Второй курс, осенний семестр 2016/17
Конспект семинара по Android на тему User Interface

Собрано 10 октября 2016 г. в 14:16

Содержание

Пункт #1
View и ViewGroup

1.1. Пользовательский интерфейс

В приложении под Android все элементы пользовательского интерфейса (англ. User Interface, UI) строятся на основе View и ViewGroup объектов. View – это объект, который рисует что-то, с чем пользователь может взаимодействовать, на экране. ViewGroup – это объект, который содержит внутри себя другие View (и ViewGroup) объекты, и таким образом определяет, как будет выглядеть интерфейс.

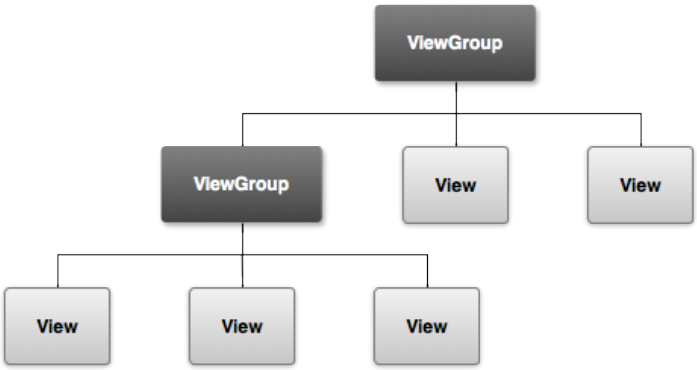


Рис. 1: Пример иерархии, которая определяет UI

Пункт #2
Макеты (Layouts)

2.1. Макет

Макет (англ. Layout) определяет то, как будет выглядеть структура UI. Создать layout можно двумя способами:

- **Описать элементы UI в XML-файле.** Андроид предоставляет прямолинейный XML синтаксис, который соответствует классам и подклассам View .
- **Создать элементы layout во время выполнения.** Приложение может создавать объекты View и ViewGroup (и изменять их свойства) программно.

Фреймворк Андроида позволяет гибко использовать только один или сразу оба из этих способов, для создания и управления UI приложения. Преимуществом создания UI в XML является тот факт, что внешний вид приложения отделяется от кода, который управляет логикой.

2.2. Стандартные макеты

Каждый подкласс класса ViewGroup предоставляет уникальный способ для отображения View , которые он содержит внутри.

Замечание: Хотя и можно помещать один layout в другой, чтобы добиться желаемого внешнего вида, не стоит этим злоупотреблять. Следует стараться сделать иерархию как можно меньше. Layout рисуется быстрее, если у него мало вложенных layout (широкая иерархия лучше, чем глубокая).

Вот некоторые наиболее часто используемые виды layout, которые предоставляются платформой Android:

- **Linear Layout**. Layout, который распологает свои дочернии View в один горизонтальный или вертикальный ряд. Он создает scrollbar (прокрутку), если размер содержимого превосходит размеры экрана.
- **Relative Layout**. Позволяет определить положение дочерних объектов относительно друг друга (например, расположить объект A слева от объекта B) или родителей (например, выровнить по верхней части родителя).
- **Web View**. Отображает web-страницы.

Рис. 2: Linear Layout

Рис. 3: Relative Layout

Рис. 4: Web View

2.3. Создание макета через Adapter

Когда содержимое layout динамически изменяется или заранее не известно, можно использовать layout, который является подклассом AdapterView , чтобы заполнять layout различными View во время выполнения. Подклассы класса AdapterView используют Adapter , чтобы привязать данные к layout. Adapter выступает в роли посредника между источником данных и AdapterView – Adapter получает данные (из таких источников как массив или база данных) и конвертирует каждую запись во View , который может быть добавлен в layout AdapterView .

Некоторые типичные layout, которые создаются с помощью Adapter :

- **List View**. Отображает прокручиваемую колонку.
- **Grid View**. Отображает прокручиваемую таблицу.

Рис. 5: List View

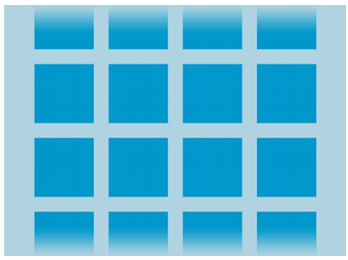


Рис. 6: Grid View

Пункт #3
Элементы управления вводом (Input Controls)

3.1. Элементы управления вводом

Элементы управления вводом это интерактивные компоненты в UI приложения. Андроид предоставляет множество разнообразных контроллеров, которые можно использовать в UI: кнопки, текстовые поля, флажки, кнопки масштабирования, кнопки переключения, и много других.

3.2. Button, ImageButton

Кнопка, состоящая из текста или иконки (или и того и другого), которые сообщают какое действие произойдет, когда пользователь коснется кнопки.

XML элемент, который создает кнопку с текстом, используя класс Button :

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

3.2.1. Реагирование на событие нажатия

Когда пользователь нажимает кнопку, объект Button получает событие нажатия.

Чтобы задать обработчик события нажатия для кнопки, нужно добавить в XML layout к элементу *<Button>* атрибут *android:onClick*. Значением этого атрибута должно быть имя метода, который будет вызываться в ответ на событие нажатия. Activity , к которой привязан layout, должна затем реализовать соответствующий метод.

В качестве примера, layout кнопки, использующий *android:onClick*:

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

Внутри Activity , к которой привязан этот layout, следующий метод обрабатывает событие нажатия:

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

Метод, объявленный в атрибуте *android:onClick*, должен иметь точно такую сигнатуру, как показано выше. А именно, метод должен:

- Быть *public*.
- Возвращать *void*
- Иметь единственный параметр типа View (это будет View , который был нажат).

3.2.2. Использование OnClickListener

Также можно объявить обработчик события нажатия программно, а не через XML layout. Это может быть необходимо, если кнопка создается во время исполнения.

Чтобы объявить обработчик программно, нужно создать объект класса View.OnClickListener и привязать его к кнопке, вызвав *setOnClickListener(View.OnClickListener)*. Например:

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

Если предыдущий листинг не понятен, то нужно прочитать про анонимные вложенные классы.

3.3. EditText

Текстовое поле позволяет пользователю вводить текст. Поле может быть однострочным, а может быть многострочным.

С помощью атрибута *android:inputType* можно указать тип клавиатуры, которая должна использоваться для объекта EditText . Например, если нужно, чтобы пользователь ввел адрес электронной почты, нужно использовать тип *textEmailAddress*:

```
<EditText
    android:id="@+id/email_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/email_hint"
    android:inputType="textEmailAddress" />
```

3.3.1. Указание действия на клавиатуре

В дополнении к изменению типа клавиатуры, Андроид позволяет указать действие, которое нужно сделать, когда пользователь завершил ввод. Выбор действия влияет на кнопку, которая появится на месте кнопки перевода строки и на действие, которое нужно выполнить (например, «Найти» или «Отправить»).

Действие можно указать, задав атрибут *android:imeOptions*. Например, вот как можно указать действие "Отправить":

```
<EditText
    android:id="@+id/search"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/search_hint"
    android:inputType="text"
    android:imeOptions="actionSend" />
```

3.3.2. Реагирование на событие кнопки

Если через атрибут *android:imeOptions* указано действие (такое как "actionSend"), то можно слушать различные события, используя TextView.OnEditorActionListener . Интерфейс TextView.OnEditorActionListener предоставляет метод *onEditorAction*, который

распознает тип действия с помощью ID действивя, например *IME_ACTION_SEND* или *IME_ACTION_SEARCH*.

Например, вот как можно слушать событие, которое генерируется, когда пользователь нажимает на клавиатуре кнопку «Отправить»:

```
EditText editText = (EditText) findViewById(R.id.search);
editText.setOnEditorActionListener(new OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
        boolean handled = false;
        if (actionId == EditorInfo.IME_ACTION_SEND) {
            sendMessage();
            handled = true;
        }
        return handled;
    }
});
```

3.3.3. AutoCompleteTextView

Для поля ввода можно добавить автодополнение.

3.4. CheckBox

Checkbox позволяет пользователю выбрать одну или несколько опция из набора. Обычно, каждый checkbox следует располагать на отдельной строке.

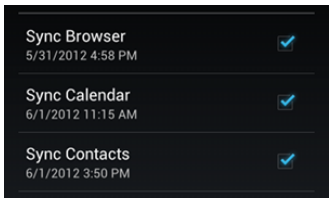


Рис. 7: Checkbox

3.5. RadioButton

RadioButton позволяет пользователю выбрать одну опцию из набора. Этот элемент стоит использовать для взаимоисключающего набора опций, которые должны отображаться друг за другом.

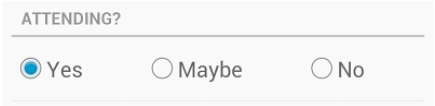


Рис. 8: RadioButton

3.6. ToggleButton

ToggleButton позволяет пользователю менять опцию между двумя состояниями.

Рис. 9: Toggle buttons

Рис. 10: Switches (начиная с Android 4.0+)

3.7. Spinner

Spinner предоставляет быстрый способ выбрать одно значение из набора. По умолчанию, Spinner показывает выбранное значение. Касание отображает выпадающее меню со всеми другими доступными значениями, из которых пользователь может выбрать новое.

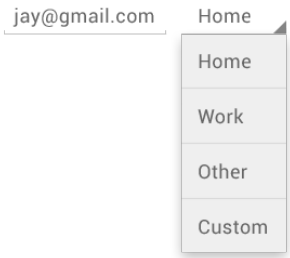


Рис. 11: Spinner

3.8. DatePickerDialog, TimePickerDialog

Андроид предоставляет элементы управления, которые позволяют пользователю выбрать время или дату.

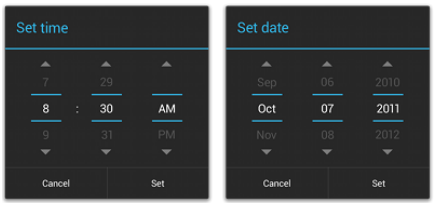


Рис. 12: Pickers

Пункт #4
Логи

4.1. Логи

При тестировании приложения может быть полезно посмотреть логи работы. Они отображаются в окне logcat. Чтобы его найти, нужно тыкнуть кнопку Monitor, которая находится внизу слева.

4.2. Уровни логов

У логов бывают разные уровни: Verbose, Debug, Info, Warn, Error, Assert. Как видно из названий, чем дальше, тем более специфичные вещи туда попадают. В окне logcat можно выбрать, какие именно уровни нам интересны. Тогда будут показаны сообщения, уровни которых не ниже того, что мы указали.

4.3. Просмотр логов

Вот мы запустили программу и хотим посмотреть логи. Для этого смотрим в окошко logcat и видим там следующую картину.

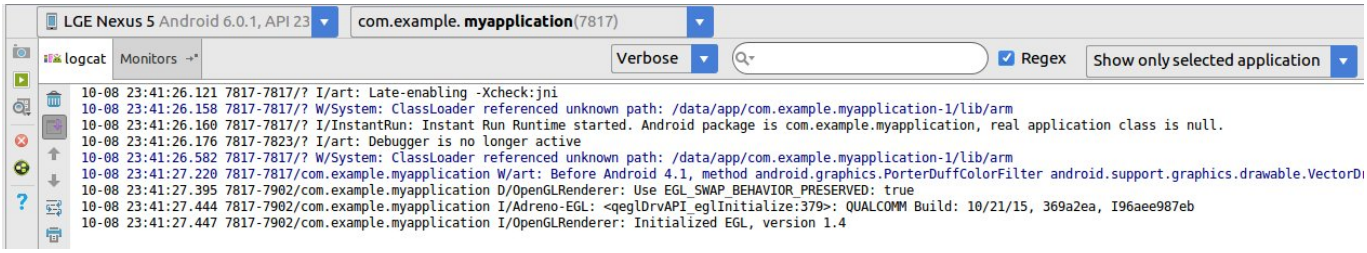


Рис. 13: тут показаны вообще все логи, которые есть

Можем поменять уровень просматриваемых логов.

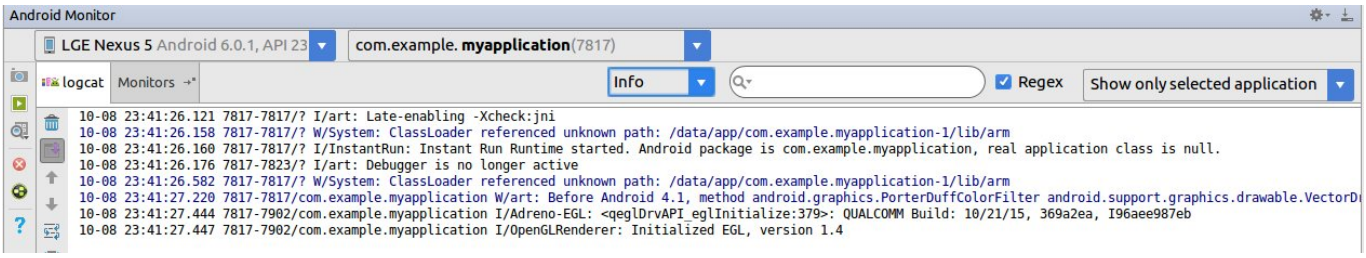


Рис. 14: а тут только логи уровня Info и выше

Также рядом расположена строка поиска по логам.

Посмотрим теперь на строки, которые нам выводятся. Каждая запись начинается с *L/t:*, где *L* обозначает уровень лога, а *t* – тег. Тег – это что-то типа метки, чтобы легче было найти нужное сообщение.

4.4. Запись в логи

Мы можем сами создавать логи. Делается это с помощью класса Log и его методов Log.v() Log.d() Log.i() Log.w() и Log.e(). Названия методов соответствуют уровню логов, которые они запишут. Есть еще метод Log.wtf(), который расшифровывают как "What a Terrible Failure!"и который записывает логи с уровнем Assert.

У каждого такого метода два параметра – тег и текст сообщения. Напишем по логу каждого уровня и посмотрим, что будет.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Log.v("mySuperTag", "Hello, Verbose");
    Log.d("mySuperAnotherTag", "Hello, Debug");
    Log.i("oneMoreTag", "Hello, Info");
    Log.w("myTag", "Hello, Warn");
    Log.e("myTag", "Hello, Error");
    Log.wtf("myTag", "What a Terrible Failure!");
}
```

Жмем Alt+Enter, и все компилируется.

Посмотрим, как работает.

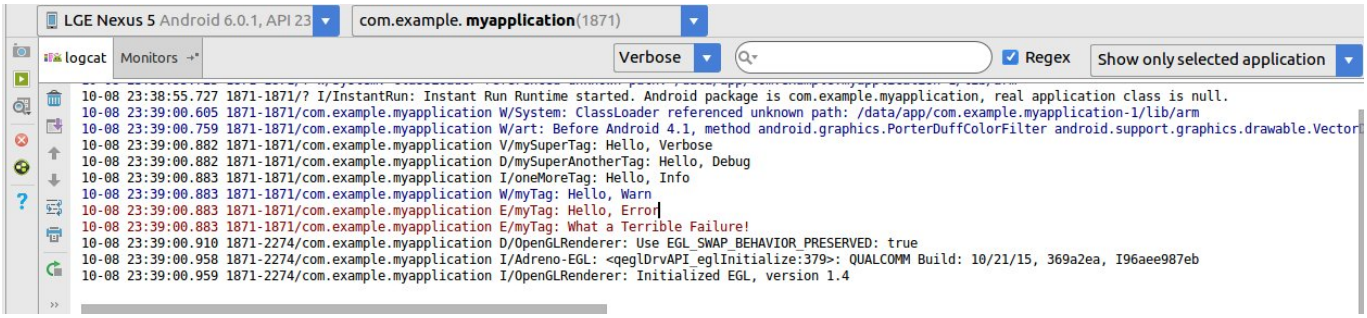


Рис. 15:

Собственно, все работает так, как мы и хотели, разве что Log.wtf() думает, что он не Assert, а Error. Ну бывает.

Пункт #5
Всплывающие сообщения (Toasts)

5.1. Вывод всплывающего сообщения

Приложение может показывать всплывающие сообщения с помощью класса Toast. У него есть статический метод makeText(), который создает View-элемент Toast. В качестве параметров он принимает:

- **context** – это нам пока не надо, просто напишем здесь *this*.
- **text** – это текст, который будет показан на экране.
- **duration** – продолжительность показа (Toast.LENGTH_LONG - длинная, Toast.LENGTH_SHORT - короткая).

Toast создан, и, чтобы он отобразился на экране, вызывается метод show().

```
1 @Override
2 protected void onCreate(Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4     setContentView(R.layout.activity_main);
5
6     Toast.makeText(this, "[^_^]", Toast.LENGTH_LONG).show();
7 }
```

Запускаем, смотрим и радуемся.

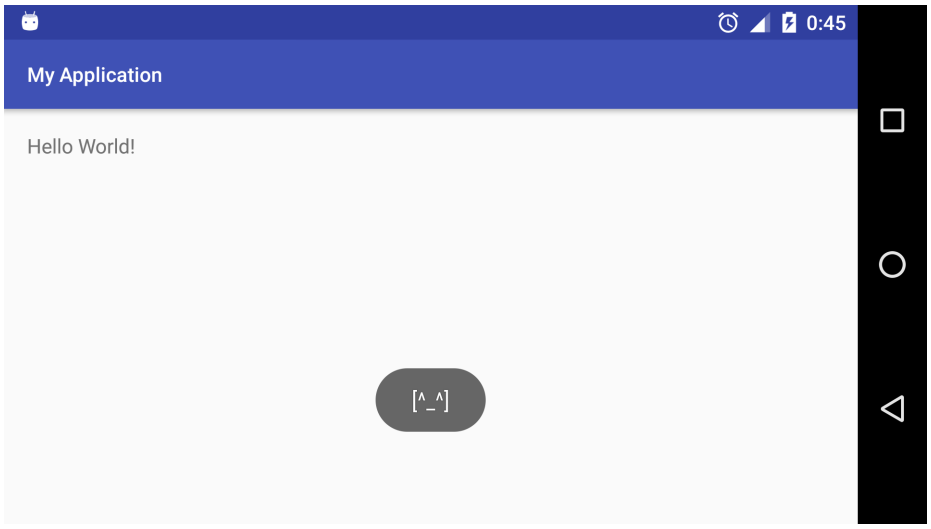
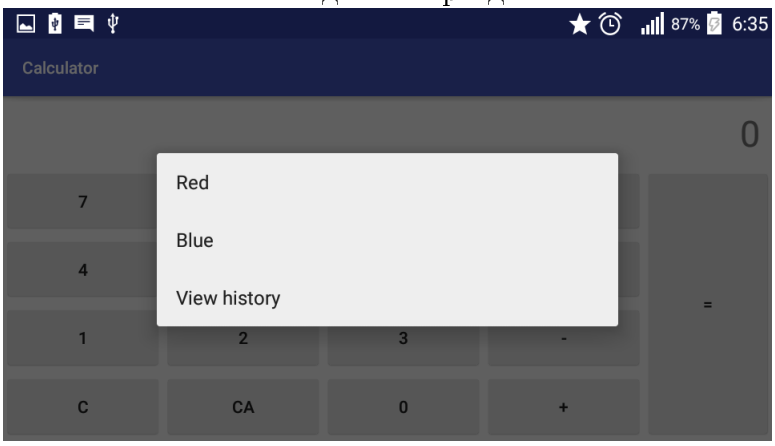


Рис. 16:

Пункт #6
Контекстные меню (Contextual menus)

6.1. Контекстное меню

Контекстное меню создается при долгом нажатии на экран.



6.2. Создание контекстного меню

Чтобы создать контестное меню, нужно в классе, соответвующем текущей Activity:

- Вызвать метод registerForContextMenu(View view_i);

  Хотим чтобы появлялось контекстное меню при долгом нажати на view_i.

- Создать обработчик onCreateContextMenu, который будет вызываться каждый при открытии контекстного меню.

  void onCreateContextMenu(android.view.ContextMenu menu, view_i, ContextMenuInfo menuInfo);

  где,

  - menu – то, куда добавим пункты.

    оно при каждом вызове onCreateContextMenu новое.

  - view_i – элемент экрана, для которого вызвалось контекстное меню.

```
protected void onCreate(Bundle savedInstanceState) {

    //Some code

    // registerForContextMenu(view);

    registerForContextMenu(findViewById(R.id.numberField));
}

public void onCreateContextMenu(android.view.ContextMenu menu, View view_i,
    ContextMenuInfo menuInfo) {
    switch (view_i.getId()) {
        case R.id.numberField:
            // menu.add(groupId, itemId, order, label);
            menu.add(0, COLOR_BLUE_ITEM_ID, 2, "Blue");
            menu.add(0, COLOR_RED_ITEM_ID, 0, "Red");
            menu.add(1, VIEW_HISTORY_ITEM_ID, 100500, "View history");
            break;
    }
}
```

6.3. Обработка событий контекстного меню

Для обработки события выбора пункта меню добавим метод boolean onContextItemSelected(MenuItem item);

```
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case ContextMenu.COLOR_RED_ITEM_ID:
            display.setTextColor(Color.RED);
            break;
        case ContextMenu.COLOR_BLUE_ITEM_ID:
            display.setTextColor(Color.BLUE);
            break;
        case ContextMenu.VIEW_HISTORY_ITEM_ID:
            showHistory();
            break;
        default:
            Log.e("myTag", "unknown context menu item");
    }
    return super.onContextItemSelected(item);
}
```

6.4. Создание контекстного меню из макета (.xml)

Пусть xml файл лежит в app/res/menu/context_menu.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:title="disable"
        android:id="@+id/contextmenu_disable" />
    <item
        android:id="@+id/contextmenu_clearall"
        android:orderInCategory="1000"
        android:title="clear all ">
    </item>
    <item
            android:id="@+id/contextmenu_backspace"
            android:title="backspace">
    </item>
</menu>
```

XML меню часто удобней: описыватся, как и большая чать UI в *.xml; меньше думаешь об ID.

Создание menu из xml выглядит коротко:

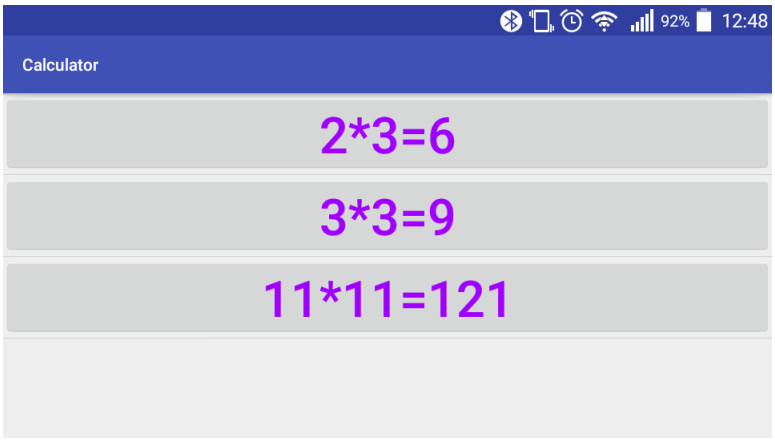
```
public void onCreateContextMenu(android.view.ContextMenu menu, View view_i,
    ContextMenuInfo menuInfo) {
    switch (v.getId()) {
        case R.id.btnClear:
            getMenuInflater().inflate(R.menu.context_menu, menu);
            break;
}
```

Обработка выбора пункта аналогична:

```
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.contextmenu_backspace:
            backspace();
            break;
        case R.id.contextmenu_clearall:
            clear();
            break;
        case R.id.contextmenu_disable:
            clearButtonsEnabled = false;
            break;
        default:
            Log.e("myTag", "unknown context menu item");
    }
    return super.onContextItemSelected(item);
}
```

Пункт #7
Список (ListView)

7.1. Класс ListView



ListView – виджет прокручиваемого списка.

7.2. Adapter, Класс ArrayAdapter, создание списка

Добавим в XML нужной Activity ListView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_history"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- added -->
    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/lvHistory" />

</LinearLayout>
```

Добавим в onCreate строчки:

```
String [] history = {"2*3=6", "3*3=9", "11*11=121"};

ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
                R.layout.my_super_list_item, history);

((ListView) findViewById(R.id.lvHistory)).setAdapter(adapter);
```

Заполненный ListView появится на экране.
Для заполнения ListView используется адаптер.
В этом пример используется ArrayAdapter<>.

Праметры конструктора ArrayAdapter<>:

- this – контекст
- R.layout.my_super_list_item – собственная разметка для элемента ListView.
- вместо history может быть массив объектов с методом toString();

7.3. Создание списка из макета (.xml)

String [] можно получать из xml.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="default_history">
        <item>2*3=6</item>
        <item>3*3=9</item>
        <item>11*11=121</item>
        <item>32*32=1024</item>
    </string-array>
</resources>
```

Код заполнения почти не изменился:

```
String [] history = getResources().getStringArray(R.array.default_history); // !!!

ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
                R.layout.my_super_list_item, history);

((ListView) findViewById(R.id.lvHistory)).setAdapter(adapter);
```