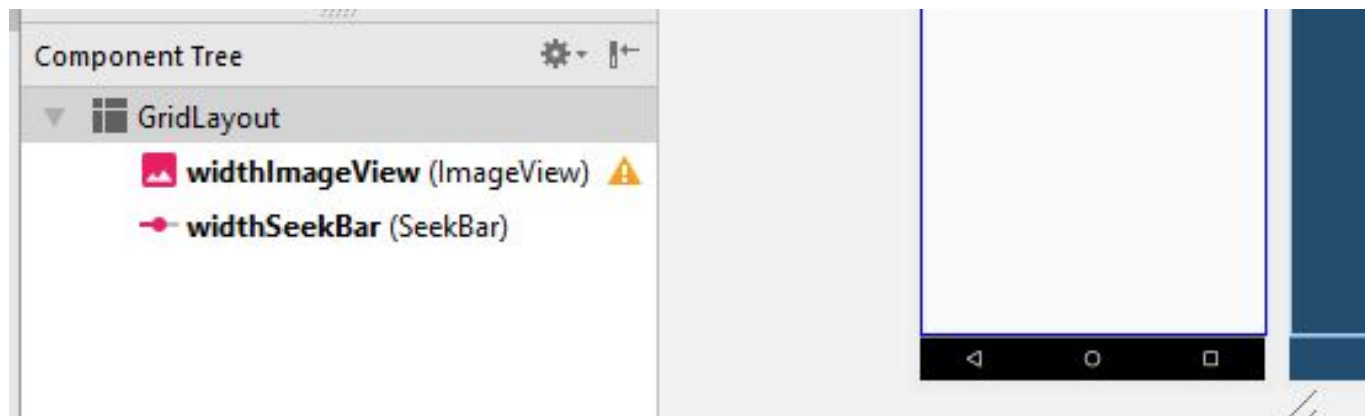
Добавить в проект java-класс `ColorDialogFragment` (пока без определения)

## Макет для выбора толщины линии

- Создать новый ресурс макета (res/layout → New → Layout Resource File) под именем `fragment_line_width`
- `GridLayout.id = lineWidthDialogGridLayout`



- Добавить компоненты `ImageView` и `SeekBar` и задать им свойство `id`



- Создать ресурс `@dimen/line_imageview_height=50dp`



# Построение графического интерфейса.

## Макет для выбора толщины линии

Компонент	Свойство	Значение
lineWidthDialogGridLayout	columnCount	1
	orientation	vertical
	useDefaultMargins	true
	padding top	@dimen/activity_vertical_margin
	padding bottom	@dimen/activity_vertical_margin
	padding left	@dimen/activity_horizontal_margin
	padding right	@dimen/activity_horizontal_margin
widthImageView	Параметры layout	
	layout:height	@dimen/line_imageview_height
	layout:gravity	fill_horizontal
	Другие свойства	
contentDescription	@string/line_imageview_description	
widthSeekBar	Параметры layout	
	layout:gravity	fill_horizontal
	Другие свойства	
max	50	

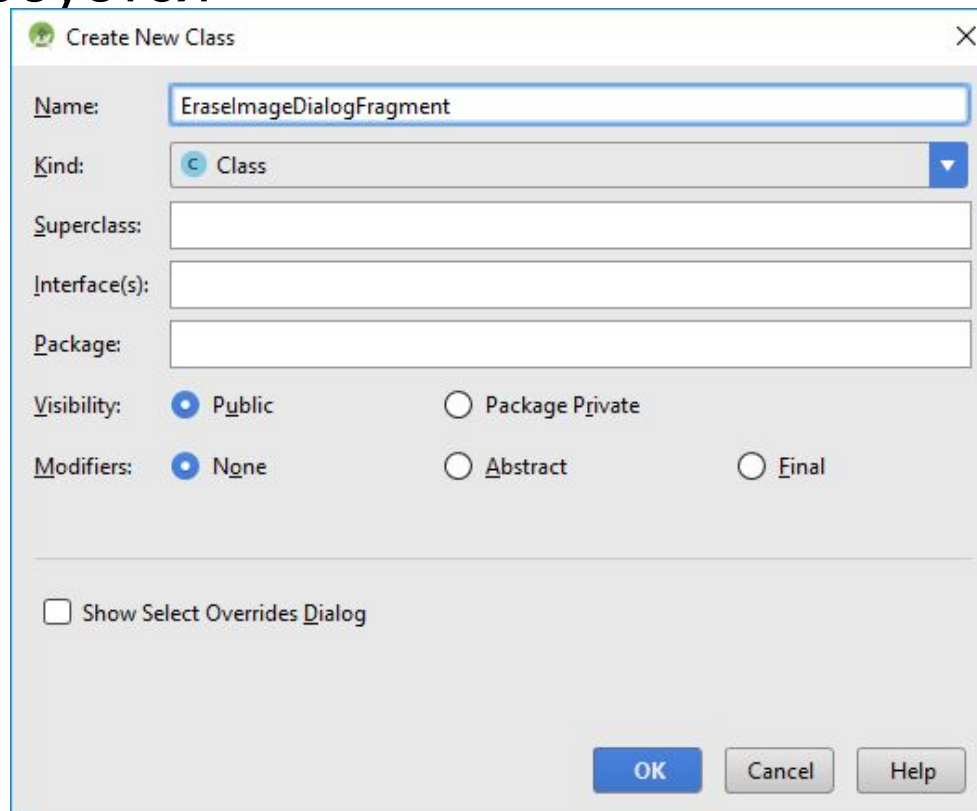
# Построение графического интерфейса.

## Макет для выбора толщины линии

- Создать новый класс LineWidthDialogFragment

## Класс для стирания рисунка

- Макет не требуется



Create New Class

Name:

Kind:  Class

Superclass:

Interface(s):

Package:

Visibility:  Public  Package Private

Modifiers:  None  Abstract  Final

Show Select Overrides Dialog

OK Cancel Help



# Описание классов

- **MainActivity** — родительская активность для фрагментов приложения
- **MainActivityFragment** — управляет DoodleView и обработкой событий акселерометра
- **DoodleView** — предоставляет функции рисования, сохранения и печати
- **ColorDialogFragment** — subclass DialogFragment, отображаемый командой меню для выбора цвета
- **LineWidthDialogFragment** — subclass DialogFragment, отображаемый командой меню для выбора толщины линии
- **EraseImageDialogFragment** — subclass DialogFragment, отображаемый командой меню для стирания текущего рисунка

# Класс MainActivity

## Дополнительные библиотеки

```
import android.content.pm.ActivityInfo;  
import android.content.res.Configuration;
```

## Дополнения в onCreate()

```
// Определение размера экрана
```

```
int screenSize =  
    getResources().getConfiguration().screenLayout &  
    Configuration.SCREENLAYOUT_SIZE_MASK;
```

```
// Альбомная ориентация только для сверхбольших планшетов
```

```
if (screenSize == Configuration.SCREENLAYOUT_SIZE_XLARGE)  
    setRequestedOrientation(  
        ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);  
else  
    setRequestedOrientation(  
        ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
```

# Класс MainActivityFragment

## Пол

```
private DoodleView doodleView; // Обработка событий касания и рисования

// Отслеживание ускорения
private float acceleration;
private float currentAcceleration;
private float lastAcceleration;

// Для предотвращения нескольких диалогов одновременно
private boolean dialogOnScreen = false;

// Используется для обнаружения встряхивания устройства
private static final int ACCELERATION_THRESHOLD = 100000;

// Используется для идентификации запросов на использование
// внешнего хранилища; необходимо для работы функции сохранения
private static final int SAVE_IMAGE_PERMISSION_REQUEST_CODE = 1;
```

# Класс MainActivityFragment

## Методы (создание главного

```
public MainActivityFragment() {...}

// Вызывается при создании представления фрагмента
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {...}
```

- фрагмент, как и активность, может размещать элементы на панели действий приложения и в меню (метод **setHasOptionsMenu()** )
- получение ссылки на **DoodleView** для дальнейшей работы (**findViewById()** )
- инициализация параметров ускорения (акселерометра) через объект **SensorManager**

# Класс MainActivityFragment

## Методы (регистрация движений)

```
// Начало прослушивания событий датчика
```

```
@Override
```

```
public void onResume() {...}
```

```
// Включение прослушивания событий акселерометра
```

```
private void enableAccelerometerListening() {...}
```

- прослушивание показаний акселерометра должно быть включено только в то время, когда MainActivityFragment **находится** на экране, поэтому необходима перезагрузка **onResume()**
- для регистрации слушателей объекта акселерометра используется объект **SensorManager**, через который приложение взаимодействует с датчиками устройства

# Класс MainActivityFragment

## Методы (прекращение регистрации

```
// Прекращение прослушивания событий акселерометра
```

```
@Override
```

```
public void onPause() {...}
```

```
// Отказ от прослушивания событий акселерометра
```

```
private void disableAccelerometerListening() {...}
```

- метод **onPause()** перегружен, чтобы отключить прослушивание событий акселерометра на то время, когда MainActivityFragment **не находится** на экране
- метод **disableAccelerometerListening()** вызывает метод **unregisterListener** класса **SensorManager** для прекращения прослушивания событий акселерометра, чтобы уменьшить нагрузку на устройство

# Класс MainActivityFragment

## Методы (обработка движений)

```
private final SensorEventListener sensorEventListener =
    new SensorEventListener() {
        // Проверка встряхивания по показаниям акселерометра
        @Override
        public void onSensorChanged(SensorEvent event) {...}

        // Обязательный метод интерфейса SensorEventListener
        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {...}
    };
```

- метод **onSensorChanged()** анонимного внутреннего класса на базе **SensorEventListener** обрабатывает события акселерометра; при перемещении устройства пользователем этот метод определяет, следует ли рассматривать данное перемещение как встряхивание
- выполняется проверка на наличие других диалоговых окон
- очень важно быстро обрабатывать события датчика или копировать данные событий, потому что массив значений датчика заново используется для каждого события.
- **onAccuracyChanged()** обязателен для интерфейса, но ничего не делает

# Класс MainActivityFragment

## Методы (создание и обработка событий)

```
Λ // Подтверждение стирания рисунка
private void confirmErase() {...}

// Отображение команд меню фрагмента
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {...}

// Обработка выбора команд меню
@Override
public boolean onOptionsItemSelected(MenuItem item) {...}
```

- метод **onOptionsItemSelected()** обрабатывает действие пользователя, выбравшего команду меню
- для получения идентификатора ресурса выбранной команды меню используется метод **MenuItem.getItemId()**



# Класс MainActivityFragment

## Методы (инициирование сохранения)

```
// При необходимости метод запрашивает разрешение  
// или сохраняет изображение, если разрешение уже имеется  
private void saveImage() {...}
```

- метод **saveImage()** вызывается методом **onOptionsItemSelected()**, когда пользователь выбирает команду Save в меню команд
- **saveImage()** перед выполнением операции проверяет, имеет ли приложение необходимое разрешение; если разрешение отсутствует, приложение запрашивает его у пользователя, прежде чем пытаться выполнять операцию.
- для работы с разрешениями используется класс **Manifest.permission** и методы **shouldShowRequestPermissionRationale()**, **requestPermissions()**

# Класс MainActivityFragment

Методы (обработка запроса на доступ к внешней памяти)

```
// Вызывается системой, когда пользователь предоставляет
// или отклоняет разрешение для сохранения изображения
@Override
public void onRequestPermissionsResult(int requestCode,
                                     String[] permissions, int[] grantResults) {...}
```

- **onRequestPermissionsResult()** получает код запроса на разрешение доступа к внешнему хранилищу (SAVE\_IMAGE\_PERMISSION\_REQUEST\_CODE) и в операторе выбора (switch) при наличии разрешения инициирует сохранение изображения
- если пользователь не дает необходимое разрешение, то при следующей попытке сохранения в диалоговое окно будет включен флажок «Никогда не спрашивать снова»; если пользователь установит этот флажок и отклонит разрешение, то при будущих попытках сохранения метод **onRequestPermissionResult()** будет вызываться с аргументом PackageManager.PERMISSION\_DENIED. Приложение должно обработать эту ситуацию и сообщить пользователю, как изменить разрешения приложения в

# Класс MainActivityFragment

## Методы (получение доступа к области

```
// Метод возвращает объект DoodleView
```

```
public DoodleView getDoodleView() { return doodleView; }
```

```
// Проверяет, отображается ли диалоговое окно
```

```
public void setDialogOnScreen(boolean visible) { dialogOnScreen = visible; }
```

- МЕТОДЫ **getDoodleView()** и **setDialogOnScreen()** ВЫЗЫВАЮТСЯ МЕТОДАМИ СУБКЛАССОВ DialogFragment (LineWidthDialogFragment, ColorDialogFragment и др.)
- **getDoodleView()** возвращает ссылку на объект DoodleView текущего фрагмента, чтобы реализация DialogFragment могла назначить цвет и толщину линии или стереть изображение
- **setDialogOnScreen()** устанавливает флаг присутствия диалогового окна на экране

# Класс DoodleView



# Класс DoodleView

Обрабатывает касания пользователя и рисует соответствующие линии

Пол

```
// пользовательское представление, на котором рисует пользователь
public class DoodleView extends View {

    // Смещение, необходимое для продолжения рисования
    private static final float TOUCH_TOLERANCE = 10;

    private Bitmap bitmap; // Область рисования для вывода или сохранения
    private Canvas bitmapCanvas; // Используется для рисования на Bitmap
    private final Paint paintScreen; // Используется для вывода Bitmap на экран
    private final Paint paintLine; // Используется для рисования линий на Bitmap
    // Данные нарисованных контуров Path и содержащихся в них точек
    private final Map<Integer, Path> pathMap = new HashMap<>();
    private final Map<Integer, Point> previousPointMap = new HashMap<>();
```

**pathMap** связывает идентификатор каждого пальца с объектом Path для рисуемых линий

**previousPointMap** хранит последнюю точку каждого пальца

# Класс DoodleView

## Конструктор

```
// Конструктор DoodleView инициализирует объект DoodleView
public DoodleView(Context context, AttributeSet attrs) {
    super(context, attrs); // Конструктору View передается контекст
    paintScreen = new Paint(); // Используется для вывода на экран

    // Исходные параметры рисуемых линий
    paintLine = new Paint();
    paintLine.setAntiAlias(true); // Сглаживание краев
    paintLine.setColor(Color.BLACK); // По умолчанию черный цвет
    paintLine.setStyle(Paint.Style.STROKE); // Сплошная линия
    paintLine.setStrokeWidth(5); // Толщина линии по умолчанию
    paintLine.setStrokeCap(Paint.Cap.ROUND); // Закругленные концы
}
```



# Класс DoodleView

## Методы (создание области рисования)

```
// Создание объектов Bitmap и Canvas на основании размеров View
@Override
public void onSizeChanged(int w, int h, int oldW, int oldH) {
    bitmap = Bitmap.createBitmap(getWidth(), getHeight(),
        Bitmap.Config.ARGB_8888);
    bitmapCanvas = new Canvas(bitmap);
    bitmap.eraseColor(Color.WHITE); // Bitmap стирается белым цветом
}
```

- размер DoodleView определяется только после того, как представление будет заполнено и добавлено в иерархию представлений MainActivity; следовательно, определить размер объекта Bitmap, используемого при рисовании, в методе onCreate() не удастся
- **onSizeChanged()** вызывается при изменении размера DoodleView, например при добавлении в иерархию представлений активности или при повороте устройства

# Класс DoodleView

## Методы (очистка рисунка)

```
public void clear() {  
    pathMap.clear(); // Удалить все контуры  
    previousPointMap.clear(); // Удалить все предыдущие точки  
    bitmap.eraseColor(Color.WHITE); // Очистка изображения  
    invalidate(); // Перерисовать изображение  
}
```

- очищает коллекции pathMap и previousPointMap
- стирает Bitmap, заполняя все пиксели белым цветом
- вызывает унаследованный от View метод invalidate, чтобы сообщить о необходимости перерисовки View



# Класс DoodleView

## Методы (получение и установка свойств)

*// Назначение цвета рисуемой линии*

```
public void setDrawingColor(int color) { paintLine.setColor(color); }
```

*// Получение цвета рисуемой линии*

```
public int getDrawingColor() { return paintLine.getColor(); }
```

*// Назначение толщины рисуемой линии*

```
public void setLineWidth(int width) { paintLine.setStrokeWidth(width); }
```

*// Получение толщины рисуемой линии*

```
public int getLineWidth() { return (int) paintLine.setStrokeWidth(); }
```

- **getDrawingColor()** используется в ColorDialogFragment
- **getLineWidth()** используется в LineWidthDialogFragment

# Класс DoodleView

## Методы (перерисовка)

```
// Перерисовка при обновлении DoodleView на экране
@Override
protected void onDraw(Canvas canvas) {
    // Перерисовка фона
    canvas.drawBitmap(bitmap, left: 0, top: 0, paintScreen);

    // Для каждой выводимой линии
    for (Integer key : pathMap.keySet())
        canvas.drawPath(pathMap.get(key), paintLine); // Рисование линии
}
```

- **onDraw()** вызывается автоматически, когда представлению требуется перерисовка
- **drawBitmap()** отображает объект `bitmap` на холсте
- в цикле для каждого целочисленного ключа в `pathMap` соответствующий объект `Path` передается методу **drawPath()** объекта `canvas` для прорисовки с использованием объекта `paintLine`, определяющего толщину и цвет линии

# Класс DoodleView

## Методы (регистрация

```
// Обработка события касания
@Override
public boolean onTouchEvent(MotionEvent event) {
    int action = event.getActionMasked(); // Тип события
    int actionIndex = event.getActionIndex(); // Указатель (палец)

    // Что происходит: начало касания, конец, перемещение?
    if (action == MotionEvent.ACTION_DOWN ||
        action == MotionEvent.ACTION_POINTER_DOWN) {
        touchStarted(event.getX(actionIndex), event.getY(actionIndex),
            event.getPointerId(actionIndex));
    } else if (action == MotionEvent.ACTION_UP ||
        action == MotionEvent.ACTION_POINTER_UP) {
        touchEnded(event.getPointerId(actionIndex));
    } else {
        touchMoved(event);
    }

    invalidate(); // Перерисовка
    return true;
}
```

- **onTouchEvent()** вызывается при событии касания
- возможны многоточечные касания
- каждому указателю присваивается идентификатор, получаемый методом **getPointerId()** и используемый для идентификации объектов Path, рисуемых в данный момент

# Класс DoodleView

## Методы (новая линия при первом касании)

```
// Вызывается при касании экрана
private void touchStarted(float x, float y, int lineID) {
    Path path; // Для хранения контура с заданным идентификатором
    Point point; // Для хранения последней точки в контуре

    // Если для lineID уже существует объект Path
    if (pathMap.containsKey(lineID)) {
        path = pathMap.get(lineID); // Получение Path
        path.reset(); // Очистка Path с началом нового касания
        point = previousPointMap.get(lineID); // Последняя точка Path
    } else {
        path = new Path();
        pathMap.put(lineID, path); // Добавление Path в Map
        point = new Point(); // Создание нового объекта Point
        previousPointMap.put(lineID, point); // Добавление Point в Map
    }

    // Переход к координатам касания
    path.moveTo(x, y);
    point.x = (int) x;
    point.y = (int) y;
}
```

- **touchStarted()** вызывается при первом касании
- сохраняются исходные координаты касания
- при многоточечном касании метод срабатывает для каждого указателя



# Класс DoodleView

## Методы (рисование при движении указателя по экрану)

```
// Вызывается при перемещении пальца по экрану
private void touchMoved(MotionEvent event) {
    // Для каждого указателя (пальца) в объекте MotionEvent
    for (int i = 0; i < event.getPointerCount(); i++) {
        // Получить идентификатор и индекс указателя
        int pointerID = event.getPointerId(i);
        int pointerIndex = event.findPointerIndex(pointerID);

        // Если существует объект Path, связанный с указателем
        if (pathMap.containsKey(pointerID)) {
            // Получить новые координаты для указателя
            float newX = event.getX(pointerIndex);
            float newY = event.getY(pointerIndex);

            // Получить объект Path и предыдущий объект Point,
            // связанный с указателем
            Path path = pathMap.get(pointerID);
            Point point = previousPointMap.get(pointerID);
```

- **touchMoved()** вызывается при движении указателя по экрану
- в цикле для каждого указателя строится новый фрагмент линии между предыдущей и новой точками

# Класс DoodleView

Методы (рисование при движении указателя по экрану)

```
// Получить объект Path и предыдущий объект Point,
// связанный с указателем
Path path = pathMap.get(pointerID);
Point point = previousPointMap.get(pointerID);

// Вычислить величину смещения от последнего обновления
float deltaX = Math.abs(newX - point.x);
float deltaY = Math.abs(newY - point.y);

// Если расстояние достаточно велико
if (deltaX >= TOUCH_TOLERANCE || deltaY >= TOUCH_TOLERANCE) {
    // Расширение контура до новой точки
    path.quadTo(point.x, point.y, x2: (newX + point.x) / 2,
               y2: (newY + point.y) / 2);

    // Сохранение новых координат
    point.x = (int) newX;
    point.y = (int) newY;
}
}
```

- линия строится только при достаточно большом смещении ( $\geq \text{TOUCH\_TOLERANCE}$ ), чтобы не реагировать на дрожание указателя

# Класс DoodleView

## Методы (отведение указателя от

```
// Вызывается при завершении касания  
private void touchEnded(int lineID) {  
    Path path = pathMap.get(lineID); // Получение объекта Path  
    bitmapCanvas.drawPath(path, paintLine); // Рисование на bitmapCanvas  
    path.reset(); // Сброс объекта Path  
}
```

- **touchEnded()** вызывается, когда пользователь отводит указатель от экрана
- - в методе **touchMoved()** рисование происходит методом **path.quadTo()**; - каждая новая кривая должна начинаться в новой точке, поэтому при завершении касания необходимо сбросить path  
(path.reset());  
- после сброса линия исчезает с экрана;  
- для её сохранения рисуем путь на холсте:  
**bitmapCanvas.drawPath()**



# Класс DoodleView

## Методы (сохранение изображения в

```
// Сохранение текущего изображения в галерею
public void saveImage() {
    // Имя состоит из префикса "Doodlz" и текущего времени
    final String name = "Doodlz" + System.currentTimeMillis() + ".jpg";

    // Сохранение изображения в галерею устройства
    String location = MediaStore.Images.Media.insertImage(
        getContext().getContentResolver(), bitmap, name,
        "Doodlz Drawing");
}
```

- **saveImage()** вызывается из главного фрагмента
- имя файла генерируется автоматически (переменная name)
- содержимое bitmap сохраняется в приложении Photos (в галерею устройства)

# Класс DoodleView

## Методы (сохранение изображения в

```
if (location != null) {  
    // Вывод сообщения об успешном сохранении  
    Toast message = Toast.makeText(getContext(),  
        R.string.message_saved,  
        Toast.LENGTH_SHORT);  
    message.setGravity(Gravity.CENTER, message.getXOffset() / 2,  
        message.getYOffset() / 2);  
    message.show();  
} else {  
    // Вывод сообщения об ошибке сохранения  
    Toast message = Toast.makeText(getContext(),  
        R.string.message_error_saving, Toast.LENGTH_SHORT);  
    message.setGravity(Gravity.CENTER, message.getXOffset() / 2,  
        message.getYOffset() / 2);  
    message.show();  
}  
}
```

- location содержит URL созданного изображения или null
- после попытки сохранения выводится диалоговое окно с сообщением об успехе или неуспехе сохранения

# Класс DoodleView

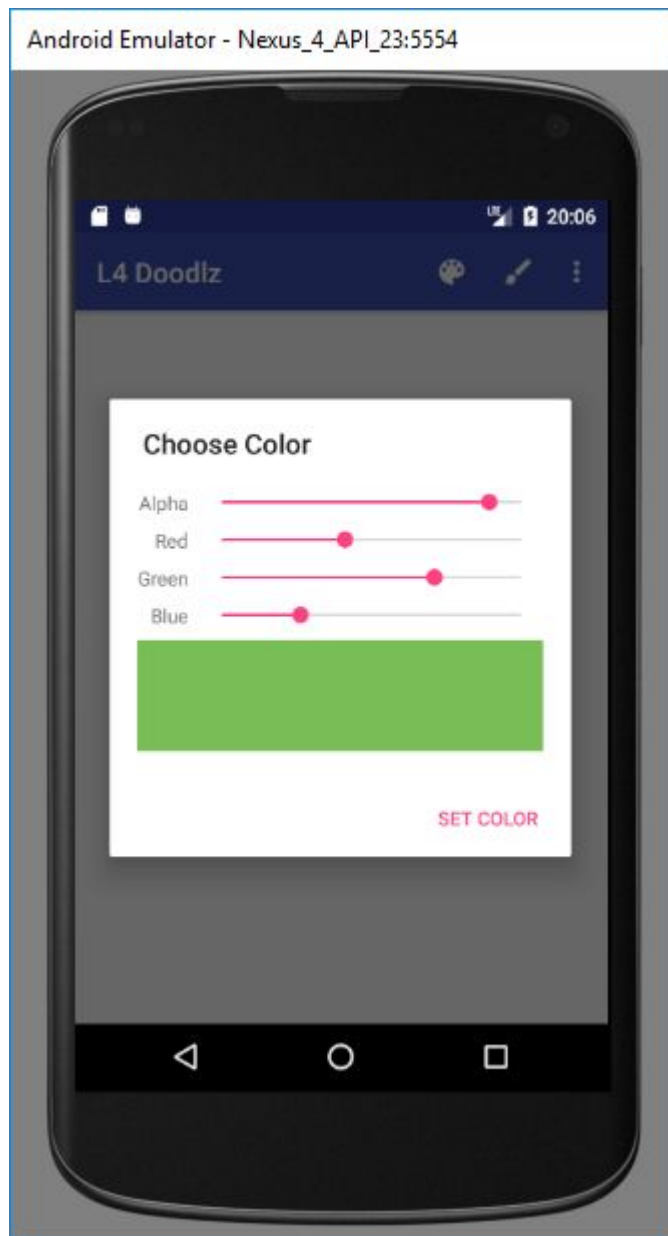
## Методы (печать изображения)

```
// Печать текущего изображения
public void printImage() {
    if (PrintHelper.systemSupportsPrint()) {
        // Использование класса PrintHelper для печати
        PrintHelper printHelper = new PrintHelper(getContext());

        // Изображение масштабируется и выводится на печать
        printHelper.setScaleMode(PrintHelper.SCALE_MODE_FIT);
        printHelper.printBitmap("Doodlz Image", bitmap);
    } else {
        // Вывод сообщения о том, что система не поддерживает печать
        Toast message = Toast.makeText(getContext(),
            R.string.message_error_printing, Toast.LENGTH_SHORT);
        message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
            message.getYOffset() / 2);
        message.show();
    }
}
```

- **printImage()** вызывается из главного фрагмента и использует класс PrintHelper из библиотеки поддержки
- если система поддерживает возможность печати, то диалог печати и процедуру сохранения полностью реализует PrintHelper

# Класс ColorDialogFragment



# Класс ColorDialogFragment

Расширяет DialogFragment для создания окна AlertDialog, в котором определяется цвет линии

## Импортируемые пакеты и

```
package com.example.someone.14doodlz;

// ColorDialogFragment.java
// Используется для выбора цвета линии в DoodleView

import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.graphics.Color;
import android.content.Context;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.view.View;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
```



# Класс ColorDialogFragment

## Пол

```
; // Класс диалогового окна выбора цвета
```

```
public class ColorDialogFragment extends DialogFragment {  
    private SeekBar alphaSeekBar;  
    private SeekBar redSeekBar;  
    private SeekBar greenSeekBar;  
    private SeekBar blueSeekBar;  
    private View colorView;  
    private int color;
```

- **alphaSeekBar, redSeekBar, greenSeekBar, blueSeekBar** предназначены для доступа к соответствующим регуляторам компонент цвета
- **colorView** представляет компонент диалогового окна, показывающий образец цвета
- **color** хранит выбранный цвет (32 разряда)

# Класс ColorDialogFragment

## Методы (инициализация)

```
// Создание и возвращение объекта AlertDialog
@Override
public Dialog onCreateDialog(Bundle bundle) {
    // Создание диалогового окна
    AlertDialog.Builder builder =
        new AlertDialog.Builder(getActivity());
    View colorDialogView = getActivity().getLayoutInflater().inflate(
        R.layout.fragment_color, null);
    builder.setView(colorDialogView); // Добавление GUI в диалоговое окно

    // Назначение сообщения AlertDialog
    builder.setTitle(R.string.title_color_dialog);

    // Получение значений SeekBar
    alphaSeekBar = (SeekBar) colorDialogView.findViewById(
        R.id.alphaSeekBar);
    redSeekBar = (SeekBar) colorDialogView.findViewById(
        R.id.redSeekBar);
    greenSeekBar = (SeekBar) colorDialogView.findViewById(
        R.id.greenSeekBar);
    blueSeekBar = (SeekBar) colorDialogView.findViewById(
        R.id.blueSeekBar);
    colorView = colorDialogView.findViewById(R.id.colorView);
}
```



# Класс ColorDialogFragment

## Методы (инициализация)

```
// Регистрация слушателей событий SeekBar
alphaSeekBar.setOnSeekBarChangeListener(colorChangeListener);
redSeekBar.setOnSeekBarChangeListener(colorChangeListener);
greenSeekBar.setOnSeekBarChangeListener(colorChangeListener);
blueSeekBar.setOnSeekBarChangeListener(colorChangeListener);

// Использование текущего цвета линии для инициализации
final DoodleView doodleView = getDoodleFragment().getDoodleView();
color = doodleView.getDrawingColor();
alphaSeekBar.setProgress(Color.alpha(color));
redSeekBar.setProgress(Color.red(color));
greenSeekBar.setProgress(Color.green(color));
blueSeekBar.setProgress(Color.blue(color));
```

- для всех интерактивных элементов цветовых компонент назначается одинаковый обработчик
- положение «ползунков» выставляется на основе текущего цвета линии в DoodleView

# Класс ColorDialogFragment

## Методы (инициализация)

```
// Добавление кнопки назначения цвета
builder.setPositiveButton(R.string.button_set_color,
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            doodleView.setDrawingColor(color);
        }
    }
);

return builder.create(); // Возвращение диалогового окна
}
```

- **setPositiveButton()** создаёт в диалоге кнопку положительного ответа и назначает обработчик её нажатия
- выбранный цвет передаётся в DoodleView

# Класс ColorDialogFragment

## Методы (получение ссылки на

```
// Получение ссылки на MainActivityFragment  
private MainActivityFragment getDoodleFragment() {  
    return (MainActivityFragment) getSupportFragmentManager().findFragmentById(  
        R.id.doodleFragment);  
}
```

- **getDoodleFragment()** использует объект `FragmentManager` для получения ссылки на `MainActivityFragment`
- используется в методах `onCreateDialog()`, `onAttach()`, `onDetach()`

# Класс ColorDialogFragment

Методы (действия при добавлении и удалении диалога)

```
// Сообщает DoodleFragment, что диалоговое окно находится на экране  
@Override
```

```
public void onAttach(Context context) {  
    super.onAttach(context);  
    MainActivityFragment fragment = getDoodleFragment();  
    if (fragment != null)  
        fragment.setDialogOnScreen(true);  
}
```

```
// Сообщает MainActivityFragment, что диалоговое окно не отображается
```

```
@Override  
public void onDetach() {  
    super.onDetach();  
    MainActivityFragment fragment = getDoodleFragment();  
    if (fragment != null)  
        fragment.setDialogOnScreen(false);  
}
```

- методы вызываются при добавлении/удалении диалога в родительской активности
- setDialogOnScreen() устанавливает и сбрасывает флаг отображения диалогового окна

# Класс ColorDialogFragment

## Методы (обработка событий компонентов SeekBar)

```
// OnSeekBarChangeListener для компонентов SeekBar в диалоговом окне
private final OnSeekBarChangeListener colorChangeListener =
    new OnSeekBarChangeListener() {
        // Отображение обновленного цвета
        @Override
        public void onProgressChanged(SeekBar seekBar, int progress,
            boolean fromUser) {

            if (fromUser) // Изменено пользователем (не программой)
                color = Color.argb(alphaSeekBar.getProgress(),
                    redSeekBar.getProgress(), greenSeekBar.getProgress(),
                    blueSeekBar.getProgress());
            colorView.setBackgroundColor(color);
        }
    }
```

- используется анонимный внутренний класс **OnSeekBarChangeListener**
- проверка fromUser необходима, чтобы исключить срабатывание метода при программной установке «ползунка»



# Класс ColorDialogFragment

## Методы (обработка событий компонентов)

```
@Override
public void onStartTrackingTouch(SeekBar seekBar) {}
// ОБЯЗАТЕЛЬНЫЙ МЕТОД

@Override
public void onStopTrackingTouch(SeekBar seekBar) {}
// ОБЯЗАТЕЛЬНЫЙ МЕТОД
};
```

- методы `onStartTrackingTouch()` и `onStopTrackingTouch()` объявлены в интерфейсе `OnSeekBarChangeListener` и должны быть определены
- в нашей задаче отслеживать начало и окончание передвижения «ползунка» необходимости нет, поэтому методы пустые

# Класс LineWidthDialogFragment





# Класс LineWidthDialogFragment

Расширяет DialogFragment для создания окна AlertDialog, в котором определяется толщина линии

Импортируемые пакеты и классы. Поле

```
import android.content.Context;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.view.View;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;

// Класс диалогового окна выбора цвета
public class LineWidthDialogFragment extends DialogFragment {
    private ImageView widthImageView;
```

поле **widthImageView** предназначено для доступа к образцу линии с текущей толщиной

# Класс LineWidthDialogFragment

## Методы (инициализация диалога)

```
// Создает и возвращает AlertDialog
@Override
public Dialog onCreateDialog(Bundle bundle) {
    // Создание диалогового окна
    AlertDialog.Builder builder =
        new AlertDialog.Builder(getActivity());
    View lineWidthDialogView =
        getActivity().getLayoutInflater().inflate(
            R.layout.fragment_line_width, null);
    builder.setView(lineWidthDialogView); // Добавление GUI

    // Назначение сообщения AlertDialog
    builder.setTitle(R.string.title_line_width_dialog);

    // Получение ImageView
    widthImageView = (ImageView) lineWidthDialogView.findViewById(
        R.id.widthImageView);
}
```

- макет фрагмента `fragment_line_width` загружается из ресурсов
- `widthImageView` получает ссылку на элемент фрагмента, изображающий линию

# Класс LineWidthDialogFragment

## Методы (инициализация)

```
// Настройка widthSeekBar
final DoodleView doodleView = getDoodleFragment().getDoodleView();
final SeekBar widthSeekBar = (SeekBar)
    lineWidthDialogView.findViewById(R.id.widthSeekBar);
widthSeekBar.setOnSeekBarChangeListener(lineWidthChanged);
widthSeekBar.setProgress(doodleView.getLineWidth());

// Добавление кнопки Set Line Width
builder.setPositiveButton(R.string.button_set_line_width,
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            doodleView.setLineWidth(widthSeekBar.getProgress());
        }
    });
return builder.create(); // Возвращение диалогового окна
}
```

- положение «ползунка» выставляется на основе текущей толщины линии в DoodleView
- **setPositiveButton()** создаёт в диалоге кнопку положительного ответа и назначает обработчик её нажатия
- выбранная толщина линии передаётся в DoodleView

# Класс LineWidthDialogFragment

## Методы (получение ссылки на DoodleFragment)

```
// Возвращает ссылку на MainActivityFragment  
private MainActivityFragment getDoodleFragment() {...}  
  
// Сообщает MainActivityFragment, что диалоговое окно находится на экране  
@Override  
public void onAttach(Context context) {...}  
  
// Сообщает MainActivityFragment, что окно не отображается  
@Override  
public void onDetach() {...}
```

- методы **getDoodleFragment()**, **onAttach()**, **onDetach()** идентичны методам класса **ColorDialogFragment** с такими же названиями



# Класс LineWidthDialogFragment

## Методы (обработка событий компонента

### SeekBar)

```

// OnSeekBarChangeListener для SeekBar в диалоговом окне толщины линии
private final OnSeekBarChangeListener lineWidthChanged =
    new OnSeekBarChangeListener() {
        final Bitmap bitmap = Bitmap.createBitmap(
            400, 100, Bitmap.Config.ARGB_8888);
        final Canvas canvas = new Canvas(bitmap); // Рисует на Bitmap

        @Override
        public void onProgressChanged(SeekBar seekBar, int progress,
            boolean fromUser) {

            // Настройка объекта Paint для текущего значения SeekBar
            Paint p = new Paint();
            p.setColor(
                getDoodleFragment().getDoodleView().getDrawingColor());
            p.setStrokeCap(Paint.Cap.ROUND);
            p.setStrokeWidth(progress);

            // Стирание объекта Bitmap и перерисовка линии
            bitmap.eraseColor(
                getResources().getColor(android.R.color.transparent,
                    getContext().getTheme()));
            canvas.drawLine(30, 50, 370, 50, p);
            widthImageView.setImageBitmap(bitmap);
        }
    }

```

образец линии отображается  
на объекте **bitmap**

используется анонимный  
класс

# Класс LineWidthDialogFragment

## Методы (обработка событий компонента

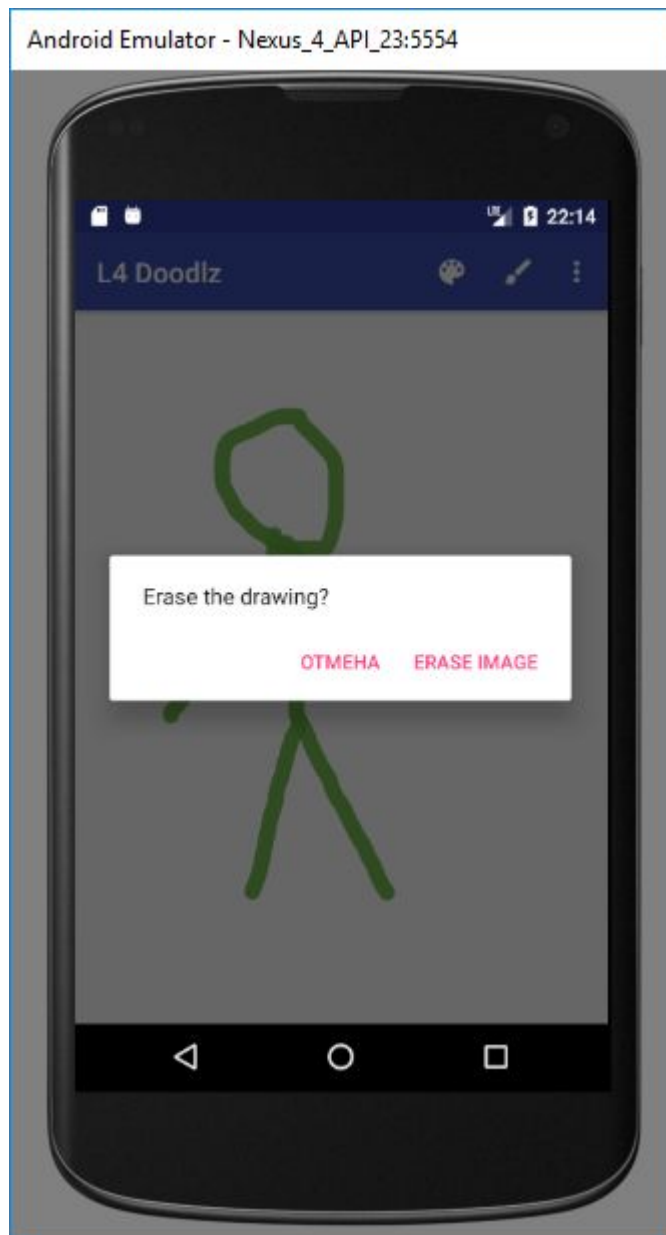
```
@Override
public void onStartTrackingTouch(SeekBar seekBar) {}
// ОБЯЗАТЕЛЬНЫЙ МЕТОД

@Override
public void onStopTrackingTouch(SeekBar seekBar) {}
// ОБЯЗАТЕЛЬНЫЙ МЕТОД
};
```

- методы `onStartTrackingTouch()` и `onStopTrackingTouch()` объявлены в интерфейсе `OnSeekBarChangeListener` и должны быть определены
- в нашей задаче отслеживать начало и окончание передвижения «ползунка» необходимости нет, поэтому методы пустые



# Класс EraseImageDialogFragment



# Класс EraseImageDialogFragment

Расширяет DialogFragment для создания окна AlertDialog, в котором пользователь подтверждает стирание всего изображения

Импортируемые пакеты и

```
package com.example.someone.14doodlz;  
// фрагмент для стирания изображения  
  
import android.content.Context;  
import android.app.AlertDialog;  
import android.app.Dialog;  
import android.support.v4.app.DialogFragment;  
import android.content.DialogInterface;  
import android.os.Bundle;
```

- поля отсутствуют

# Класс EraseImageDialogFragment

## Методы (инициализация диалога)

```
// Создание и возвращение объекта AlertDialog
@Override
public Dialog onCreateDialog(Bundle bundle) {
    AlertDialog.Builder builder =
        new AlertDialog.Builder(getActivity());
    // Назначение сообщения AlertDialog
    builder.setMessage(R.string.message_erase);
    // Добавление кнопки стирания
    builder.setPositiveButton(R.string.button_erase,
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                getDoodleFragment().getDoodleView().clear(); // Очистка
            }
        }
    );
    // Добавление кнопки стирания
    builder.setNegativeButton(android.R.string.cancel, null);
    return builder.create(); // Возвращает диалоговое окно
}
```

- создание диалога с двумя кнопками
- при нажатии кнопки положительного ответа («Стереть изображение») вызывается метод `clear()` класса `DoodleView`

# Класс EraseImageDialogFragment

## Методы (получение ссылки на DoodleFragment)

```
// Возвращает ссылку на MainActivityFragment  
private MainActivityFragment getDoodleFragment() {...}  
  
// Сообщает MainActivityFragment, что диалоговое окно находится на экране  
@Override  
public void onAttach(Context context) {...}  
  
// Сообщает MainActivityFragment, что окно не отображается  
@Override  
public void onDetach() {...}
```

- методы **getDoodleFragment()**, **onAttach()**, **onDetach()** идентичны методам класса **ColorDialogFragment** с такими же названиями

# Интернационализация

L4Doodlz - [D:\ALEK\PAPERS\REITING\PVS\BookDeitel\androidfp3\_examples\L4Doodlz] - ...\app\Translations Editor - Android Studio 3.0.1

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

L4Doodlz app Translations Editor

Android DoodleView.java fragment\_color.xml values\strings.xml Translations Editor

Project Structure Captures Build Variants Favorites

app

- manifests
- java
- res
  - drawable
  - layout
  - menu
  - mipmap
  - values
    - colors.xml
    - dimens.xml
    - strings.xml (2)
      - strings.xml
      - strings.xml (ru)
    - styles.xml
- Gradle Scripts
  - build.gradle (Project: L4Doodlz)
  - build.gradle (Module: app)
  - gradle-wrapper.properties (Gradle)
  - proguard-rules.pro (ProGuard)
  - gradle.properties (Project Properties)
  - settings.gradle (Project Settings)
  - local.properties (SDK Location)

Translations Editor

Show All Keys Show All Locales ? Order a translation...

Key	Resource Folder	Untransla...	Default Value	Russian (ru)
app_name	app\src\main\res	<input type="checkbox"/>	L4 Doodlz	L4 Doodlz
action_settings	app\src\main\res	<input type="checkbox"/>	Settings	Настройки
button_erase	app\src\main\res	<input type="checkbox"/>	Erase Image	Стереть
button_set_color	app\src\main\res	<input type="checkbox"/>	Set Color	Задать цвет
button_set_line_width	app\src\main\res	<input type="checkbox"/>	Set Line Width	
line_imageview_description	app\src\main\res	<input type="checkbox"/>	This displays the line thickness	
label_alpha	app\src\main\res	<input type="checkbox"/>	Alpha	
label_red	app\src\main\res	<input type="checkbox"/>	Red	
label_green	app\src\main\res	<input type="checkbox"/>	Green	
label_blue	app\src\main\res	<input type="checkbox"/>	Blue	
menuitem_delete	app\src\main\res	<input type="checkbox"/>	Erase Drawing	
menuitem_color	app\src\main\res	<input type="checkbox"/>	Color	
menuitem_line_width	app\src\main\res	<input type="checkbox"/>	Line Width	
menuitem_save	app\src\main\res	<input type="checkbox"/>	Save	
menuitem_print	app\src\main\res	<input type="checkbox"/>	Print	
message_erase	app\src\main\res	<input type="checkbox"/>	Erase the drawing?	
message_error_saving	app\src\main\res	<input type="checkbox"/>	There was an error saving the image	
message_saved	app\src\main\res	<input type="checkbox"/>	Your saved painting can be viewed in t	
message_error_printing	app\src\main\res	<input type="checkbox"/>	Your device does not support printing	

Key: button\_set\_line\_width

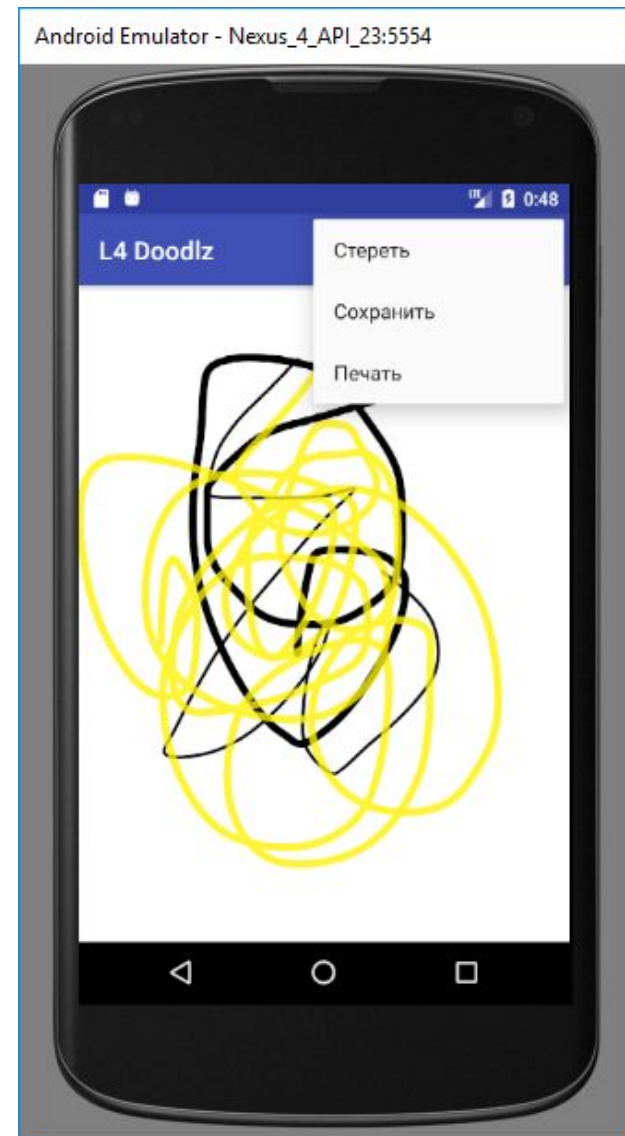
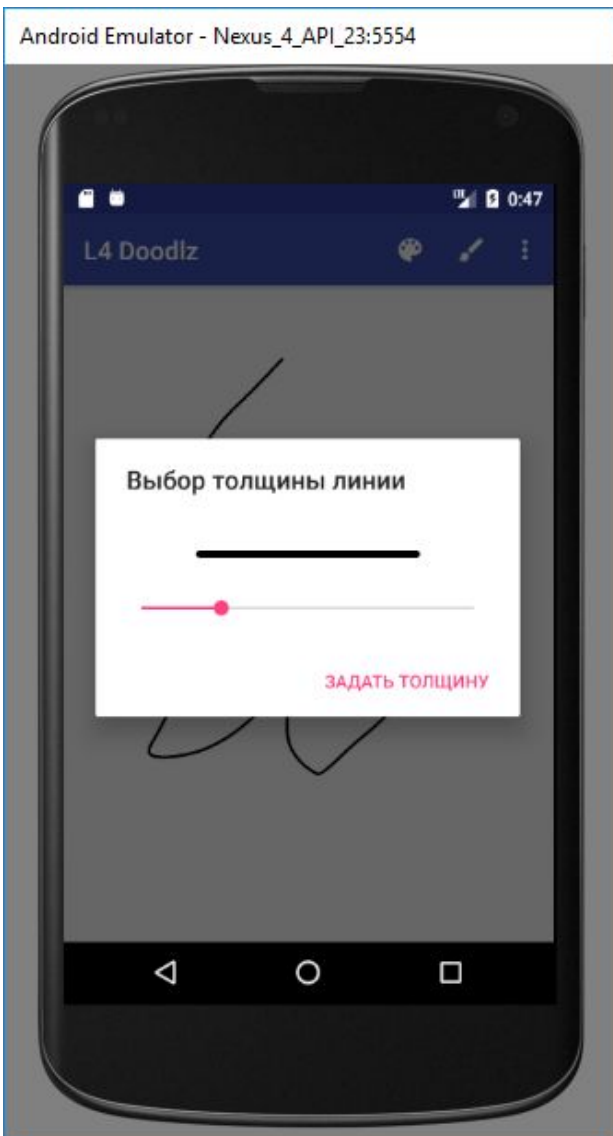
Default Value: Set Line Width

Translation:

Gradle Preview Device File Explorer

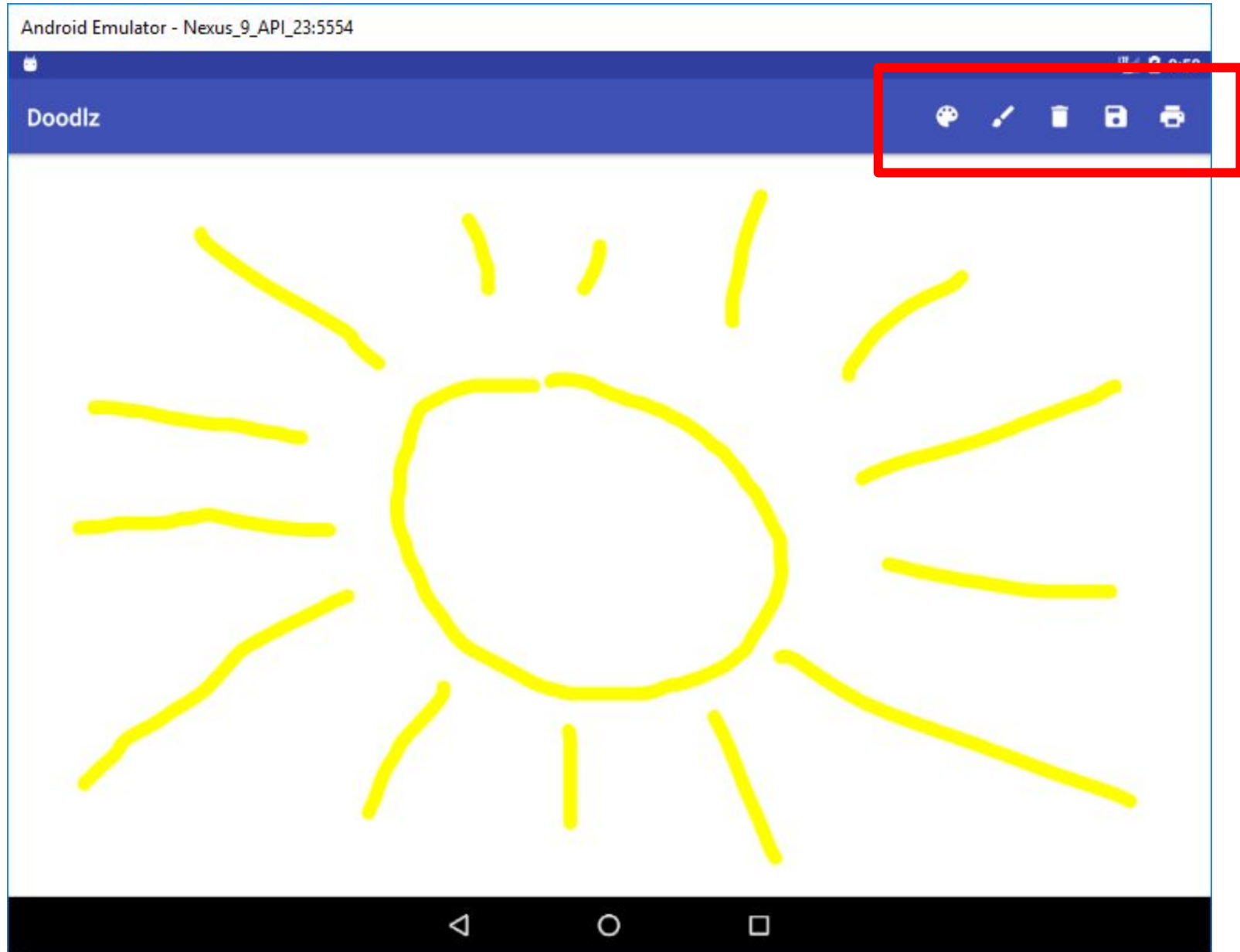


# Интернационализация





# Проверка на планшете



# Приложение

- [исходный текст класса MainActivity](#)
- [исходный текст класса MainActivityFragment](#)
- [исходный текст класса DoodleView](#)
- [исходный текст класса ColorDialogFragment](#)
- [исходный текст класса  
LineWidthDialogFragment](#)
- [исходный текст класса  
EraseImageDialogFragment](#)