Освой программирование играючи
Сайт Александра Климова

(http://developer.alexanderklimov.ru/)

/ Моя кошка замечательно разбирается в программировании. Стоит мне объяснить проблему ей - и все становится ясно. */*
John Robbins, Debugging Applications, Microsoft Press, 2000

(http://feeds.feedburner.com/alexanderklimov/VJcl)

- Android (http://developer.alexanderklimov.ru/android)
- C#/Visual Basic (http://developer.alexanderklimov.ru/dotnet/)
- Windows Phone (http://developer.alexanderklimov.ru/windowsphone/wp.php)
- WPF (http://developer.alexanderklimov.ru/wpf/wpf.php)
- PHP (http://developer.alexanderklimov.ru/php)
- Arduino (http://developer.alexanderklimov.ru/arduino)

Главная (http://developer.alexanderklimov.ru/android/index.php)
Теория (http://developer.alexanderklimov.ru/android/theory/)
Palette (http://developer.alexanderklimov.ru/android/views.php)
ListView (http://developer.alexanderklimov.ru/android/listview/)
Котошоп (http://developer.alexanderklimov.ru/android/catshop/)
Анимация (http://developer.alexanderklimov.ru/android/animation/)
SQLite (http://developer.alexanderklimov.ru/android/sqlite/)
OpenGL ES (http://developer.alexanderklimov.ru/android/opengles/)
Библиотеки (http://developer.alexanderklimov.ru/android/library/)
Игры (http://developer.alexanderklimov.ru/android/games/)
Wear (http://developer.alexanderklimov.ru/android/wear/)
Эмулятор (http://developer.alexanderklimov.ru/android/emulator/)
Советы (http://developer.alexanderklimov.ru/android/tips-android.php)
Статьи (http://developer.alexanderklimov.ru/android/articles-android.php)
Книги (http://developer.alexanderklimov.ru/android/books.php)

Java. Экспресс-курс (http://developer.alexanderklimov.ru/android/java/java.php)

Дизайн (http://developer.alexanderklimov.ru/android/design/)

Отладка (http://developer.alexanderklimov.ru/android/debug/)

Open Source (http://developer.alexanderklimov.ru/android/opensource.php)

Полезные ресурсы (http://developer.alexanderklimov.ru/android/links.php)

Класс AsyncTask

Создание новой асинхронной задачи

Класс **AsyncTask** предлагает простой и удобный механизм для перемещения трудоёмких операций в фоновый поток. Благодаря ему вы получаете удобство синхронизации обработчиков событий с графическим потоком, что позволяет обновлять элементы пользовательского интерфейса для отчета о ходе выполнения задачи или для вывода результатов, когда задача завершена.

AsyncTask создает, синхронизирует потоки, а также управляет ими, что позволяет создавать асинхронные задачи, состоящие из операций, выполняющихся в фоновом режиме, и обновлять пользовательский интерфейс по их завершении.

Напрямую с классом **AsyncTask** нельзя, вам нужно наследоваться от него (extends). Ваша реализация должна предусматривать классы для объектов, которые будут переданы в качестве параметров методу execute(), для переменных, что станут использоваться для оповещения о ходе выполнения, а также для переменных, где будет храниться результат. Формат такой записи следующий:

```
AsyncTask<[Input Parameter Type], [Progress Report Type], [Result Type]>
```

Если не нужно принимать параметры, обновлять информацию о ходе выполнения или выводить конечный результат, просто укажите тип **Void** во всех трёх случаях.

Каркас реализации AsyncTask, в котором используются строковой параметр и два целочисленных значения, нужных для оповещения о выполнении работы и о конечном результате:

```
private class MyAsyncTask extends AsyncTask<String, Integer, Integer> {
    @Override
    protected void onProgressUpdate(Integer... progress) {
        // [... Обновите индикатор хода выполнения, уведомления или другой
        // элемент пользовательского интерфейса ...]
    }

        @Override
    protected void onPostExecute(Integer... result) {
        // [... Сообщите о результате через обновление пользовательского
        // интерфейса, диалоговое окно или уведомление ...]
    }

   @Override
    protected Integer doInBackground(String... parameter) {
        int myProgress = 0;
        // [... Выполните задачу в фоновом режиме, обновите переменную myProgress...]
        publishProgress(myProgress);
        // [... Продолжение выполнения фоновой задачи ...]
        // Верните значение, ранее переданное в метод onPostExecute
        return result;
    }
}
```

У **AsyncTask** есть несколько основных методов, которые нужно освоить в первую очередь. Обязательным является метод **doInBackground()**, остальные опциональны, но используются часто.

- **doInBackground()** – основной метод, который выполняется в новом потоке. **Не имеет доступа к UI**. Именно в этом методе должен находиться код для тяжелых задач. Принимает набор параметров тех типов, которые определены в реализации вашего класса. Этот метод выполняется в фоновом потоке, поэтому в нем не должно быть никакого взаимодействия с элементами пользовательского интерфейса. Размещайте здесь трудоёмкий код, используя метод publishProgress(), который позволит обработчику onProgressUpdate() передавать изменения в пользовательский интерфейс. Когда фоновая задача завершена, данный метод возвращает конечный результат для обработчика onPostExecute(), который сообщит о нём в поток пользовательского интерфейса.
- **onPreExecute()** – выполняется перед doInBackground(). **Имеет доступ к UI**
- **onPostExecute()** – выполняется после doInBackground() (может не вызываться, если AsyncTask был отменен). **Имеет доступ к UI**. Используйте его для обновления пользовательского интерфейса, как только ваша фоновая задача завершена. Данный обработчик при вызове синхронизируется с потоком GUI, поэтому внутри него вы можете безопасно изменять элементы пользовательского интерфейса.
- **onProgressUpdate**. **Имеет доступ к UI**. Переопределите этот обработчик для публикации промежуточных обновлений в пользовательский интерфейс. При вызове он синхронизируется с потоком GUI, поэтому в нём вы можете безопасно изменять элементы пользовательского интерфейса.

Вкратце о том, что значит имеет/не имеет доступ к UI. Все ваши кнопки, текстовые метки, ImageView (всё, что отображается на экране) являются часть пользовательского интерфейса - User Interface (UI). Ваша задача - не допустить, чтобы в методе **doInBackground()** было обращение к какому-

нибудь элементу. Например, нельзя установить текст в TextView через метод **setText()** или поменять цвет шрифта в EditText. В примерах вы увидите, как нужно делать подобные вещи.

Простой пример для знакомства. Кот полез на крышу

Напишем простой пример с использованием названных методов. Предположим, мы хотим описать процесс восхождения котов на крышу. Всегда оставалось загадкой, как они умудряются залезать на крышу, но, очевидно, что это очень сложная работа, которую удобно использовать в отдельном потоке.

▶ Подробности под ~~котом~~ катом

Выведем на экран слово **Полез на крышу** в методе **onPreExecute()**, эмулируем тяжёлый код в методе **doInBackground()**, выведем на экран слово **Залез** в методе **onPostExecute()**.

Создадим новый проект и добавим на экран кнопку, индикатор прогресса и текстовую метку:

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

    <Button
        android:id="@+id/buttonStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onclick"
        android:text="Поехали" >
    </Button>

    <ProgressBar
        android:id="@+id/progress"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >
    </ProgressBar>

    <TextView
        android:id="@+id/tvInfo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="" >
    </TextView>

</LinearLayout>
```

При щелчке на кнопке должна запуститься тяжёлая задача. Это может быть загрузка файла из сети, обработка большого изображения, сохранение больших данных в файл или в базу данных. В нашем случае - кот полез на крышу. В TextView будем выводить текущую информацию. Компонент ProgressBar будет показывать нам, что приложение не зависло во время выполнения задачи.

Сначала напишем код, а потом будет объяснение к нему.

```
// Если этот код работает, его написал Александр Климов,
// а если нет, то не знаю, кто его писал.
package ru.alexanderklimov.asynctaskdemo;

import ...

public class MainActivity extends Activity {

        CatTask cattask;
        TextView tvInfo;

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

                tvInfo = (TextView) findViewById(R.id.tvInfo);
        }

        public void onclick(View v) {
                cattask = new CatTask();
                cattask.execute();
        }

        class CatTask extends AsyncTask<Void, Void, Void> {

                @Override
                protected void onPreExecute() {
                        super.onPreExecute();
                        tvInfo.setText("Полез на крышу");
                }

                @Override
                protected Void doInBackground(Void... params) {
                        try {
                                TimeUnit.SECONDS.sleep(5);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                        return null;
                }

                @Override
```

```
                protected void onPostExecute(Void result) {
                        super.onPostExecute(result);
                        tvInfo.setText("Залез");
                }
        }
}
```

Запустите проект и нажмите на кнопку. Сначала появится текст "Полез на крышу", который через 5 секунд сменится надписью "Залез". Индикаторе прогресса при этом будет постоянно крутиться.

У кота одна задача - залезть на крышу. Поэтому мы создали новую задачу **CatTask**, которая наследуется от AsyncTask. Для первого примера мы пока использовали **Void**, чтобы пока не связываться с параметрами.

В методе **onPreExecute()** мы устанавливаем начальный текст перед выполнением задачи.

В методе **doInBackground()** идёт имитация тяжёлой работы. Напоминаю, что здесь нельзя писать код, связанный с пользовательским интерфейсом. Ради интереса поместите в методе строчку:

```
tvInfo.setText("Лезу на крышу. Чуть-чуть осталось");
```

В этом случае приложение запустится, но при выполнении задачи завершится с ошибкой. Оно вам надо?

В методе **onPostExecute()** мы выводим сообщение, которое появится после выполнения задачи.

Обратите внимание, что если мы нажмем на кнопку, пока работает AsyncTask, то создастся и запустится новая задача поверх старой. Получается, что две задачи будут одновременно работать с экраном активности. Нужно избегать подобных ситуаций. Позже я покажу, как это сделать.

На данный момент для избежания конфликтов можно скрыть кнопку в методе **onPreExecute()** и показать её снова в **onPostExecute()**:

```
buttonStart.setVisibility(View.INVISIBLE); // прячем кнопку
buttonStart.setVisibility(View.VISIBLE);  // показываем кнопку
```

Сама задача CatTask (объект AsyncTask) создаётся в UI-потоке. Также в UI-потоке вызывается его метод **execute()**

> Каждый экземпляр класса AsyncTask может быть запущен всего один раз. Попытка повторного вызова метода execute() приведет к выбросу исключения.

Кстати, коты прислали фотографию, которая объясняет, зачем они лезут на крышу. Интересно, откуда они узнали про эту статью?

Использование параметров

Для первого примера мы не использовали параметры. Наша задача была показать сообщения до и после выполнения задачи. Но часто нам необходимо знать промежуточные результаты выполняемой задачи. Например, мы хотим знать, что кот сейчас на втором этаже и ему осталось ещё двенадцать (бедный кот).

Когда мы создавали свой класс **CatTask**, то использовали угловые скобки, в которых необходимо указать три типа данных:

1. Тип входных данных
2. Тип промежуточных данных, которые используются для вывода промежуточных результатов
3. Тип возвращаемых данных

В первом примере мы указали <Void, Void, Void>. Эта запись означает, что мы не будем использовать параметры.

Во втором примере попробуем воспользоваться ими.

Используем второй параметр - промежуточные данные

Начнём со второго параметра, который отвечает за промежуточные данные. Итак, нам нужно знать текущий этаж, на котором находится кот. Если дом 14-этажный, то у нас должны быть значения от 1 до 14 типа *Integer*:

```
class CatTask extends AsyncTask<Void, Integer, Void>
```

Метод **onPreExecute()** оставляем без изменений, здесь мы просто выводим сообщение о начале штурма здания.

Переходим к методу **doInBackground()**. Мы пока не используем входящие данные, поэтому пока здесь используется **Void**. В самом методе создаётся цикл от 0 до 14 и при каждом проходе цикла увеличивается счётчик *counter* на единицу.

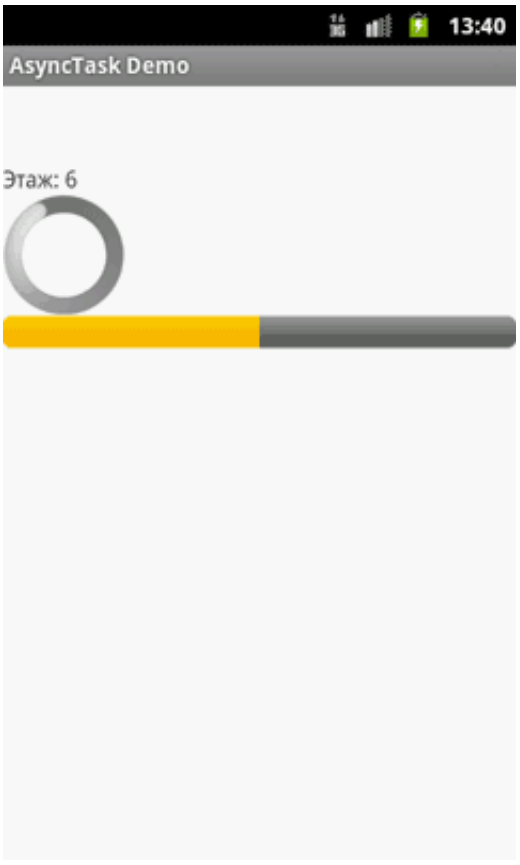
В методе **doInBackground()** вызываются два метода. Первый метод **getFloor()** - наш. Здесь мы реализуем свою логику тяжёлой работы. Воспринимайте его как эмуляцию загрузки файлов или другую сложную работу. В нашем случае при покорении очередного этажа кот должен пометить его, нацарапать на стене неприличное слово из трёх букв типа МЯУ или МУР. В вашем приложении возможно это будет код обработки большой картинки, скачанной по сети. Пока у нас здесь просто пауза на одну секунду.

Второй метод **publishProgress()** - системный метод. Когда мы в методе **doInBackground()** вызываем метод **publishProgress()** и передаём туда данные, то срабатывает метод **onProgressUpdate()**, который получает эти данные. Тип принимаемых данных равен второму типу из угловых скобок, у нас это Integer. Метод **onProgressUpdate()** используется для вывода промежуточных результатов и имеет доступ к UI. Таким образом, метод **publishProgress()** является своеобразным мостиком для передачи данных из **doInBackground()** в **onProgressUpdate()**. Мы передаём значение счётчика, которое выводится в текствой метке.

```
@Override
protected void onProgressUpdate(Integer... values) {
        super.onProgressUpdate(values);
        tvInfo.setText("Этаж: " + values[0]);
}
```

При запуске проекта, вы увидите, как в текстовом поле будут меняться числа от 0 до 14.

Для наглядности можно добавить на экран горизонтальный ProgressBar, который будет показывать в удобном виде график прохождения этажей.



Полностью код будет следующим:

```
// Если этот код работает, его написал Александр Климов,
```

```
// а если нет, то не знаю, кто его писал.
package ru.alexanderklimov.asynctaskdemo;

import ...

public class MainActivity extends Activity {

        CatTask cattask;
        TextView tvInfo;
        ProgressBar progress;
        Button buttonStart;
        ProgressBar horizontalprogress;

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

                progress = (ProgressBar) findViewById(R.id.progress);
                tvInfo = (TextView) findViewById(R.id.tvInfo);
                buttonStart = (Button) findViewById(R.id.buttonStart);
                horizontalprogress = (ProgressBar) findViewById(R.id.progress2);

        }

        public void onclick(View v) {
                cattask = new CatTask();
                cattask.execute();
        }

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

        class CatTask extends AsyncTask<Void, Integer, Void> {

                @Override
                protected void onPreExecute() {
                        super.onPreExecute();
                        tvInfo.setText("Полез на крышу");
                        buttonStart.setVisibility(View.INVISIBLE);
                }

                @Override
                protected Void doInBackground(Void... params) {
                        try {
                                int counter = 0;
```

```
                        for (int i = 0; i < 14; i++) {
                                getFloor(counter);
                                publishProgress(++counter);
                        }
                        // разъединяемся
                        TimeUnit.SECONDS.sleep(1);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                return null;
        }

        @Override
        protected void onProgressUpdate(Integer... values) {
                super.onProgressUpdate(values);
                tvInfo.setText("Этаж: " + values[0]);
                horizontalprogress.setProgress(values[0]);
        }

        @Override
        protected void onPostExecute(Void result) {
                super.onPostExecute(result);
                tvInfo.setText("Залез");
                buttonStart.setVisibility(View.VISIBLE);
                horizontalprogress.setProgress(0);
        }

        private void getFloor(int floor) throws InterruptedException {
                TimeUnit.SECONDS.sleep(1);
        }
    }
}
```

Используем первый параметр - входящие данные

Теперь попробуем задействовать первый параметр, который отвечает за входящие данные. На практике это может быть список адресов, с которых надо загрузить картинки или что-то в этом роде.

Изменим объявление метода **doInBackground()**:

```
protected Void doInBackground(String... urls)
```

Теперь у метода есть входные параметры типа *String*. Сразу поменяем первый параметр в классе CatTask:

```
class CatTask extends AsyncTask<String, Integer, Void> {
}
```

Теперь нужно передать в метод **execute()** список адресов файлов для загрузки. Метод **doInBackground** принимает эти данные и в цикле по одному загружает эти файлы. Дальше без изменений. После прохождения каждого этажа (загрузки каждого файла) вызывается метод **publishProgress()**, в который передаются данные.

В обработчике кнопки вызовем метод **execute()**, которому передадим набор строк, так как мы указали этот тип в угловых скобках на первом месте.

```
cattask.execute("cat1.jpg", "cat2.jgp", "cat3.jpg", "cat4.jpg");
```

Полностью код будет следующим:

▶ Показать код (щелкните мышкой)

Ещё раз закрепим материал.

Метод **execute()** вызывается в основном потоке, чтобы начать выполнение задачи. В него можно передать набор данных определенного типа. Если что-то передаём, то этот тип будет указан первым в угловых скобках при создании наследника AsyncTask.

Методы **onPreExecute()** и **onPostExecute()** вызываются системой в начале и конце выполнения задачи.

Основной метод для тяжёлой работы - **doInBackground()**. Можно передать методу какие-то данные. В этом случае смотрим на **execute()**.

Метод **publishProgress()** нужен в том случае, когда требуется обрабатывать промежуточные данные. Его нужно явно вызвать в методах doInBackground(), onPreExecute() или onPostExecute(). На вход передаем промежуточные данные определенного типа. Этот тип указан вторым в угловых скобках при описании AsyncTask.

Метод **onProgressUpdate()** получает на вход промежуточные результаты от метода **publishProgress()**. Так как метод **onProgressUpdate()** принимает на вход набор параметров, то при передаче одного значения от **publishProgress** нужно взять первый элемент массива.

Используем третий параметр - возвращаемые данные

Осталось рассмотреть вариант использования третьего параметра.

В третьем параметре указывается тип объекта, который должен вернуться из AsyncTask. Получить его можно двумя способами:

- Он передаётся методу **onPostExecute()**, который вызывается по окончании задачи
- Существует метод **get()**, способный возвращать нужный объект

Если мы собираемся что-то возвращать, то это надо указать в возвращаемом значении у метода **doInBackground()** и в входящем параметре для метода **onPostExecute()**.

Начнём с метода **doInBackground()** и изменим его описание следующим образом:

```
@Override
protected Integer doInBackground(String... urls) {
    // здесь без изменений
        return 2012;
}
```

Как видите, наш метод теперь возвращает тип Integer. Пусть это будет число 2012 - напоминание про обман племени майя, которые обещали в этом году конец света. Даже коты ждали его.

Eclipse подчеркнёт исправленный вариант, указывая не соответствие в параметрах у класса AsyncTask. Исправим:

```
class CatTask extends AsyncTask<String, Integer, Integer>
```

Теперь обратимся к методу **onPostExecute()**. Тип Integer должен стать входящим параметром. Поменяем объявление метода:

```
protected void onPostExecute(Integer result)
```

В нашем случае входящий параметр не несёт смысловой нагрузки, просто выведем значение в текстовом поле.

Полный код для самопроверки:

▶ Показать код (щелкните мышкой)

Метод get()

Выше уже упоминалось, что метод **get()** возвращает значение задачи. Тут следует рассмотреть два момента. Первый момент - вызовем метод после выполнения задачи. Второй момент - попробуем получить значение во время выполения задачи.

Начнём с первого варианта. Добавим на экран активности ещё одну кнопку **buttonResult** с надписью "Получить результат":

```
<Button
    android:id="@+id/buttonResult"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onResultClick"
    android:text="Получить результат" />
```

Добавим код для обработчика кнопки:

```
public void onResultClick(View v) {
        if (cattask == null)
                return;
        int result = -1;
        try {
                result = cattask.get();
                Toast.makeText(this, "Полученный результат: " + result, Toast.LENGTH_LONG)
                                .show();
        } catch (InterruptedException e) {
                e.printStackTrace();
        } catch (ExecutionException e) {
                e.printStackTrace();
        }
}
```

Запустите приложение, далее запустите задачу нажатием первой кнопки, дождитесь окончания задачи и нажмите вторую кнопку. Во всплывающем сообщение вы увидите то же число "2012", который возвращает метод **onPostExecute()**.

А что случится, если задача находится в стадии выполнения, а мы вызовем метод **get()**? Давайте проверим.

Запустим снова приложение, нажмём на первую кнопку для запуска задачи и, не дожидаясь окончания задачи, нажмите на вторую кнопку.

Создатся ощущение, что программа остановилась. ProgressBar перестанет обновляться, текстовые сообщения также не выводятся в TextView. Но после окончания задачи появится текст и вернётся результат. Что же произошло?. Метод **get()** блокирует основной UI-поток и ждёт завершения AsyncTask. Это продолжается до тех пор, пока не выполнится код в методе **doInBackground()**. После этого метод **get()** возвращает результат и освобождает поток.

Таким образом метод блокирует поток, в котором он выполняется, и не отпустит, пока не получит какой-то результат или не выскочит исключение.

Тайм-аут

Есть ещё реализация метода с тайм-аутом. В этом случае метод **get()** будет ждать указанное время, а потом сгенерирует Exception. Если же задача уже была завершена, то метод выполнится сразу и ждать ничего не будет.

Перепишем код для второй кнопки:

```
public void onResultClick(View v) {
        if (cattask == null)
                return;
        int result = -1;
        try {
                result = cattask.get(1, TimeUnit.SECONDS);
                Toast.makeText(this, "Полученный результат: " + result,
                                Toast.LENGTH_LONG).show();
        } catch (InterruptedException e) {
                e.printStackTrace();
        } catch (ExecutionException e) {
                e.printStackTrace();
        } catch (TimeoutException e) {
                Log.e("AsyncTaskDemo", "get timeout, result = " + result);
                e.printStackTrace();
        }
}
```

Смотрим, что получилось. Приложение снова подвисает, но через секунду оживает и продолжает работу. Метод **get()** ждёт одну секунду, и если не получает результат, то генерирует исключение **TimeoutException**. Мы обрабатываем исключение и выводим в лог соответствующее сообщение. А приложение продолжат выполнять задачу и после успешной обработки метода **onPostExecute()** выводит результат.

Метод cancel() - отменяем задачу

Метод **cancel()** позволяет указать на отмену уже выполняющейся задачи. У метода один boolean-параметр, который указывает, может ли система прервать выполнение потока.

Кроме того, в методе **doInBackground()** можно проверять метод **isCancelled()**. Как только мы выполним метод **cancel()**, метод **isCancelled()** будет возвращать *true* и мы должны завершить метод **doInBackground()**. Таким образом, метод **cancel()** служит своеобразной меткой, что задачу нужно отменить. А метод **isCancelled()** будет считывать результат предыдущего метода и позволет выполнит код для завершения задачи.

Вернёмся к нашей истории с упрямым котом, который лезет на крышу. Оставим на экране две кнопки для запуска и отмены задачи. Саму задачу немного упростим, убрав отвлекающий код (используем один из первых вариантов примера в начале статьи).

```
// Если этот код работает, его написал Александр Климов,
// а если нет, то не знаю, кто его писал.
```

```
package ru.alexanderklimov.asynctaskdemo;

import ...

public class MainActivity extends Activity {

        CatTask cattask;
        TextView tvInfo;
        Button buttonStart;
        ProgressBar horizontalprogress;

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

                tvInfo = (TextView) findViewById(R.id.tvInfo);
                buttonStart = (Button) findViewById(R.id.buttonStart);
                horizontalprogress = (ProgressBar) findViewById(R.id.progress2);
        }

        public void onclick(View v) {
                cattask = new CatTask();
                cattask.execute();
        }

        public void onCancelClick(View v) {
                cattask.cancel(true);
        }

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

        class CatTask extends AsyncTask<Void, Integer, Void> {

                @Override
                protected void onPreExecute() {
                        super.onPreExecute();
                        tvInfo.setText("Полез на крышу");
                        buttonStart.setVisibility(View.INVISIBLE);
                }

                @Override
                protected Void doInBackground(Void... params) {
                        try {
                                int counter = 0;
```

```
                        for (int i = 0; i < 14; i++) {
                                getFloor(counter);
                                publishProgress(++counter);
                                if (isCancelled())
                                        return null;
                        }
                        // разъединяемся
                        TimeUnit.SECONDS.sleep(1);

                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                return null;
        }

        @Override
        protected void onProgressUpdate(Integer... values) {
                super.onProgressUpdate(values);
                tvInfo.setText("Этаж: " + values[0]);
                horizontalprogress.setProgress(values[0]);
        }

        @Override
        protected void onPostExecute(Void result) {
                super.onPostExecute(result);
                tvInfo.setText("Залез");
                buttonStart.setVisibility(View.VISIBLE);
                horizontalprogress.setProgress(0);
        }

        @Override
        protected void onCancelled() {
                super.onCancelled();
                tvInfo.setText("Передумал лезть на крышу");
                buttonStart.setVisibility(View.VISIBLE);
                horizontalprogress.setProgress(0);
        }

        private void getFloor(int floor) throws InterruptedException {
                TimeUnit.SECONDS.sleep(1);
        }
    }
}
```

Когда мы нажимаем на кнопку "Отменить операцию", то в методе **cancel()** используем параметр, равный *true*. В методе **doInBackground()** при работе цикла идёт проверка отмены (метод isCancelled()). Если приложение видит, что пользователь выбрал отмену задачи, то вместо метода **onPostExecute()** вызывается метод **onCancelled()**, в котором и прописываем свою логику кода.

В Android 4.0 появился ещё один метод **onCancelled(Void result)**, способный принимать результат от метода **doInBackground()**.

```
@Override
protected void onCancelled(Void result) {
        Log.d("AsyncTask", "onCancelled(Void) start");
        super.onCancelled(result);
        Log.d("AsyncTask", "onCancelled(Void) finish");
}
```

Текущее состояние задачи

Мы можем определить, в каком состоянии сейчас находится задача. Существует три состояния:

- **PENDING** – задача не запущена
- **RUNNING** – задача выполняется
- **FINISHED** – задача успешно завершена. Метод onPostExecute() отработал

Вы можете самостоятельно проверить, в каком состоянии находится ваша задача, добавив код в нужном месте:

```
Toast.makeText(this, cattask.getStatus().toString(), Toast.LENGTH_SHORT).show();
```

Вкратце, когда вы создаёте задачу (new CatTask()), то состояние **PENDING**. Когда вы запускаете задачу (execute()) и задача работает, то состояние **RUNNING**. Когда задача завершится, то состояние **FINISHED**. Если вы отменили задачу, то состояние по-прежнему будет **RUNNING**, поэтому используйте проверку через метод **isCancelled()** для более точного определения состояния:

```
if (cattask.isCancelled())
    Toast.makeText(this, "Отмена задачи", Toast.LENGTH_SHORT).show();
else
    Toast.makeText(this, cattask.getStatus().toString(), Toast.LENGTH_SHORT).show();
```

В моих опытах после отмены сначала показывало состояние **RUNNING**, а после повторной проверки уже показывало **FINISHED**, возможно состояние не сразу устанавливается.

Поворот экрана

Как вы знаете, при повороте экрана активность пересоздаётся. А что происходит с задачей? При повороте старые объекты будут потеряны, в том числе и ссылка на нашу задачу. Фактически получается, что при повороте создаётся ещё одна задача, которая начинает выполняться с самого начала, при этом где-то выполняется и старая задача.

Ситуация не совсем приятная. Для ознакомления с решениями этой задачи рекомендую почитать статью Урок 91. AsyncTask. Поворот экрана (http://startandroid.ru/uroki/vse-uroki-spiskom/154-urok-91-asynctask-povorot-ekrana.html), а заодно и другие материалы с сайта, связанные с AsyncTask.

Примеры

Используем AsyncTask для загрузки изображений из сети (http://developer.alexanderklimov.ru/android/asynctask-imageinet.php)

Используем AsyncTask для загрузки текстового файла из сети (http://developer.alexanderklimov.ru/android/asynctask-readfile.php)