Лекция 7. БАЗОВЫЕ ЭЛЕМЕНТЫ НАВИГАЦИИ

7.1 Меню

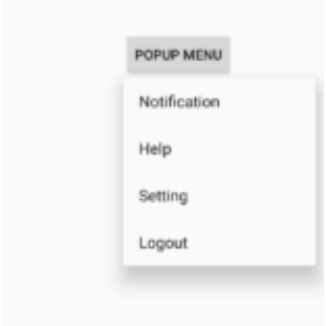
Начиная с версии Android 3.0 (уровень API 11) меню переместилось в *ActionBar*. Существуют следующие виды меню:

1)меню параметров и строка действий (Option Menu ActionBar**):** пункты меню параметров размещаются в строке действий в виде сочетания отображаемых на экране вариантов действий и раскрывающегося списка дополнительных вариантов выбора (рис. 7.1);

Рис.7.1 ***Меню параметров и строка действий***

2) **контекстное меню и режим контекстных действий (Contex Menu)***:* это плавающее меню, которое открывается, когда пользователь длительно нажимает на элемент. В режиме контекстных действий в строке, расположенной вверху экрана, отображаются пункты действий, затрагивающие выбранный контент, причем пользователь может выбрать сразу несколько элементов;

3)**всплывающее меню (Popup menu)***:* отображается вертикальный список пунктов, который привязан к представлению, вызвавшему меню.



7.1.1 Определение меню в файле XML

Для определения пунктов меню используется стандартный формат XML в ресурсе меню. После этого ресурс меню можно будет загружать как объект *Menu* в активности или фрагменты.

Использовать меню в ресурсах рекомендуется по нескольким причинам:

1) в XML проще визуализировать структуру;

2) позволяет отделить контент меню от кода, определяющего работу приложения;

3) позволяет создавать альтернативные варианты меню для разных версий платформы, размеров экрана и других конфигураций.

Чтобы определить меню, создается файл в папке *res/menu/имя.xml* проекта и определяется меню со следующими элементами (результат выполнения представлен на рис):

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

    <item android:id="@+id/newA"
        android:icon="@drawable/metro_android2_black"
        android:title="@string/newel"/>

    <item android:id="@+id/nextB"
        android:icon="@drawable/beret"
        android:title="@string/nextel" />

</menu>
```

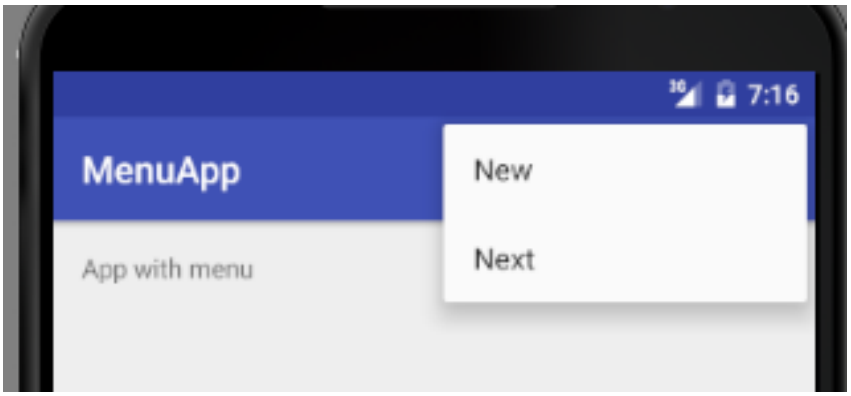


Рис.7.2 Пример приложения с меню

В фале описания меню:

— <*menu*>: определяет класс *Menu*, который является контейнером для пунктов. Элемент <*menu*> должен быть корневым узлом файла, в котором может находиться один или несколько элементов <*item*> и <*group*>;

— <*item*>: создает класс *MenuItem*, который представляет один пункт меню. Этот элемент может содержать вложенный элемент <*menu*> для создания вложенных меню:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">

  <item android:id="@+id/file"
    android:title="@string/file" >
    <!-- вложенное меню -->
    <menu>

        <item android:id="@+id/create_new"
            android:title="@string/create_new" />

        <item android:id="@+id/open"
            android:title="@string/open" />

    </menu>
 </item>

</menu>
```

— <*group*>: необязательный, невидимый контейнер для элементов *item*. Он позволяет разделять пункты меню на категории и назначать им одинаковые свойства, такие как активное состояние и видимость. Содержит <*item*> и должна быть дочерней от <*menu*>.

Элемент <*item*> поддерживает несколько атрибутов, с помощью которых можно определить внешний вид и поведение. Пункты меню чаще всего имеют следующие атрибуты:

— *android*:*id*: идентификатор ресурса;

— *android*:*icon*: ссылка на графический элемент, который будет использоваться в качестве значка пункта меню;

— *android*:*title*: ссылка на строку, которая будет использоваться в качестве названия пункта меню;

— *android*:*orderInCategory*: порядок следования элемента в меню;

— *android*:*showAsAction*: указывает, когда и как этот пункт должен отображаться в строке действий. Принимает следующие значения:

=["ifRoom" | "never" | "withText" | "always" | "collapseActionView"]

— *ifRoom*: если есть место для него, если нет, то для всех элементов с надписью «ifRoom», элементы с наименьшими значениями *OrderInCategory* отображаются как действия, а остальные элементы отображаются в меню переполнения;

— *withText*: включает текст заголовка;

— *never*: никогда не помещать этот элемент в *actionbar*;

— *always*: всегда размещать этот элемент в *actionbar*.

```
<item android:id="@+id/newA"
android:icon="@drawable/metro_android2_black"
        android:title="@string/newel"/>
        <!--android:showAsAction="ifRoom"/>-->
```

…

Место, где отображаются на экране пункты меню параметров, определяется версией платформы, для которой разработано приложение. Если приложение написано для версии Android 2.3.x (уровень API 10) *или более ранней*, содержимое меню параметров отображается внизу экрана, когда пользователь нажимает кнопку *Меню*. Если приложение предназначено для версии Android 3.0 (уровень API 11) *и более поздних*, пункты меню

параметров будут отображаться в *строке действий*. Чтобы обеспечить быстрый доступ к важным действиям, можно принудительно разместить несколько пунктов меню в строке действий, добавив *android*:*showAsAction*="*ifRoom*" к соответствующим элементам (рис.7.3).

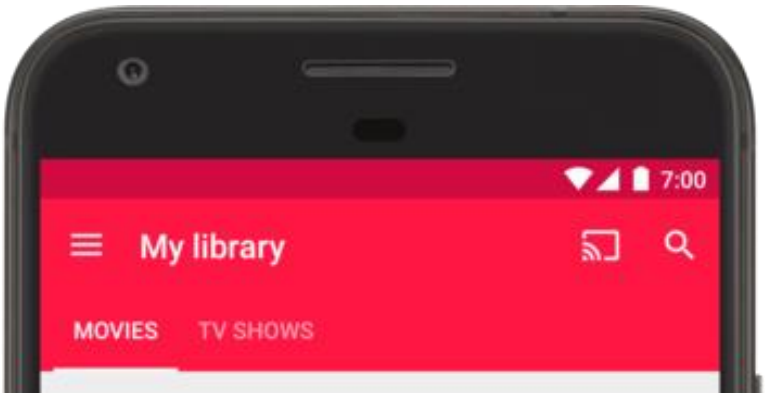


Рис.7.3 Пример *ActionBar*

В современных устройствах меню является частью *ActionBar*.

7.1.3 Вывод меню на экран

Мы определили меню, но само определение элементов в файле еще не создает меню. Это лишь декларативное описание. Чтобы вывести его на экран, надо вызвать его в классе *Activity*. Для этого надо переопределить метод *onCreateOptionsMenu*:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

   MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_ops, menu);
    return true;
}
```

Программа должна сконвертировать созданный ресурс меню в программный объект. Для этой цели существует специальный метод *MenuInflater*.*inflate*(). Он предназначен для вывода меню при нажатии кнопки на устройстве. В качестве первого параметра метод принимает ресурс, представляющий декларативное описание меню в XML, и наполняет им объект *menu*, переданный в качестве второго параметра. Метод должен возвращать значение *true*, чтобы меню было видимым на экране.

Метод *getMenuInflater*() возвращает экземпляр класса *MenuInflater*, который используем для чтения данных меню из XML.

Всего можно одновременно вывести на экран шесть пунктов меню. Если пунктов больше, то будет выведено пять плюс шестой пункт *More*, который позволит увидеть остальные пункты при нажатии.

7.1.4 Обработка нажатий

Когда пользователь выбирает пункт меню (в том числе пункты из *строки действий*), система вызывает метод *onOptionsItemSelected*() активности. Этот метод передает в

параметрах класс *MenuItem*. Например, (результат работы фрагмента листинга представлен на рис. 7.4):

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

    // TODO
    Toast.makeText(this, item.getTitle(), Toast.LENGTH_LONG).show();
    return super.onOptionsItemSelected(item);
}
```

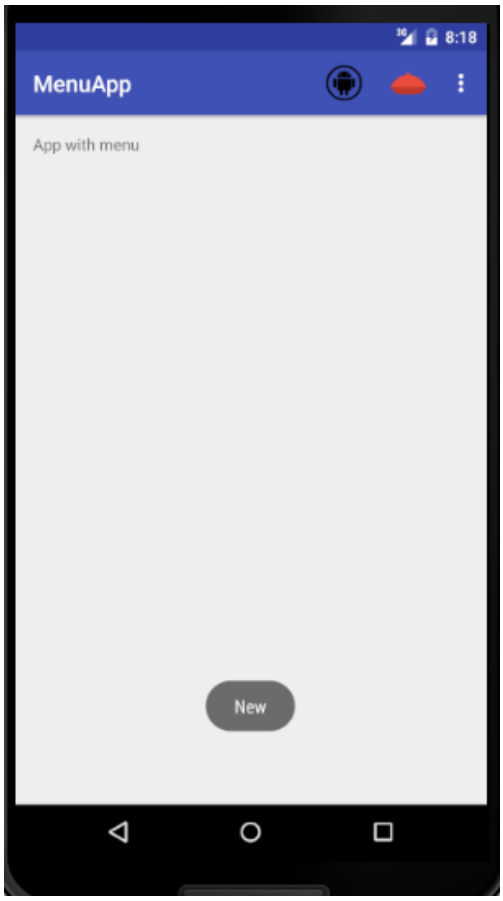


Рис.7.4 Обработка пунктов меню

Когда пункт меню успешно обработан, возвращается *true*. Если пункт меню не обрабатывается, следует вызвать реализацию суперкласса *onOptionsItemSelected*() (реализация по умолчанию возвращает значение *false*).

Идентифицировать пункт можно, вызвав метод *getItemId()*, который возвращает уникальный идентификатор пункта меню (определенный атрибутом *android:id* из ресурса меню). Этот идентификатор можно сопоставить с известными пунктами, чтобы выполнить соответствующее действие:

```
@Override
    public boolean onOptionsItemSelected(MenuItem item) {

        // TODO
        switch (item.getItemId()) {
            case R.id.newA:
                newA();
                return true;
            case R.id.nextB:
                next();
```

```
                return true;
            case R.id.saveC:
                saveC();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
```

Если в активности имеются фрагменты, система сначала вызовет метод *onOptionsItemSelected*(), а затем будет вызывать этот метод для каждого фрагмента (в том порядке, в котором они были добавлены), пока он не возвратит значение *true* или не закончатся фрагменты. Меню можно обработать следующим образом:

```
MenuItem mItem=menu.findItem(R.id.newA);
mItem.setOnMenuItemClickListener(

        new MenuItem.OnMenuItemClickListener() {
            @Override
            public boolean onMenuItemClick(MenuItem item) {
               // ....
               return false;
            }
        });
```

7.1.5 Программное создание меню и изменение пунктов меню во время выполнения приложения

Если требуется изменять меню в зависимости от событий, которые возникают в течение жизненного цикла активности, сделать это можно в методе *onPrepareOptionsMenu*(). Этот метод вызывается каждый раз, когда меню отображается или перерисовывается. Он передает объект *Menu* в том виде, в котором он в данный момент существуе. Его можно изменить путем, например, добавления, удаления или отключения пунктов меню:

```
@Override
public boolean onPrepareOptionsMenu (Menu menu){

     menu.add("Delete");
     menu.add("Move");
   //  menu.removeItem(R.id.nextB);

 return super.onPrepareOptionsMenu(menu);
 }
```

У метода *add()* могут быть четыре параметра:

идентификатор группы – позволяет связывать пункт меню с группой других пунктов этого меню;

идентификатор пункта – для обработчика события выбора пункта меню;

порядок расположения пункта в меню – позволяет определять позицию в меню, по умолчанию (*Menu.NONE* или 0) пункты идут в том порядке, как задано в коде;

заголовок – текст, который выводится в пункте меню (результат выполнения представлен на рис.7.5):

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_ops, menu);
    MenuItem menuItem= menu.add(Menu.NONE,111, Menu.NONE, "Move");
    menuItem.setShowAsAction(MenuItem.SHOW_AS_ACTION_ALWAYS);
    return true;
}
```

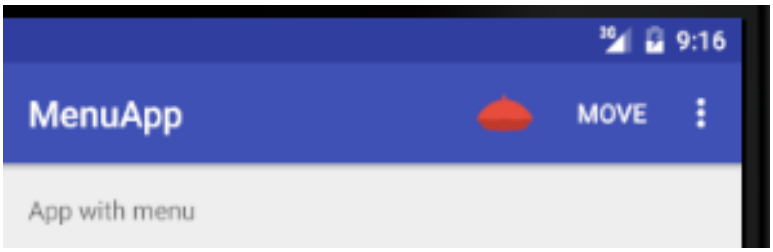


Рис.7.5 Примен добавления пунктов меню в коде

7.1.6 Создание контекстного меню

Кроме стандартного меню в Android используется также *контекстное меню*, вызываемое при нажатии на объект в течение двух-трёх секунд (событие long-press). В отличие от обычного меню, в контекстном меню не поддерживаются значки и быстрые клавиши. Второе важно отличие – контекстное меню применимо к *View*, а меню к *Activity*. Поэтому в приложении может быть одно меню и несколько контекстных меню, например, у каждого элемента *TextView*.

Существует два способа предоставления возможности контекстных действий:

— *в плавающем контекстном меню:* меню отображается в виде плавающего списка пунктов меню (наподобие диалогового окна). Пользователи могут каждый раз выполнять контекстное действие только с одним элементом (представлено на рис.7.6);

Рис.7.6 Плавающее контекстное меню

— *в режиме контекстных действий* (представлен на рис.7.7): этот режим является системной реализацией *ActionMode*, которая отображает *строку контекстных действий*

вверху экрана с пунктами действий, которые затрагивают выбранные элементы. Когда этот режим активен, пользователи могут одновременно выполнять действие с несколькими элементами (если это допускается приложением). Если этот режим предусмотрен, именно *его рекомендуется использовать* для отображения контекстных действий.

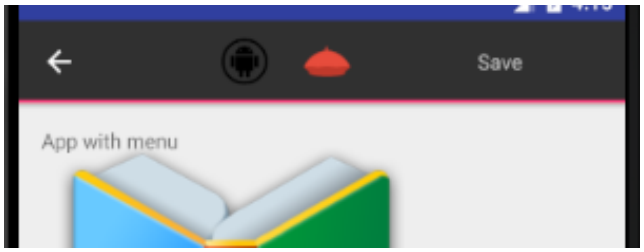


Рис.7.7 Плавающее меню в режиме контекстных действий

Рассмотрим алгоритм *создания плавающего контекстного меню***.** Вначале зарегистрируем класс *View*, с которым следует связать контекстное меню, вызвав метод *registerForContextMenu*() и передав ему *View*:

```
setContentView(R.layout.activity_menu);
ImageView img=(ImageView) findViewById(R.id.image_book);
registerForContextMenu(img);
```

Если активность использует *ListView* или *GridView* и требуется, чтобы каждый элемент предоставлял одинаковое контекстное меню. Зарегистрируем все элементы для контекстного меню. Затем реализуем метод *onCreateContextMenu*() в активности или фрагменте:

```
@Override
public void onCreateContextMenu( ContextMenu menu,View v,
                        ContextMenu.ContextMenuInfo menuInfo) {

    super.onCreateContextMenu(menu, v, menuInfo);
    getMenuInflater().inflate(
            R.menu.menu_ops,menu);
}
```

Когда зарегистрированное представление примет событие длительного нажатия, система вызовет метод *onCreateContextMenu*(). Именно здесь определяются пункты меню. Делается это обычно путем загрузки ресурса меню. Реализуем метод *onContextItemSelected*(). Когда пользователь выбирает пункт меню, система вызывает этот метод, с тем чтобы вы могли выполнить соответствующее действие:

```
@Override
public boolean onContextItemSelected(

        MenuItem item) {
    switch (item.getItemId()) {
        case R.id.newA:
            newA();
            return true;
        case R.id.nextB:
            next();
```

```
            return true;
        case R.id.saveC:
            saveC();
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

Рассмотрим вариант реализации *режима контекстных действий*. Он представляет собой системную реализацию класса *ActionMode*. Когда пользователь использует этот режим, выбирая элемент, вверху экрана открывается *строка контекстных действий*, содержащая действия, которые пользователь может выполнить с выбранными в данный момент элементами. Режим контекстных действий отключается, а строка контекстных действий исчезает, когда пользователь снимет выделение со всех элементов, нажмет кнопку «*Назад*» или выберет действие «*Готово*», расположенное с левой стороны строки.

Строка контекстных действий не обязательно бывает связана со строкой действий. Они работают независимо друг от друга, даже несмотря на то, что визуально строка контекстных действий занимает положение строки действий.

Для включения режима контекстных действий для отдельных представлений следует:

1) захватить элемент и добавить обработчик на длительное нажатие:

```
EditText etx=(EditText)findViewById(R.id.edit_text);

etx.setOnLongClickListener(
        new View.OnLongClickListener() {
            @Override
            public boolean onLongClick(View v) {
                return false;
            }
        }
);
```

2) реализовать интерфейс *ActionMode.Callback*. В его методах обратного вызова можно указать действия для строки контекстных действий, реагировать на нажатия пунктов и обрабатывать другие события жизненного цикла для режима действий:

```
private ActionMode.Callback
        mActionModeCallback=new ActionMode.Callback(){

    @Override
    public boolean onCreateActionMode(ActionMode actionMode, Menu menu) {
        return false;
    }

    @Override
    public boolean onPrepareActionMode(ActionMode actionMode, Menu menu) {
        return false;
    }

    @Override
    public boolean onActionItemClicked(ActionMode actionMode, MenuItem menuItem) {
        return false;
    }

    @Override
    public void onDestroyActionMode(ActionMode actionMode) {
      actionMode = null;
```

```
    }
};
```

Например, может быть такая реализация (в случае если меню определено в ресурсах):

```
@Override
public boolean onCreateActionMode(ActionMode actionMode, Menu menu) {

    actionMode.getMenuInflater().inflate(
            R.menu.menu_ops, menu);
    return true;
}
```

3) вызывать *startActionMode*(), когда требуется показать строку (например, когда пользователь выполняет длительное нажатие представления):

```
EditText etx=(EditText)findViewById(R.id.edit_text);
etx.setOnLongClickListener(
        new View.OnLongClickListener() {

            @Override
            public boolean onLongClick(View v) {
               startActionMode(mActionModeCallback);
                v.setSelected(true);
                return true;
            }
        }
```

При вызове метода *startActionMode*() система возвращает созданный класс *ActionMode*. Сохранив его в составной переменной, можно вносить изменения в строку контекстных действий в ответ на другие события (рис. 7.8).

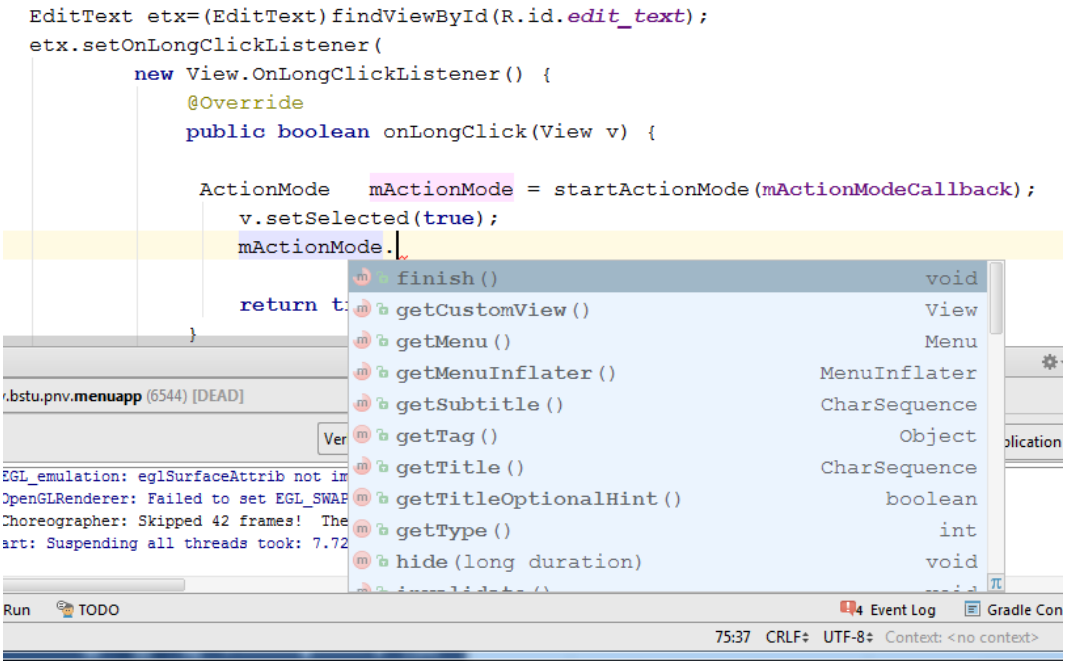


Рис.7.8 Управление строкой контекстных действий

7.1.7 Создание всплывающего меню

Всплываюшее или *PopupMenu* является модальным меню, привязанным к *View*. Оно отображается ниже представления, к которому привязано, если там есть место, либо поверх него (рис. 7.9).

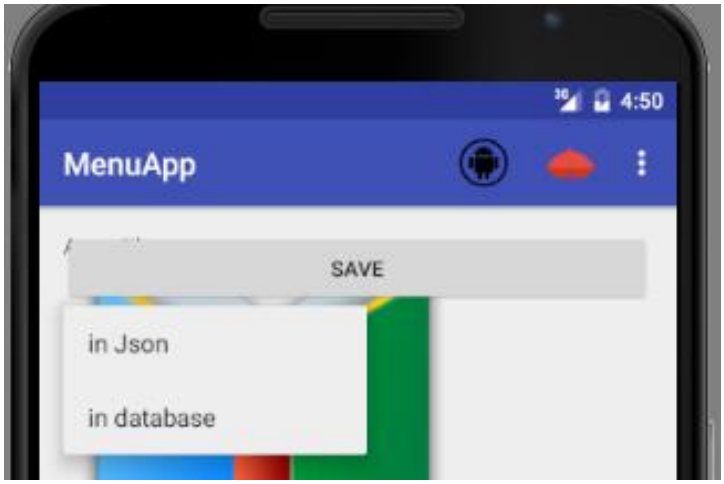


Рис.7.9 Всплывающее меню

Если для определения меню используется XML:

```
<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:id="@+id/sJ"
          android:title="@string/in_json"/>

    <item android:id="@+id/sDB"
          android:title="@string/in_database"/>

</menu>
```

то:

1) создается экземпляр класса *PopupMenu* с помощью конструктора, принимающий текущие *Context* и *View* приложения;

2) с помощью *MenuInflater* загружается ресурс меню в объект *Menu*, возвращенный методом *PopupMenu.getMenu*(). На API уровня 14 и выше вместо этого можно использовать *PopupMenu.inflate*();

3) вызывается метод *PopupMenu.show*():

```
Button button = (Button) findViewById(R.id.btn_popup);
button.setOnClickListener(
        new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                PopupMenu popup = new PopupMenu(v.getContext(),v);
                popup.inflate(R.menu.menu_popup_btn);
                popup.show();
```

```
        }
    });
```

7.1.8 Создание приложения «Блокнот»

Создадим приложение – аналог блокнота, позволяющий записывать и читать данные, только из одного фиксированного файла.

На экране активности разместим компонент *EditText* на весь экран. Заменим файл *activity_main.xml* следующим содержимым:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="top|left"
        android:inputType="textMultiLine" />

</LinearLayout>
```

Создадим меню. Оно будет содержать два пункта «*Открыть*» и «*Сохранить*». Понадобятся иконки. Для этого есть много ресурсов с иконками (в свободном доступе). Например, https://material.io/icons/. Скачаем иконки и разместим файлы в папке *drawable* (рис. 7.10):

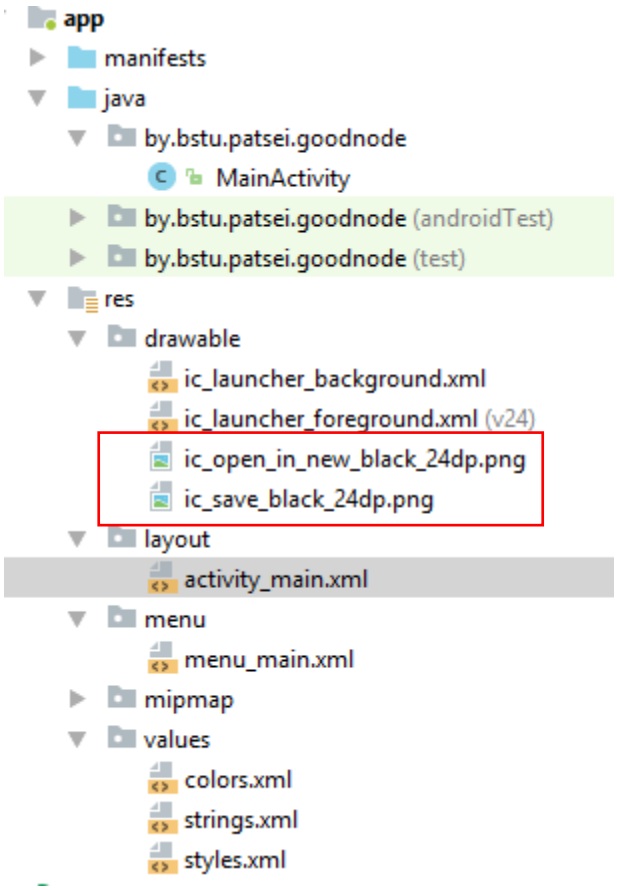


Рис.7.10 Размещение иконок в проекте

Создадим еще один файл с пунками меню, определенными в *res/menu/menu_main.xml* и со следующим содержимым:

```
<menu 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"
    tools:context=".MainActivity">

    <item
        android:id="@+id/action_open"
        android:icon="@drawable/ic_open_in_new_black_24dp"
        android:orderInCategory="100"
        app:showAsAction="ifRoom|withText"
        android:title="@string/action_open" />

    <item
        android:id="@+id/action_save"
        android:icon="@drawable/ic_save_black_24dp"
        android:orderInCategory="100"
        app:showAsAction="ifRoom|withText"
        android:title="@string/action_save" />

</menu>
```

Как видно, здесь идет обращение к строковым ресурсам. Поэтому изменим содержание файла *string.xml*:

```
<resources>

        <string name="app_name">GoodNode</string>
        <string name="action_open">Открыть</string>
        <string name="action_save">Сохранить</string>
</resources>
```

Добавим меню и для методов обработчиков напишем *openFile*() и *saveFile*() в которых, реализуем операции по открытию и сохранению файла. Содержимое класса *MainActivity* будем следующим*:*

```
public class MainActivity extends AppCompatActivity {

    private final static String FILENAME = "sample.txt"; // имя файла
        private EditText mEditText;

     @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            //Окно редактора
            mEditText = (EditText) findViewById(R.id.editText);
        }

    @Override
     public boolean onCreateOptionsMenu(Menu menu) {
            getMenuInflater().inflate(R.menu.menu_main, menu);
            return true;
        }

     @Override
     public boolean onOptionsItemSelected(MenuItem item) {
```

```
            // выбор и обработка пунктов меню
            switch (item.getItemId()) {
                case R.id.action_open:
                    openFile(FILENAME);
                    return true;
                case R.id.action_save:
                    saveFile(FILENAME);
                    return true;
                default:
                    return true;
            }
        }

// Метод для открытия файла
  private void openFile(String fileName) {
   try {
        InputStream inputStream = openFileInput(fileName);
        if (inputStream != null) {
          InputStreamReader isr = new InputStreamReader(inputStream);
           BufferedReader reader = new BufferedReader(isr);
           String line;
           StringBuilder builder = new StringBuilder();
           while ((line = reader.readLine()) != null) {
                   builder.append(line + "\n");
                   }
            inputStream.close();
            mEditText.setText(builder.toString());
               }
       } catch (Throwable t) {
          Toast.makeText(getApplicationContext(),
           "Exception: " + t.toString(), Toast.LENGTH_LONG).show();
           }
   }

// Метод для сохранения файла
 private void saveFile(String fileName) {
   try {
      OutputStream outputStream = openFileOutput(fileName, 0);
      OutputStreamWriter osw = new OutputStreamWriter(outputStream);
               osw.write(mEditText.getText().toString());
               osw.close();
       } catch (Throwable t) {
            Toast.makeText(getApplicationContext(),
            "Exception: " + t.toString(), Toast.LENGTH_LONG).show();
           }
       }
   }
```

Запускам приложение. Должен получится результат, представленный на рис. 7.11.

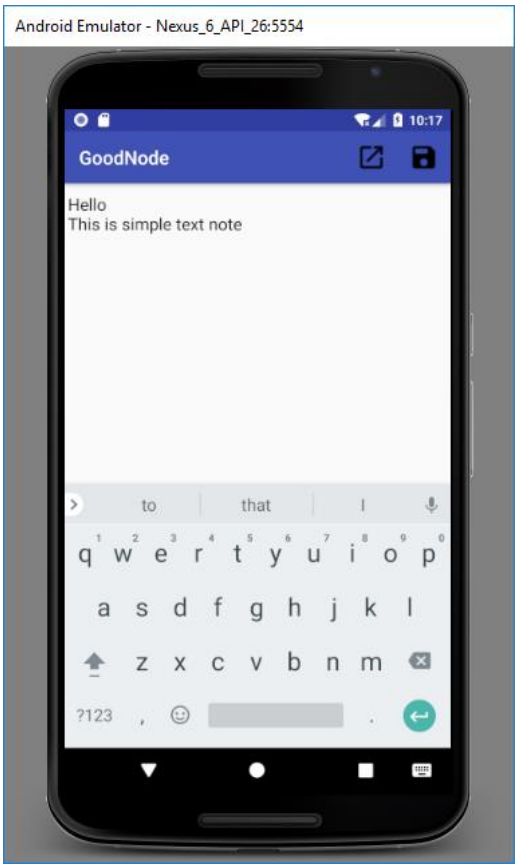


Рис.7.11 Результат раоты приложения «Блокнот»

7.2 Диалоговые окна

В некоторых случаях требуется показать диалоговое окно, где пользователю нужно сделать какой-нибудь выбор или показать сообщение об ошибке. В Android уже есть собственные встроенные диалоговые окна, которые гибко настраиваются под задачи. Использование диалоговых окон для простых задач позволяет сократить число классов в приложении, экономя ресурсы памяти.

Диалоговые окна в Android представляют собой *полупрозрачные «плавающие» активности*, частично перекрывающие родительский экран, из которого их вызвали. Как правило, они затеняют родительскую активность позади себя с помощью фильтров размывания или затемнения. Вы можете установить заголовок с помощью метода *setTitle*() и содержимое с помощью метода *setContentView*().

Android поддерживает следующие типы диалоговых окон:

— *Dialog*: базовый класс для всех типов диалоговых окон;

— *AlertDialog*: диалоговое окно с кнопками, списком, флажками или переключателями;

— *CharacterPickerDialog*: диалоговое окно, позволяющее выбрать символ с ударением, связанный с базовым символом;

— *DatePickerDialog*: диалоговое окно выбора даты с элементом *DatePicker*;

— *TimePickerDialog*: диалоговое окно выбора времени с элементом *TimePicker*.

Поскольку *ProgressDialog*, *TimePickerDialog* и *DatePickerDialog* – расширение класса *AlertDialog*, они также могут иметь командные кнопки. Если ни один из существующих типов диалоговых окон не подходит, то можно создать собственное диалоговое окно.

7.2.1 Класс Dialog

Класс *Dialog* является базовым для всех классов диалоговых окон. Каждое диалоговое окно должно быть определено внутри активности, в которой будет использоваться. Для его отображения необходимо вызвать метод *showDialog*() и передать в качестве параметра идентификатор диалога (константа, которую надо объявить в коде программы).

Метод *dismissDialog*() прячет диалоговое окно (но не удаляет), не отображая его на экране. Окно остается в пуле диалоговых окон данной активности. При повторном отображении при помощи метода *showDialog*() будет использована кэшированная версия окна.

Метод *removeDialog*() удаляет окно из пула окон данной активности. При повторном вызове метода *showDialog*() диалоговое окно придется создавать снова.

Рассмотрим пример создания диалогового окна на основе класса *Dialog*. Создаем простейшую разметку для диалогового окна - текстовое поле внутри *LinearLayout*:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

        <TextView
            android:id="@+id/dialogTextView"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:clickable="true"
            android:textSize="22sp"/>

</LinearLayout>
```

В разметку главной активности добавим кнопку для вызова диалогового окна:

```
public void onClick(View v)
{
    // Выводим диалоговое окно на экран
    dialog.show();
}
```

В коде для главной активности определим:

```
public class MainActivity extends FragmentActivity {
Dialog dialog;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
      ...

        dialog = new Dialog(MainActivity.this);
        // Установите заголовок
        dialog.setTitle("Dialog window");
        // Передайте ссылку на макет
        dialog.setContentView(R.layout.dialog);
        // Найдите элемент TextView внутри вашей разметки
        // и установите ему соответствующий текст
       TextView text = (TextView)
```

```
                dialog.findViewById(R.id.dialogTextView);
        text.setText("Delete list?");
    }
```

Запустим приложение. На экране должен отобразится диалог (рис7.12).

Рис.7.12 Интерфейс с диалоговым окном

По умолчанию при показе диалогового окна главная активность затемняется. В документации есть константы, позволяющие управлять степенью затемнения:

```
WindowManager.LayoutParams lp = dialog.getWindow().getAttributes();
lp.dimAmount = 0.5f; // уровень затемнения от 1.0 до 0.0
dialog.getWindow().setAttributes(lp);
```

Метод *onCreateDialog*() вызывается один раз при создании окна. После начального создания при каждом вызове метода *showDialog*() будет срабатывать обработчик *onPrepareDialog*(). Переопределив этот метод, можно изменять диалоговое окно при каждом его выводе на экран. Это позволит привнести контекст в любое из отображаемых значений. Если требуется перед каждым вызовом диалогового окна изменять его свойства (например, текстовое сообщение или количество кнопок), то это можно реализовать внутри метода. В метод передают идентификатор диалога и сам объект *Dialog*:

```
@TargetApi(Build.VERSION_CODES.N)
@Override
public void onPrepareDialog(int id, Dialog dialog) {

 switch(id) {
  case (ID_DIALOG) :
    SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
    Date currentTime = new Date(java.lang.System.currentTimeMillis());
    String dateString = sdf.format(currentTime);
    AlertDialog timeDialog = (AlertDialog)dialog;
    timeDialog.setMessage(dateString);
    break;
    }
}
```

Если в активности должны вызываться несколько различных диалоговых окон, сначала необходимо определить целочисленный идентификатор для каждого диалога. Эти идентификаторы можно использовать в вызове метода *showDialog*() и в обработчике события *onCreateDialog*() в операторе *switch*:

```
static final int DIALOG_ID_2 = 0;
static final int DIALOG_ID_1 = 1;

protected Dialog onCreateDialog(int id) {

    Dialog dialog = null;
    switch(id) {
        case DIALOG_ID_1:
            // Код для работы
            break;
        case DIALOG_ID_2:
            // Код для работы с диалогом
            break;
        default:
            dialog = null;
    }
    return dialog;
}
```

7.2.2 Класс AlertDialog

Диалоговое окно *AlertDialog* является расширением класса *Dialog*, и это наиболее используемый класс диалоговых окон. В создаваемых окнах можно задавать элементы: заголовок; текстовое сообщение; кнопки (от одной до трех); список; флажки; переключатели.

Рассмотрим пример создания *AlertDialog*. Сначала создадим ресурс:

```
    <string name="app_name">AlertDialogSimple</string>
    <string name="exit">Выход</string>
    <string name="save_data">Сохранить данные?</string>
    <string name="yes">Да</string>
    <string name="no">Нет</string>
    <string name="cancel">Отмена</string>
    <string name="saved">Сохранено</string>
```

Так как в приложении может использоваться несколько видов диалоговых окон, то определим отдельный идентификатор *DIALOG_ ID_1*. Затем создадим объект класса *AlertDialog.Builder*, передав в качестве параметра контекст приложения. Используя методы класса *Builder*, задаём для создаваемого диалога заголовок (метод *setTitle*()), текстовое сообщение в теле диалога (метод *setMessage*()), значок (метод *setIcon*()), а также кнопки через методы. Для отображения окна вызывается метод *show*():

```
public class MainActivity extends Activity {

    final int DIALOG_ ID_1 = 1;
```

```
  public void onclick(View v) {
    // вызываем диалог
    showDialog(DIALOG_ID_1);
}

  protected Dialog onCreateDialog(int id) {
    if (id == DIALOG_ID_1) {
        AlertDialog.Builder al = new AlertDialog.Builder(this);
        // заголовок
        al.setTitle(R.string.exit);
        al.setMessage(R.string.save_data);
        al.setIcon(android.R.drawable.ic_dialog_info);
        al.setPositiveButton(R.string.yes, myClickListener);
        al.setNegativeButton(R.string.no, myClickListener);
        al.setNeutralButton(R.string.cancel, myClickListener);
  return al.create();
    }

    return super.onCreateDialog(id);
}
```

На экране отобразится следующий диалог (рис. 7.13):

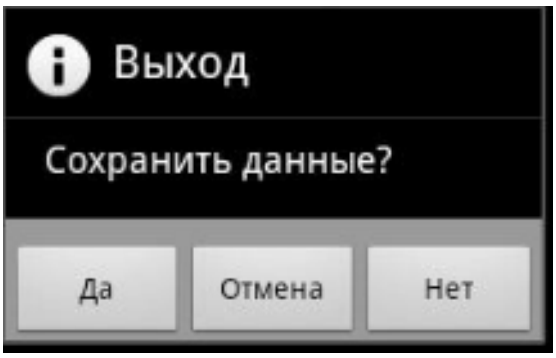


Рис.7.13 Отображение диалога, созданного на основе *AlerDialog*

Обработка нажатия кнопки определяется через параметр метода:

```
OnClickListener myClickListener = new OnClickListener() {

    public void onClick(DialogInterface dialog, int which) {

        switch (which) {
            // положительная кнопка
            case Dialog.BUTTON_POSITIVE:
              break;
            // негативная кнопка
            case Dialog.BUTTON_NEGATIVE:
              break;
            // нейтральная кнопка
            case Dialog.BUTTON_NEUTRAL:
                    break;
        }
    }
};
```

В *AlertDialog* можно добавить только по одной кнопке каждого типа: *Positive*, *Neutral* и *Negative*, т. е. максимально возможное количество кнопок в диалоге – три.

Диалоговое окно *AlertDialog* очень гибкое в настройках.

Чтобы диалоговые окна сохраняли своё состояние, рекомендуется использовать методы активности *onCreateDialog*() и *onPrepareDialog*().

7.2.3 Класс DialogFragment

DialogFragment – отображается как диалог и имеет соответствующие методы. Построить диалог можно двумя способами: используя свой *layout*-файл и через *AlertDialog.Builder*.

Сначала создаем класс, например *DialogF.java* (рис.7.14)

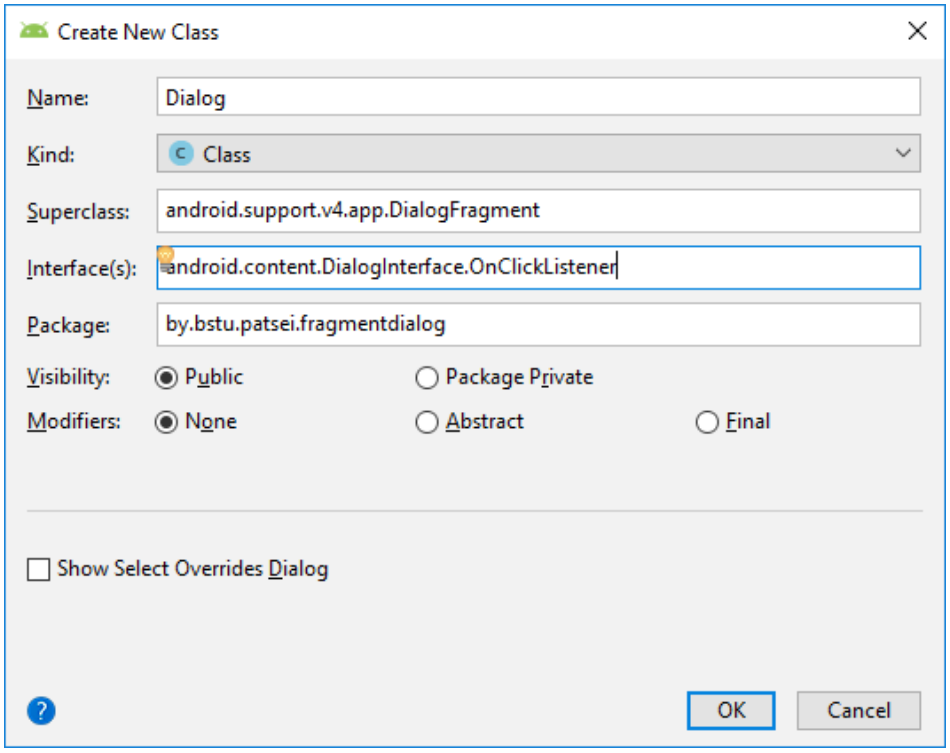


Рис.7.14 Добавление класса диалога *DialogFragment*

Для создания диалога с помощью билдера используется метод обратного вызова *onCreateDialog*. Диалог будет иметь заголовок, сообщение и три кнопки. Обработчиком для кнопок назначаем текущий фрагмент. В *onclick* выводится соответствующий текст в лог. Диалог сам закроется по нажатию на кнопку, а метод *dismiss* здесь не нужен (методы *onDismiss* и *onCancel* – это закрытие и отмена диалога):

```
public class  DialogF  extends DialogFragment implements DialogInterface.OnClickListener
{

      final String LOG_TAG = "myLogs";

       public Dialog onCreateDialog(Bundle savedInstanceState) {
            AlertDialog.Builder adb = new AlertDialog.Builder(getActivity())
           .setTitle("Title")
            .setPositiveButton(R.string.yes, this)
           .setNegativeButton(R.string.no, this)
           .setNeutralButton(R.string.maybe, this)
           .setMessage(R.string.message_text);
            return adb.create();
        }
        public void onClick(DialogInterface dialog, int which) {
            int i = 0;
```

```
            switch (which) {
                case Dialog.BUTTON_POSITIVE:
                    i = R.string.yes;
                    break;
                case Dialog.BUTTON_NEGATIVE:
                    i = R.string.no;
                    break;
                case Dialog.BUTTON_NEUTRAL:
                    i = R.string.maybe;
                    break;
            }
            if (i > 0)
             Log.d(LOG_TAG, "Dialog Fragment: " + getResources().getString(i));
        }

        public void onDismiss(DialogInterface dialog) {
            super.onDismiss(dialog);
            Log.d(LOG_TAG, "Dialog Fragment: onDismiss");
        }

        public void onCancel(DialogInterface dialog) {
            super.onCancel(dialog);
            Log.d(LOG_TAG, "Dialog Fragment: onCancel");
        }
    }
```

Определяем класс *MainActivity.java*, в которой создаем диалог и запускаем методом *show*, который на вход требует *FragmentManager* и строку-тэг:

```
public class MainActivity extends AppCompatActivity {

    DialogFragment dlg;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dlg = new DialogF();
    }

    public void onclick(View v) {
          dlg.show(getSupportFragmentManager(),"dlg");
    }
}
```

На экране должно появится следующее (рис.7.15):

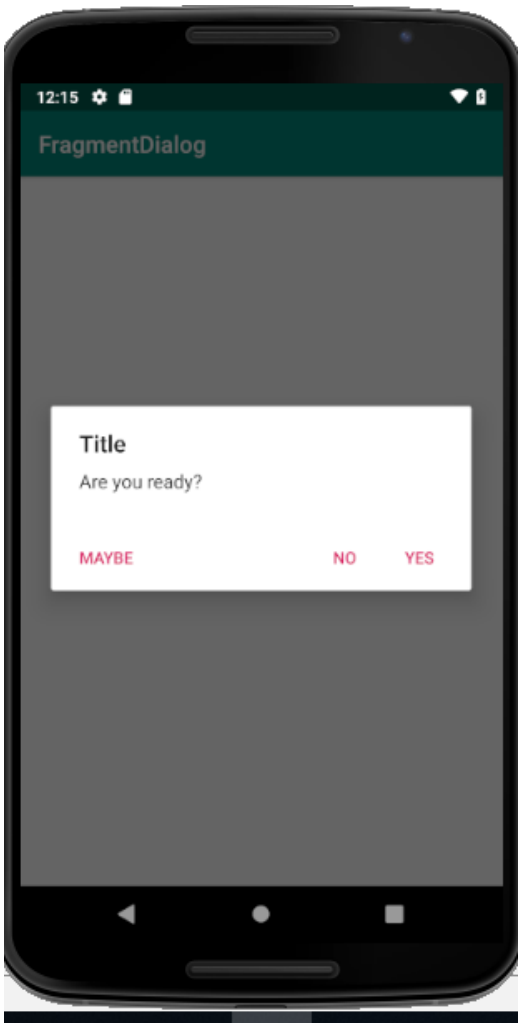


Рис.9.15 Отображение диалога *DialogFragment*

Есть еще один вариант вызова диалога. Это метод *show*(), но на вход он уже принимает не *FragmentManager*, а *FragmentTransaction*. В этом случае система сама вызовет *commit* внутри *show*, можно предварительно поместить в созданную транзакцию какие-либо еще операции или отправить ее в *BackStack*.

Можно использовать диалог-фрагмент, как обычный фрагмент и отображать его на *Activity*, а не в виде диалога.

7.2.4 Класс TimePickerDialog

TimePickerDialog – относится к стандартным диалогам, предоставляемых системой, который позволяет указать время. Рассмотрим пример:

```
public class MainActivity extends AppCompatActivity {

    TimePickerDialog tpd;
    int myHour = 14;
    int myMinute = 35;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
 tpd = new TimePickerDialog(this, myCallBack, myHour, myMinute, true);
    }
```

```
    TimePickerDialog.OnTimeSetListener myCallBack = new
    TimePickerDialog.OnTimeSetListener() {

    public void onTimeSet(TimePicker view, int hourOfDay, int minute)
       {
            myHour = hourOfDay;
            myMinute = minute;
            Toast.makeText(
               getApplicationContext(),
              "Time is " + myHour + " hours " + myMinute + " minutes",
               Toast.LENGTH_SHORT).show();
        }

    };

    public void onclick(View v) {
                      tpd.show();
        }
}
```

Здесь создается *TimePickerDialog* на основе следующего конструктора:

```
TimePickerDialog (Context context,
                  TimePickerDialog.OnTimeSetListener callBack,
                  int hourOfDay,
                  int minute,
                  boolean is24HourView)
```

где *context* – контекст, *callback* – обработчик с интерфйесом *TimePickerDialog.OnTimeSetListener*, метод которого срабатывает при нажатии кнопки *OK* на диалоге, *hourOfDay* – час, который покажет диалог, *minute* – минута, которую покажет диалог, *is24HourView* – формат времени 24 часа (иначе AM/PM). *myCallBack* – объект, реализующий интерфейс *TimePickerDialog.OnTimeSetListener*.

В результате выполнения приложения на экране будет отображено следующее (рис.7.16):

Рис.7.16 Отображение диалога *TimePickerDialog*

7.2.5 Класс DatePickerDialog

Этот класс считается устаревшим. Создать *DatePickerDialog* можно используя конструктор:

```
DatePickerDialog (Context context,
                  DatePickerDialog.OnDateSetListener callBack,
                  int year,
                        int monthOfYear,
                        int dayOfMonth)
```

где *context* – контекст, *callBack* –обработчик с интерфейсом *DatePickerDialog.OnDateSetListener*, метод которого срабатывает при нажатии кнопки *OK* на диалоге, *year* – год, который покажет диалог, *monthOfYear* – месяц, *dayOfMonth* – день. *myCallBack* – объект, реализующий интерфейс *DatePickerDialog.OnDateSetListener*, у которого только один метод – *onDateSet*. Например, (рис. 7.17):

```
public class MainActivity extends AppCompatActivity {

    DatePickerDialog tpd;
    int myYear = 2019;
    int myMonth = 02;
    int myDay = 03;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
       tpd = new DatePickerDialog(this, myCallBack, myYear, myMonth, myDay);
    }

    DatePickerDialog.OnDateSetListener myCallBack = new
DatePickerDialog.OnDateSetListener() {

    public void onDateSet(DatePicker view, int year, int monthOfYear,
                          int dayOfMonth) {
            myYear = year;
            myMonth = monthOfYear;
            myDay = dayOfMonth;
            Toast.makeText( getApplicationContext(),
                  "Today is " + myDay + "/" + myMonth + "/" + myYear,
                   Toast.LENGTH_SHORT).show();
        }
    };

    public void onclick(View v) {
        tpd.show();     }
}
```

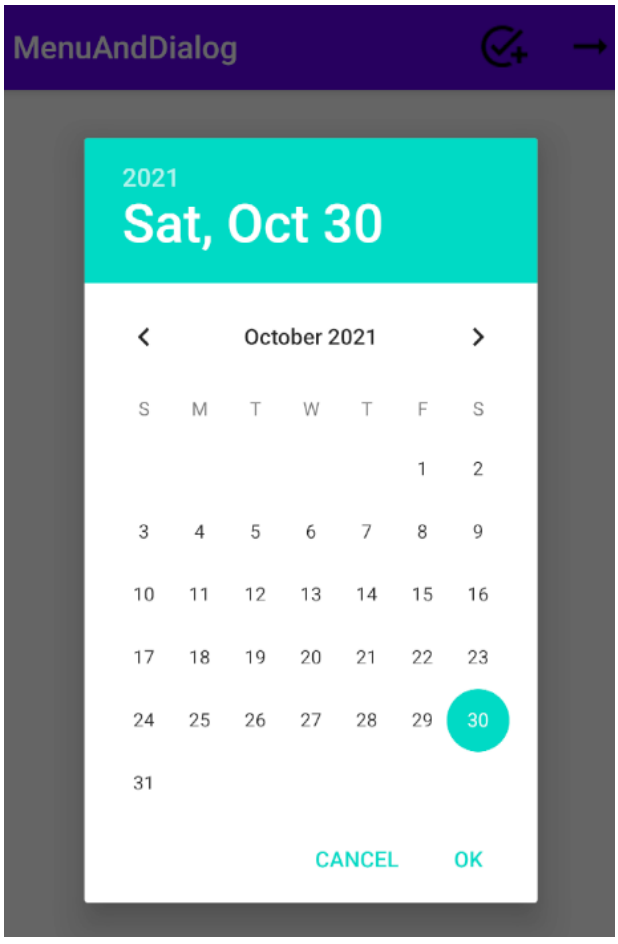


Рис.7.17 Отображение диалога *DatePickerDialog*

7.2.6 Класс ShareActionProvider

Начиная с Android 4.0 (API 14) появился класс *android.widget.ShareActionProvider* – провайдер действия передачи информации, позволяющий реализовать интерфейс передачи данных другим приложениям. Это элементы меню с подходящими приложениями для обработки данных (рис.7.18). Если пользователь выберет подходящее приложение, то его значок можно сохранить для истории, чтобы в следующий раз можно было обойтись без вызова подменю.

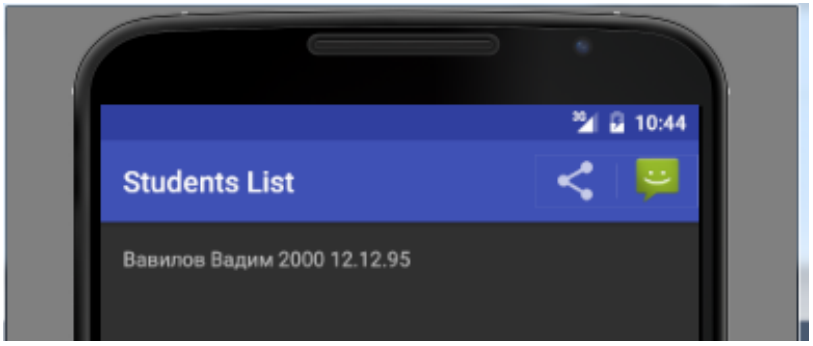


Рис.7.18 Отображение *ShareActionProvider*