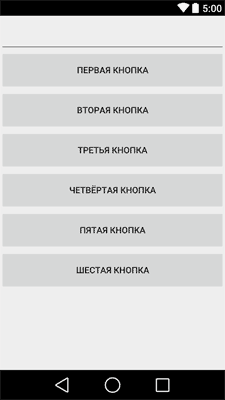
Ориентация Экрана

Вступление

Когда создавались первые портативные устройства - КПК и смартфоны, то за основу бралась настольная операционная система и допиливалась под мобильное устройство. Лишние функции удалялись, а некоторые функции добавлялись. Но при этом как-то совсем упустили из виду, что в отличие от громоздких мониторов и экранов ноутбуков, карманные устройства можно вращать в руках. Первые устройства не умели менять ориентацию экрана. Некоторые программисты самостоятельно стали создавать программы, которые умели переключаться в разные режимы. Затем эту возможность стали включать в настройки аппарата. Позже аппараты научились самостоятельно определять ориентацию экрана.

Всего существует два режима - портретный и альбомный. На большинстве телефонов используется по умолчанию портретный режим (как на паспорте). Альбомный режим знаком нам по обычным мониторам.

Рассмотрим следующий случай. Предположим, у нас в приложении имеется одно текстовое поле и шесть кнопок. Вроде всё нормально.

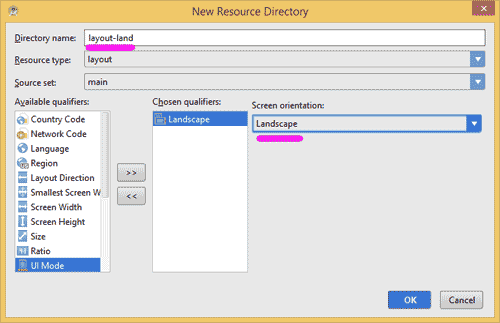


Но стоит нам повернуть устройство на 90 градусов (для эмулятора нужно нажать комбинацию клавиш Ctrl+F11), как сразу обнаруживаются проблемы. Пятая кнопка видна частично, а шестая вообще оказалась за пределами видимости. Непорядок!

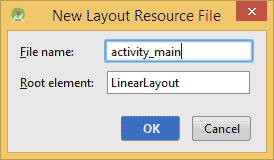


Чтобы избежать такой проблемы, необходимо как-то по другому скомпоновать кнопки. Например, расположить их не подряд друг за другом, а разбить на пары. Воспользуемся контейнером **TableLayout**. С его помощью мы можем разбить кнопки на две колонки и поместить их в три ряда.

Для этой операции нам понадобится сделать несколько важных шагов. Сначала нужно создать новую подпапку в папке **res**. Выделяем папку **res**, вызываем из него контекстное меню и последовательно выбираем команды **New | Android resource directory**. В диалоговом окне из выпадающего списка **Resource type:** выбираем **layout**. В списке **Available qualifiers:** находим элемент **Orientation** и переносим его в правую часть **Chosen qualifiers:** с помощью кнопки с двумя стрелками. По умолчанию у вас появится имя папки **layout-port** в первой строке **Directory Name:**. Но нам нужен альбомный вариант, поэтому в выпадающем списке **Screen orientation** выбираем **Landscape**. Теперь название папки будет **layout-land**.



Можно обойтись без помощи мастера, создав папку сразу через меню **New | Directory**. Этот способ годится для опытных разработчиков, которые знают, как следует назвать папку. Важно запомнить, что имя даётся не произвольно, а именно в таком виде **layout-land**. По суффиксу **-land** система понимает, что речь идет о новом режиме. Теперь нам осталось создать в созданной папке новый XML-файл **activity\_main.xml**. Вызываем контекстное меню у папки **layout-land** и выбираем команды **New | Layout Resource File**. В диалоговом окне присваиваем имя **activity\_main.xml**, которое должно совпадать с именем существующего файла. Во втором поле вводим **LinearLayout**, по мере ввода появится подсказка, облегчающая выбор.



Откроем созданный файл и модифицируем его следующим образом.

<?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">

<LinearLayout

android:orientation="vertical"

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

android:layout\_gravity="center"

android:paddingLeft="20dp"

android:paddingRight="20dp">

<EditText

android:layout\_height="wrap\_content"

android:id="@+id/editText"

android:layout\_width="match\_parent"/>

<TableLayout

android:layout\_height="wrap\_content"

android:layout\_width="wrap\_content"

android:layout\_gravity="center"

android:stretchColumns="\*">

<TableRow>

<Button

android:id="@+id/button1"

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

android:text="Первая кнопка"/>

<Button

android:id="@+id/button2"

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

android:text="Вторая кнопка"/>

</TableRow>

<TableRow>

<Button

android:id="@+id/button3"

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

android:text="Третья кнопка"/>

<Button

android:id="@+id/button4"

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

android:text="Четвёртая кнопка"/>

</TableRow>

<TableRow>

<Button

android:id="@+id/button5"

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

android:text="Пятая кнопка"/>

<Button

android:id="@+id/button6"

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

android:text="Шестая кнопка"/>

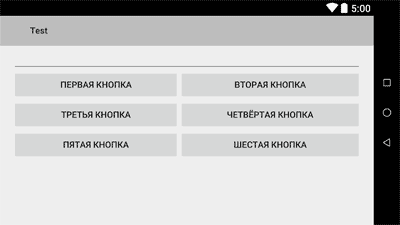
</TableRow>

</TableLayout>

</LinearLayout>

</LinearLayout>

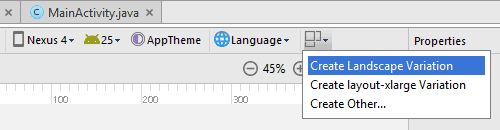
Запускаем приложение и проверяем. Отлично, теперь видны все кнопки. Поздравляю, вы гений!



Когда вы создаёте альтернативную разметку, то не забывайте включать все компоненты, к которым будете обращаться программно, иначе получите ошибку. Допустим, вы забыли добавить шестую кнопку. В портретном режиме программа будет работать, а когда пользователь перевернёт экран, то активность будет инициализировать все компоненты для работы, а кнопки-то и нет. Крах приложения и минусы в отзывах.

Студия, на помощь!

Теперь, когда вы разобрались, как создавать новый файл для альбомной ориентации, я покажу вам быстрый способ без участия мастера настройки. Переключитесь в режим дизайна вашей основной разметки. Наверху на панели инструментов выберите последний значок с выпадающим списком. Щёлкните на нём и выберите пункт **Create Landscape Variation**. Появится готовый файл в папке **res/layout-land**. Содержимое файла из портретной ориентации будет скопировано в файл и вы сможете его отредактировать по своему желанию.



Узнать ориентацию программно

Чтобы из кода узнать текущую ориентацию, можно создать следующую функцию:

private String getScreenOrientation(){

if(getResources().getConfiguration().orientation == Configuration.ORIENTATION\_PORTRAIT)

return "Портретная ориентация";

else if (getResources().getConfiguration().orientation == Configuration.ORIENTATION\_LANDSCAPE)

return "Альбомная ориентация";

else

return "";

}

Вызовите данную функцию из нужного места, например, при щелчке кнопки и узнайте текущую ориентацию. В примере использовались две распространённые системные константы для ориентации. Есть еще константа **ORIENTATION\_SQUARE** (квадратный экран). Но я таких телефонов не встречал.

Можно также вычислить ширину и высоту экрана, если высота больше ширины, то устройство в портретной ориентации, иначе - в альбомной:

private String mOrientation = "";

private boolean isLandscapeMode(Activity activity)

{

int width =

activity.getWindowManager().getDefaultDisplay().getWidth();

int height = activity.getWindowManager().getDefaultDisplay().getHeight();

boolean isLandscape = width > height;

if(isLandscape)

mOrientation = "Альбомная";

else

mOrientation = "Портретная";

return isLandscape;

}

Сейчас этот код считается устаревшим и для вычисления размера экрана используются другие методы (описано в примере Экран).

Кручу-верчу, запутать хочу!

Хорошо, мы можем определить текущую ориентацию, но в какую сторону повернули устройство? Ведь его можно повернуть влево, вправо или вообще вверх тормашками. Напишем другую функцию:

private String getRotateOrientation() {

int rotate = getWindowManager().getDefaultDisplay().getRotation();

switch (rotate) {

case Surface.ROTATION\_0:

return "Не поворачивали";

case Surface.ROTATION\_90:

return "Повернули на 90 градусов по часовой стрелке";

case Surface.ROTATION\_180:

return "Повернули на 180 градусов";

case Surface.ROTATION\_270:

return "Повернули на 90 градусов против часовой стрелки";

default:

return "Не понятно";

}

}

Раньше существовал аналогичный метод **getOrientation()**, который устарел. Используйте **getRotation()**

Установить ориентацию программно и через манифест

Если вы большой оригинал и хотите запустить приложение в стиле "вид сбоку", то можете сделать это программно. Разместите код в методе **onCreate()**:

import android.content.pm.ActivityInfo;

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_LANDSCAPE);

Учтите, что в этом случае котам не очень удобно будет пользоваться вашим приложением.

Вы можете запретить приложению менять ориентацию, если добавите нужный код в **onCreate()**.

setRequestedOrientation (ActivityInfo.SCREEN\_ORIENTATION\_LANDSCAPE); //для альбомного режима

// или

setRequestedOrientation (ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT); //для портретного режима

Но указанный способ не совсем желателен. Лучше установить нужную ориентацию через манифест, прописав в элементе <activity> параметр **android:screenOrientation**:

android:screenOrientation="portrait"

android:screenOrientation="landscape"

Кстати, существует ещё один вариант, когда устройство полагается на показания сенсора и некоторые другие:

android:screenOrientation="sensor"

В Android 4.3 (API 18) появились новые значения (оставлю пока без перевода):

* **userLandscape** - Behaves the same as "sensorLandscape", except if the user disables auto-rotate then it locks in the normal landscape orientation and will not flip.
* **userPortrait** - Behaves the same as "sensorPortrait", except if the user disables auto-rotate then it locks in the normal portrait orientation and will not flip.
* **fullUser** - Behaves the same as "fullSensor" and allows rotation in all four directions, except if the user disables auto-rotate then it locks in the user's preferred orientation.
* **locked** - to lock your app's orientation into the screen's current orientation.

После появления Android 5.0 зашёл на страницу документации и пришёл в ужас. Там появились новые значения.

android:screenOrientation=["unspecified" | "behind" |

"landscape" | "portrait" |

"reverseLandscape" | "reversePortrait" |

"sensorLandscape" | "sensorPortrait" |

"userLandscape" | "userPortrait" |

"sensor" | "fullSensor" | "nosensor" |

"user" | "fullUser" | "locked"]

Запрет на создание новой активности

На примере программной установки ориентации можно увидеть интересный эффект, о котором нужно помнить. Предположим у нас есть кнопка, позволяющая менять ориентацию. Заодно будем менять текст на кнопке, чтобы операция соответствовала надписи.

public class MainActivity extends Activity {

private Button mButton;

static final String ORIENTATION\_PORTRAIT = "Портретный режим";

static final String ORIENTATION\_LANDSCAPE = "Альбомный режим";

// определяем изменение ориентации экрана

boolean mState = false;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mButton = (Button) findViewById(R.id.button);

// установим текст по умолчанию

mButton.setText(ORIENTATION\_LANDSCAPE);

}

public void onClick(View view) {

// state FALSE: переключаемся на LANDSCAPE

if (!mState) {

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_LANDSCAPE);

mButton.setText(ORIENTATION\_PORTRAIT);

}

// state TRUE: переключаемся на PORTRAIT

else {

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);

mButton.setText(ORIENTATION\_LANDSCAPE);

}

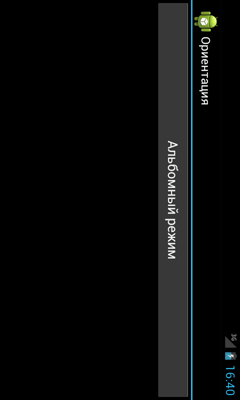
// обновляем state на противоположное значение

mState = !mState;

}

}

Теперь посмотрите, что у нас получилось. Запустите проект и нажмите на кнопку. Ориентация экрана поменялась, однако текст на кнопке остался прежним, хотя по нашей задумке он должен измениться.



Нажмём на кнопку ещё раз. Надпись изменится, но ориентация не сменится. И только повторный щелчок повернёт экран в обратную сторону.

По умолчанию, при смене ориентации Android уничтожает и пересоздаёт активность из кода, что подразумевает повторный вызов метода **onCreate()**. Поэтому при повороте активность устанавливала текст, определенный в **onCreate()**. В большинстве случаев это не мешает программе. Но если приложение воспроизводит видео, то при смене ориентации вызов **onCreate()** может привести к повторному началу воспроизведения (если так написан пример).

Чтобы активность не пересоздавалась, добавьте в манифест строчку для нужной активности:

android:configChanges="keyboardHidden|orientation|screenSize"

Во многих примерах, которые я видел, используется пара *keyboardHidden|orientation*, но, похоже, в Android 4 этого недостаточно и нужно добавить еще один атрибут **screenSize**

В этом случае система вызовет метод **onConfigurationChanged(Configuration)** и полагается на вас:

@Override

public void onConfigurationChanged(Configuration newConfig) {

super.onConfigurationChanged(newConfig);

// Проверяем ориентацию экрана

if (newConfig.orientation == Configuration.ORIENTATION\_LANDSCAPE) {

Toast.makeText(this, "landscape", Toast.LENGTH\_SHORT).show();

} else if (newConfig.orientation == Configuration.ORIENTATION\_PORTRAIT) {

Toast.makeText(this, "portrait", Toast.LENGTH\_SHORT).show();

}

}

В документации говорится, что данный способ следует избегать.

Исчезающий текст

Как уже говорилось, при смене ориентации активность пересоздаётся. При этом можно наблюдать интересный эффект с пропадающим текстом. Чтобы увидеть эффект, создадим два текстовых поля. Одному из них присвоим идентификатор, а другое поле оставим без него.

<EditText

android:id="@+id/editTest"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

<EditText

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

Запустите приложение, введите любой текст в обоих полях и смените ориентацию. Вы увидите, что у поля с идентификатором текст при повороте сохранится, а у поля без идентификатора текст исчезнет. Учитывайте данное обстоятельство.

К вышесказанному могу добавить, что при смене ориентации у поля с идентификатором вызывается метод **onTextChanged()**:

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

EditText editText = (EditText)findViewById(R.id.editTextTest);

editText.addTextChangedListener(new TextWatcher() {

@Override

public void onTextChanged(CharSequence s, int start, int before, int count) {

Toast.makeText(MainActivity.this,

"onTextChanged: " + s, Toast.LENGTH\_SHORT).show();

}

@Override

public void beforeTextChanged(CharSequence s, int start, int count, int after) {

}

@Override

public void afterTextChanged(Editable s) {

}

});

}

Проверка на существование

Если вы используете две разные разметки, то возможна ситуация, когда в альбомной ориентации используется кнопка, которой нет в портретной ориентации. Это можете привести к ошибке в коде, поэтому нужно проверить существование кнопки:

Button landscapeButton = (Button) findViewById(R.id.landscapeButton);

if (landscapeButton != null) {

// Можно работать

}

На практике такое встречается редко, но помните на всякий случай.

Запоминаем значения переменных

С поворотом экрана возникает одна очень неприятная проблема. Вдумайтесь в значение слов, что при повороте экрана активность создаётся заново. Чтобы было понятно, нужно вернуться к проекту, в котором мы считали ворон. Если вы его удалили, то придётся пройти урок заново и восстановить его.

Щёлкните несколько раз по кнопке. Допустим на экране красуется надпись "Я насчитал 5 ворон". Поворачиваем экран - куда улетели все вороны? На экране появилась надпись, что ... Впрочем, я не буду говорить вам, сами посмотрите.

А что собственно произошло? Я же вас предупреждал, что активность при повороте создаётся заново. А значит переменная **mCount** снова принимает значение 0, т.е сбрасывается в начальное значение.

Что же делать? Для этих целей у активности существует специальный метод **onSaveInstanceState()**, который вызывается системой перед методами **onPause()**, **onStop()** и **onDestroy()**. Метод позволяет сохранить значения простых типов в объекте **Bundle**. Класс **Bundle** - это простой способ хранения данных ключ/значение.

Создадим ключ с именем **KEY\_COUNT**. В Android Studio c версии 1.5 появились живые шаблоны, позволяющие быстро создать ключ. Введите до метода **onCreate()** строчными буквами слово **key**, во время набора появится подсказка. Нажимаем **Enter** и получаем заготовку. После символа подчёркивания вводим название ключа. В результате получим ключ следующего вида.

private static final String KEY\_COUNT = "COUNT";

Далее создаём метод **onSaveInstanceState()** после метода **onCreate()**. Во время набора имени метода подсказка покажет, что имеется два метода. Выбирайте метод с одним параметров (обычно он идёт вторым). Записываем в ключа значение счётчика.

@Override

protected void onSaveInstanceState(Bundle outState) {

super.onSaveInstanceState(outState);

outState.putInt(KEY\_COUNT, mCount);

}

А в методе **onCreate()** делаем небольшую проверку.

@Override

protected void onCreate(**Bundle** savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mInfoTextView = (TextView) findViewById(R.id.textView);

if (savedInstanceState != null) {

mCount = savedInstanceState.getInt(KEY\_COUNT, 0);

mInfoTextView.setText("Я насчитал " + mCount + " ворон");

}

}

У метода в параметре содержится объект **Bundle**. Только здесь он назван **savedInstanceState** вместо **outState**, но пусть вас это не вводит заблуждение. Имена вы можете придумывать сами. Главное, что объект содержит сохранённое значение переменной при повороте. При первом запуске приложения объект не существует (**null**), а потом мы его создали своим кодом. Для этого и нужна проверка. Обратите внимание, что здесь мы не прибавляем единицу к счётчику, как у кнопки. Если скопировать код у кнопки, то получится, что счётчик будет увеличиваться самостоятельно при поворотах без нажатия на кнопку. Прикольно, конечно, но может ввести в заблуждение пользователя. Хотя, если вы пишите приложение "Я твой дом труба шатал", то такой способ может пригодиться для подсчёта, сколько раз вы вертели телефон, чтобы разрушить чей-то дом.

Обращаю ваше внимание, что данный способ используется для сохранения промежуточных результатов во время действия программы. В следующих уроках вы узнаете, как можно сохранять результат между запусками приложения.

Ориентация у фрагментов

Позже вы узнаете о существовании фрагментов. Может возникнуть такая ситуация, когда вы захотите выводить конкретный фрагмент в нужной ориентации. У фрагментов есть собственный жизненный цикл, и вы можете реализовать свой код в методах фрагмента:

@Override

public void onResume() {

getActivity().setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_FULL\_SENSOR);

}

@Override

public void onPause() {

getActivity().setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT);

// ваш код

super.onPause();

}

Я с таким случаем не встречался, но оставлю как памятку.

Жизненный цикл при повороте

При повороте активность проходит через цепочку различных состояний. Порядок следующий.

1. onPause()
2. onStop()
3. onDestroy()
4. onCreate()
5. onStart()
6. onResume()