Hello, ANDROID

Programowanie na platformę Google dla urządzeń mobilnych. WYDANIE III

Przygotuj aplikację na najpopularniejszą platformę mobilną!

ED BURNETTE

Helion

Hello, Android. Programowanie na platformę Google dla urządzeń mobilnych

Autor: Ed Burnette
Tłumaczenie: Krzysztof Sawka
ISBN: 978-83-246-3140-7
Tytuł oryginału: Hello, Android: Introducing Google's Mobile Development Platform
Format: 168×237, stron: 304

Przygotuj aplikację na najpopularniejszą platformę mobilną!

- Co czyni Androida wyjątkową platformą?
- Jak zaprojektować interfejs użytkownika?
- Jak skorzystać z zaawansowanych usług geolokalizacji i dostępu do sieci?

Android to system operacyjny dla telefonów komórkowych, który swoją dynamiką wzrostu udziału w rynku przyćmił niejeden produkt. Według badań IDC dla Europy Zachodniej w ciągu niecałego roku Android zwiększył swój udział z 4 do 31 procent! Co jest tego przyczyną? Niezwykle precyzyjnie wykonana i doskonale działająca synchronizacja danych ze wszystkimi najważniejszymi usługami firmy Google, łatwy dostęp do internetu dzięki szybkiej przeglądarce oraz tysiące świetnych aplikacji, zarówno darmowych, jak i płatnych. A skoro mowa o aplikacjach, jak stworzyć własny produkt na tę dynamicznie rozwijającą się platformę? Jak wprowadzić ją na rynek i zarobić?

Na te oraz wiele innych pytań związanych z Androidem odpowie Ci ta książka! Na samym wstępie poznasz trochę faktów związanych w powstawaniem samej platformy oraz zobaczysz, jakie nowości oferowały kolejne jej wersje. Wbrew pozorom wiedza ta przyda się przy tworzeniu aplikacji! Po krótkim wstępie historycznym dowiesz się, jak szybko przygotować stanowisko pracy, oraz poznasz kluczowe pojęcia z nim związane. W trakcie lektury nauczysz się projektować interfejs użytkownika, obsługiwać dane wejściowe czy multimedia. Niezwykle ważne są wiadomości zawarte w części trzeciej. Wykorzystanie tych informacji pozwoli Ci na stworzenie nowatorskiej aplikacji. Dostęp do sieci, usług geolokalizacyjnych, baz SQL i trójwymiarowej grafiki pozwoli Ci rozwinąć skrzydła! Książka ta jest długo wyczekiwaną pozycją, omawiającą wszystkie istotne zagadnienia związane z platformą Android, dlatego musisz ją mieć!

- Nowości w poszczególnych wersjach Androida
- Instalacja narzędzi oraz przygotowanie środowiska pracy
- Kluczowe pojęcia związane z wytwarzaniem aplikacji dla Androida
- Korzystanie z zasobów
- Projektowanie interfejsu użytkownika, tworzenie elementów menu
- Wykorzystanie motywów, tworzenie aktywnych tapet
- Grafika dwuwymiarowa oraz trójwymiarowa
- Obsługa danych wejściowych
- Wykorzystanie multimediów
- Przechowywanie danych na urządzeniu mobilnym
- Usługi geolokalizacji
- Wykorzystanie bazy danych SQLite
- Wielodotykowość
- Publikowanie aplikacji w serwisie Android Market

Wykorzystaj potencjał i popularność platformy Android!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 5.
Multimedia

Swego czasu firma Apple wyprodukowała serię telewizyjnych reklam, na których widzimy sylwetki ludzkie tańczące obłędnie w rytm muzyki generowanej przez iPoda[1]. Właśnie taki rodzaj podekscytowania chcemy wywołać w użytkownikach naszych aplikacji[2]. Dodanie muzyki, efektów dźwiękowych oraz filmów wideo może sprawić, że nasza aplikacja stanie się intensywniejsza oraz bardziej pociągająca niż w przypadku „suchej" aplikacji, zawierającej sam tekst i grafikę.

W tym rozdziale nauczymy się dodawać pliki multimediów do naszej aplikacji. Nie chcemy, aby użytkownicy wykonywali jakieś tańce-połamańce na ulicy, jeżeli jednak odpowiednio przygotujemy aplikację, wywołamy przynajmniej uśmiech na ich twarzach.

5.1. Odtwarzanie dźwięku

Dookoła panuje mroczna i niespokojna noc… Rozlega się gwizdek sędziego i wszyscy ruszają… Kibice szaleją, gdy drużyna Anwil Włocławek w ostatniej sekundzie zdobywa przewagę rzutem za trzy punkty…

Dzięki dźwiękom otrzymujemy wskazówki dotyczące danego otoczenia oraz dostosowujemy emocje do zdarzeń. Uznajmy dźwięki za kolejny sposób na wtargnięcie do głowy użytkownika. Podobnie jak w przypadku grafiki służącej do przekazywania pewnych informacji użytkownikowi, dźwięki pozwalają wzmocnić i zwiększyć wyrazistość danych.

---

[1] Polscy czytelnicy mogą się zapoznać z jedną reklamą tej z serii na stronie *http://www.youtube.com/watch?v=NlHUz99l-eo — przyp. tłum.*

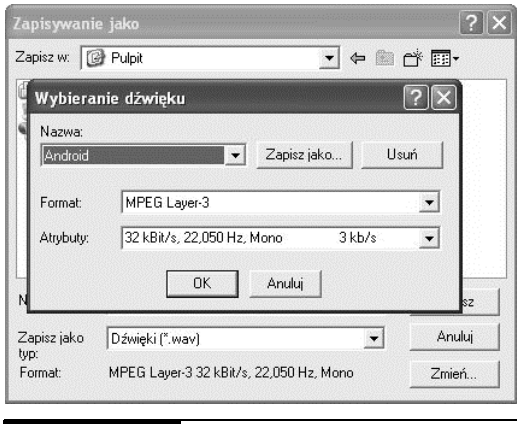
[2] Pomijając fakt, że osoby, które ukończyły 8. rok życia, nie mogą wyginać ciała w ukazany na reklamie sposób… poza momentami, gdy moje dzieci złośliwie umieszczają jaszczurkę w moich… cóż, taka mała dygresja.

Android obsługuje proces przetwarzania dźwięków i muzyki poprzez klasę `MediaPlayer`, znajdującą się w pakiecie `android.media`[3]. Wypróbujmy teraz prosty przykład, w którym dźwięk zostanie odtworzony po naciśnięciu przycisku na klawiaturze lub podkładce kierunkowej.

Rozpoczynamy od utworzenia projektu za pomocą następujących parametrów, wprowadzanych w oknie dialogowym *New Android Project*:

```
Project name: Audio
Build Target: Android 2.2
Application name: Audio
Package name: org.przyklad.audio
Create Activity: Audio
Min SDK Version: 8
```

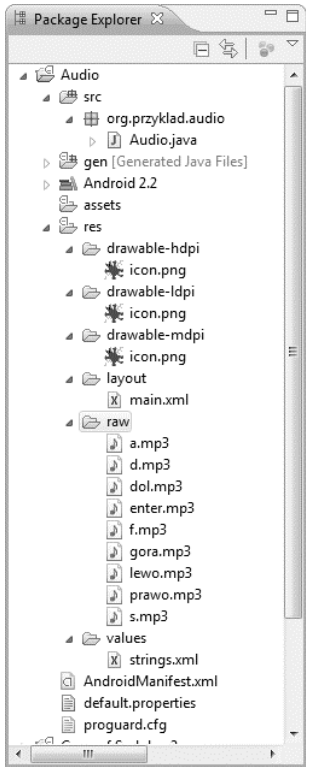
Będzie nam teraz potrzebnych kilka dźwięków do odtwarzania. Dla naszego przykładu stworzyłem własne dźwięki za pomocą aplikacji Rejestrator dźwięku (*Start/Wszystkie Programy/Akcesoria/Rozrywka/Rejestrator dźwięku*) oraz niedrogich słuchawek z mikrofonem. Po ustawieniu odpowiedniego poziomu głośności nagrałem każdy plik, wybrałem opcje Plik/Zapisz jako…, *następnie kliknąłem przycisk* Zmień… *i wybrałem format, który będzie* akceptowany przez Androida (rysunek 5.1). Na serwerze ftp wydawnictwa Helion znajdziemy wszystkie wygenerowane przeze mnie dźwięki oraz działający kod źródłowy aplikacji.



Rysunek 5.1. Efekty dźwiękowe zapisujemy w formacie zrozumiałym dla systemu Android

Skopiujmy pliki dźwiękowe do katalogu *res/raw* naszego nowego projektu. Jak pamiętamy z podrozdziału 2.4, „Korzystanie z zasobów" (strona 44), sam proces wklejenia pliku do katalogu *res* powoduje automatyczne stworzenie jego symbolu wewnątrz klasy `R`. Po skopiowaniu plików drzewo katalogów naszego projektu powinno przypominać zrzut zaprezentowany na rysunku 5.2.

[3] *http://d.android.com/guide/topics/media*



Rysunek 5.2. Pliki dźwiękowe kopiujemy do katalogu res/raw naszego projektu

Nadszedł teraz czas na zapełnienie aktywności `Audio` danymi. Najpierw zadeklarujemy pole nazwane `mp`, w którym będzie przechowywana instancja klasy `MediaPlayer`. W przypadku naszego programu tylko jedna instancja klasy `MediaPlayer` będzie aktywna w danym momencie.

Audio/src/org/przyklad/audio/Audio.java

```
package org.przyklad.audio;

import android.app.Activity;
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.KeyEvent;

public class Audio extends Activity {
    private MediaPlayer mp;

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

```
            setVolumeControlStream(AudioManager.STREAM_MUSIC);
        }
    }
```

Metoda `setVolumeControlStream()` informuje system, że po wciśnięciu przycisku zwiększania lub zmniejszania głośności przez użytkownika w trakcie działania aplikacji, powinien zostać zaktualizowany poziom głośności muzyki i innych elementów multimedialnych tego programu, a nie sygnału dzwonka.

Musimy teraz zacząć przechwytywać zdarzenia dotyczące naciskania przycisków oraz odtwarzać stosowne dźwięki. Dokonujemy tego poprzez przesłonięcie metody `Activity.onKeyDown()`.

Audio/src/org/przyklad/audio/Audio.java

```
Wiersz 1      @Override
-             public boolean onKeyDown(int kodKlaw, KeyEvent zdarzenie) {
-                 int idZas;
-                 switch (kodKlaw) {
5                 case KeyEvent.KEYCODE_DPAD_UP:
-                     idZas = R.raw.gora;
-                     break;
-                 case KeyEvent.KEYCODE_DPAD_DOWN:
-                     idZas = R.raw.dol;
10                    break;
-                 case KeyEvent.KEYCODE_DPAD_LEFT:
-                     idZas = R.raw.lewo;
-                     break;
-                 case KeyEvent.KEYCODE_DPAD_RIGHT:
15                    idZas = R.raw.prawo;
-                     break;
-                 case KeyEvent.KEYCODE_DPAD_CENTER:
-                 case KeyEvent.KEYCODE_ENTER:
-                     idZas = R.raw.enter;
20                    break;
-                 case KeyEvent.KEYCODE_A:
-                     idZas = R.raw.a;
-                     break;
-                 case KeyEvent.KEYCODE_S:
25                    idZas = R.raw.s;
-                     break;
-                 case KeyEvent.KEYCODE_D:
-                     idZas = R.raw.d;
-                     break;
30                case KeyEvent.KEYCODE_F:
-                     idZas = R.raw.f;
-                     break;
-                 default:
-                     return super.onKeyDown(kodKlaw, zdarzenie);
35                }
-
-                 // Zwalnia zasoby wcześniejszej instancji MediaPlayer
-                 if (mp != null) {
-                     mp.release();
```

```
40              }
-
-               // Tworzy nową instancję MediaPlayer do odtwarzania dźwięku
-               mp = MediaPlayer.create(this, idZas);
-               mp.start();
45
-               // Wskazuje na to, że dany klawisz został obsłużony
-               return true;
-           }
```

Na pierwszym etapie omawiana metoda wybiera zasób w oparciu o wciśnięty klawisz. Następnie w wierszu 39. wywołujemy metodę `release()`; zatrzymuje ona aktualnie odtwarzane dźwięki oraz zwalnia wszelkie zasoby powiązane z instancją klasy `MediaPlayer`. Jeżeli zapomnimy to zrobić, program zostanie samoistnie zamknięty (więcej informacji znajdziemy w ramce).

W wierszu 43. wykorzystujemy metodę `create()` do utworzenia nowej instancji klasy `MediaPlayer`, przy okazji używamy wybranego zasobu dźwiękowego, a następnie wywołujemy metodę `start()` do rozpoczęcia odtwarzania. Metoda `start()` działa w sposób asynchroniczny, zatem program wraca do wykonywania dalszej części kodu, bez względu na długość trwania dźwięku. Jeżeli chcemy zostać powiadomieni o zakończeniu odtwarzania, możemy użyć metody `setOnCompletionListener()`.

Gdy coś pójdzie nie tak

Osoby zajmujące się programowaniem multimediów odkryją szybko, że klasa `MediaPlayer` potrafi być narowistą bestią. W nowszych wersjach klasa ta została znacznie usprawniona, ciągle jednak potrafi zawiesić aplikację z najdrobniejszego powodu. Jedną z przyczyn takiego zachowania klasy `MediaPlayer` jest jej natywność, gdy na jej powierzchni znajduje się cienka warstwa środowiska Java. Natywny kod odtwarzacza został zoptymalizowany pod kątem wydajności, nie kontroli błędów.

Na szczęście silne zabezpieczenia procesów linuksowych w Androidzie znakomicie zapobiegają wszelkim uszkodzeniom w przypadku zawieszenia aplikacji. Emulator (lub urządzenie fizyczne) oraz inne aplikacje będą działały normalnie. Użytkownik będzie jedynie świadkiem zamknięcia pechowej aplikacji oraz prawdopodobnie ujrzy okno dialogowe zawierające komunikat o błędzie.

Jednakże na etapie tworzenia aplikacji posiadamy dostęp do o wiele bogatszych informacji diagnostycznych, pomagających określić, co się stało. W dzienniku systemowym Androida zostaną umieszczone komunikaty i tropy, które możemy przeglądać za pomocą widoku *LogCat* lub polecenia `adb logcat` (więcej informacji na ten temat zostało zawartych w podrozdziale 3.10, „Usuwanie błędów”, na stronie 73).

Jasiu pyta...

Jakie formaty dźwiękowe są obsługiwane przez system Android?

Sytuacja prezentuje się następująco: pewne formaty są obsługiwane wyłącznie na papierze, inne na emulatorze, a jeszcze inne na urządzeniach fizycznych. W wersji papierowej Android obsługuje następujące formaty plików (ma się to zmienić wraz z nowymi wersjami platformy):

- **WAV** (nieskompresowana modulacja PCM),
- **AAC** (format stosowany w iPodach firmy Apple, niechroniony),
- **MP3** (MPEG-3),
- **WMA** (ang. *Windows Media Audio*),
- **AMR** (kodek odpowiedzialny za sygnał mowy),
- **OGG** (format firmy Vorbis)[4],
- **MIDI** (instrumenty).

W rzeczywistości zauważyłem, że tylko formaty OGG, WAV i MP3 działają dobrze na emulatorze, a zatem jedynie takie rodzaje plików mogę zalecać do stosowania w procesie tworzenia aplikacji. Wydaje się, że natywnym formatem dźwiękowym w Androidzie jest 16-bitowy dźwięk stereo o częstotliwości próbkowania 44,1 kHz. Jednak pliki WAV spełniające wspomniane wymagania zajmują olbrzymią ilość miejsca, dlatego powinniśmy pozostać przy plikach OGG i MP3 (mono w przypadku mowy, stereo dla muzyki). Pliki OGG najlepiej się sprawdzają w przypadku krótkich dźwięków, na przykład efektów dźwiękowych w grach.

Trzymajmy się z daleka od niestandardowych częstotliwości, na przykład 8 kHz, ponieważ artefakty pojawiające się w czasie próbkowania bardzo deformują dźwięk. Najlepsze rezultaty osiągniemy przy częstotliwościach 11 kHz, 22 kHz i 44,1 kHz. Pamiętajmy, że chociaż telefon może posiadać malutki głośniczek, wielu użytkowników będzie podłączało słuchawki (tak jak w przypadku iPoda), lepiej więc, żeby były słyszane dźwięki w wysokiej jakości.

Jeżeli uruchomimy teraz program i wciśniemy któryś ze zdefiniowanych klawiszy (na przykład *Enter* lub środkowy przycisk podkładki kierunkowej), powinniśmy usłyszeć dźwięk. Jeżeli z głośników nie wydobywa się żaden dźwięk, sprawdźmy, czy mamy włączone głośniki (nie śmiać się), lub zajrzyjmy do komunikatów widoku *LogCat*. Jeżeli uruchomiliśmy aplikację na urządzeniu nieposiadającym klawiatury fizycznej, podkładki kierunkowej lub manipulatora kulkowego, wciśnijmy i przytrzymajmy przycisk *Menu*, dopóki nie pojawi się klawiatura dotykowa.

[4] *http://www.vorbis.com*

Jasiu pyta...

Jakie formaty wideo możemy oglądać na Androidzie?

Poniżej prezentuję oficjalnie obsługiwane formaty:

- **MP4** (format MPEG-4 o niskiej szybkości transmisji),
- **H.263** (3GP),
- **H.264** (AVC).

W przypadki wersji 1.5 systemu Android zalecanym formatem plików wideo jest H.263, ponieważ obsługuje go każda platforma sprzętowa oraz jest względnie wydajny pod kątem kodowania/dekodowania. Wykazuje on również kompatybilność z innymi urządzeniami, na przykład z iPhonem. Do konwersji z jednego formatu wideo na inny możemy stosować program QuickTime Pro[5]. W celu zaoszczędzenia miejsca używajmy w miarę niskiej rozdzielczości i prędkości transmisji, nie przesadzajmy jednak, gdyż możemy zupełnie utracić akceptowalną jakość obrazu.

Zauważmy, że w niektórych przypadkach efekt końcowy może brzmieć, jakby był przerywany lub opóźniany. W takim wypadku sprawdźmy inne formaty (na przykład OOG zamiast MP3) oraz mniejszą prędkość transmisji. Możemy również zbadać zastosowanie klasy `SoundPool`, która jawnie obsługuje kilka jednoczesnych strumieni dźwiękowych. W wersji Android 1.0 była ona pełna błędów oraz słabo udokumentowana, jednak od wersji 1.5 wydaje się być całkiem stabilna.

Naszą kolejną sztuczką będzie odtworzenie filmiku za pomocą jednej linijki kodu.

5.2. Odtwarzanie plików wideo

Obraz wideo to coś więcej niż zbiór zdjęć wyświetlanych kolejno po sobie. Składa się na niego również dźwięk, który musi być ściśle zsynchronizowany z poszczególnymi klatkami.

Klasa `MediaPlayer` obsługuje pliki wideo w taki sam sposób, z jakim mieliśmy do czynienia w przypadku samego dźwięku. Jedyna różnica polega tu na konieczności utworzenia klasy `Surface`, na której odtwarzacz będzie rysował klatki. Do kontroli odtwarzania używamy metod `start()` i `stop()`.

[5] *http://www.apple.com/quicktime/pro*

Nie zamierzam jednak pokazywać jeszcze jednego przykładu zawierającego klasę `MediaPlayer`, ponieważ istnieje łatwiejsza metoda wstawiania plików wideo do aplikacji: klasa `VideoView`. Aby zademonstrować jej działanie, stwórzmy za pomocą poniższych parametrów nowy projekt, zatytułowany *Wideo*:

```
Project name: Wideo
Build Target: Android 2.2
Application name: Wideo
Package name: org.przyklad.wideo
Create Activity: Wideo
Min SDK Version: 8
```

Zmieńmy układ graficzny (*res/layout/main.xml*) na następujący:

Wideow1/res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <VideoView
        android:id="@+id/wideo"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_gravity="center" />
</FrameLayout>
```

Teraz otwórzmy plik *Wideo.java* i zmieńmy metodę `onCreate()` w poniższy sposób:

Wideow1/src/org/przyklad/wideo/Wideo.java

```
package org.przyklad.wideo;

import android.app.Activity;
import android.os.Bundle;
import android.widget.VideoView;

public class Wideo extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Wypełnia widok danymi zasobu
        setContentView(R.layout.main);
        VideoView wideo = (VideoView) findViewById(R.id.wideo);

        // Wczytuje i uruchamia film
        wideo.setVideoPath("/data/przykladowewideo.3gp" );
        wideo.start();
    }
}
```

Metoda `setVideoPath()` otwiera dany plik, dopasowuje go do pojemnika przy jednoczesnym zachowaniu proporcji oraz rozpoczyna odtwarzanie.

Musimy teraz umieścić jakiś plik, który chcielibyśmy obejrzeć. W tym celu wpisujemy następujące polecenie:

```
C:\> adb push c:\kod\przykladowewideo.3gp /data/przykladowewideo.3gp
1649 KB/s (369870 bytes in 0.219s)
```

Plik *przykladowewideo.3gp* znajdziemy w pakiecie zawierającym kody źródłowe używane w tej książce, możemy również stworzyć swój własny plik. Zaprezentowany tu katalog (*/data*) został wprowadzony wyłącznie w celach pokazowych i w rzeczywistości nie powinien być używany do przechowywania plików multimedialnych.

Sposób ten działa wyłącznie w emulatorze, ponieważ w przypadku urządzeń fizycznych używany katalog jest chroniony.

Zauważmy, że Android raczej nie przejmuje się rozszerzeniem wstawianego pliku. Możemy również umieszczać i pobierać pliki za pomocą widoku *File Explorer* w perspektywie *Android*, uważam jednak, że do takich zadań lepiej nadaje się wiersz poleceń.

Jeszcze jedna sprawa: chcielibyśmy, aby odtwarzany film zajął cały ekran, włącznie z paskiem tytułowym oraz paskiem stanu. Aby to umożliwić, jedyną konieczną czynnością jest zdefiniowanie odpowiedniego motywu w pliku *AndroidManifest.xml*:

Videov1/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="org.przyklad.wideo"
        android:versionCode="1"
        android:versionName="1.0" >
    <application android:icon="@drawable/icon"
            android:label="@string/app_name" >
        <activity android:name=".Wideo"
                android:label="@string/app_name"
                 android:theme="@android:style/Theme.NoTitleBar.Fullscreen" >
            <intent-filter>
                    <action android:name="android.intent.action.MAIN" />
                    <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
        <uses-sdk android:minSdkVersion="3" android:targetSdkVersion="8" />
</manifest>
```

Gdy już przygotujemy wszystkie pliki, po uruchomieniu programu powinniśmy ujrzeć i usłyszeć filmik (rysunek 5.3). Sprawdźmy, czy plik wideo będzie odtwarzany zarówno w orientacji pionowej, jak i poziomej. Voilà! Natychmiastowa, kinematograficzna przyjemność.

Usprawnijmy teraz nieco naszą grę Sudoku, dodając klimatyczną muzykę.

Rysunek 5.3. Dzięki klasie VideoView wstawianie plików wideo nie stanowi problemu

5.3. Dodawanie dźwięków do gry Sudoku

W tym podrozdziale wykorzystamy całą zdobytą dotychczas wiedzę na temat multimediów i dodamy do naszej gry Sudoku muzykę odtwarzaną w tle. Jeden utwór będzie grał na ekranie tytułowym, a inny usłyszymy w trakcie właściwej rozgrywki. W ten sposób nie tylko się dowiemy, w jaki sposób odtwarzać muzykę, lecz również rozpatrzymy pewne zagadnienia związane z cyklem życia.

Aby dodać muzykę do ekranu powitalnego, wystarczy, że przesłonimy następujące metody w klasie `Sudoku`:

Sudokuw3/src/org/przyklad/sudoku/Sudoku.java

```
@Override
protected void onResume() {
    super.onResume();
    Muzyka.play(this, R.raw.glowna);
}

@Override
protected void onPause() {
    super.onPause();
    Muzyka.stop(this);
}
```

Jasiu pyta...

Dlaczego filmik zostaje zrestartowany po zmianie orientacji wyświetlacza?

Android domyślnie zakłada, że napisana aplikacja nie rozróżnia trybów orientacji ekranu. Aby wyświetlić ewentualne zmiany w zasobach, system niszczy i od podstaw odtwarza aktywność. Oznacza to, że metoda `onCreate()` zostaje ponownie wywołana, a to z kolei znaczy, iż plik wideo zostaje uruchomiony od początku (tak jakby cały przykład był pisany na bieżąco).

Takie zachowanie nie jest kłopotliwe w przypadku 90% programów, więc większość programistów nie musi się nim przejmować. W gruncie rzeczy jest to całkiem przydatny sposób na przetestowanie cyklu życia aplikacji oraz kodu odpowiedzialnego za zapis/odczyt jej stanu (podrozdział 2.2, „To żyje!", na stronie 38). Jednakże istnieje kilka rozwiązań dla ambitnych, pozwalających na zoptymalizowanie przejścia pomiędzy trybem pionowym a poziomym.

Najprostszym rozwiązaniem jest implementacja metody `onRetainNonConfiguration`
`↪Instance()` wewnątrz aktywności, dzięki czemu pewne dane zostaną zachowane podczas wywołań metod `onDestroy()` i `onCreate()`. W nowym wystąpieniu aktywności wykorzystujemy wtedy metodę `getLastNonConfigurationInstance()` do odzyskania tych danych. Możemy przechowywać wszelkie rodzaje informacji, nawet referencje do bieżącej intencji oraz uruchomionych wątków.

Bardziej zaawansowaną metodą jest zastosowanie właściwości `android:config`
`↪Changes=` wewnątrz pliku *AndroidManifest.xml* w taki sposób, aby Android wiedział, którymi zmianami możemy się zająć. Na przykład, jeśli ustawimy jej wartość `keyboardHidden|orientation`, w momencie obrócenia klawiatury aktywność nie zostanie zniszczona i odtworzona. Zamiast tego system wywoła `onConfiguration`
`↪Changed(Configuration)` z założeniem, że mamy sytuację pod kontrolą[6].

Jeśli przypomnimy sobie, co przeczytaliśmy w podrozdziale 2.2, „To żyje!" (strona 38), metoda `onResume()` zostaje wywołana w momencie, gdy aktywność jest gotowa do interakcji z użytkownikiem. Jest to odpowiedni moment, aby uruchomić muzykę, zatem wstawiamy tutaj wywołanie `Muzyka.play()`. Wkrótce zdefiniujemy klasę `Muzyka`.

Odnośnik `R.raw.glowna` dotyczy pliku *res/raw/glowna.mp3*. Pliki dźwiękowe znajdziemy w projekcie *Sudokuw3*, który można pobrać wraz z innymi przykładowymi projektami z serwera ftp wydawnictwa Helion.

[6] Szczegółowe informacje znajdziemy na stronie *http://d.android.com/reference/android/app/Activity.html#Configuration-Changes*.

Jasiu pyta...

Czy nie powinniśmy wprowadzić muzyki do usługi przetwarzanej w tle?

Nie poświęciliśmy zbyt wiele miejsca klasie `Service`, mogliśmy się jednak na nią natknąć w różnych przykładach wykorzystujących muzykę, które można znaleźć w internecie. Zasadniczo klasa `Service` służy do rozpoczęcia procesu przetwarzanego w tle, który będzie działał nawet po zamknięciu aktywności. Usługi są podobne do demonów systemu Linux (ale nie identyczne). Jeżeli tworzymy odtwarzacz muzyki ogólnego przeznaczenia i chcemy, aby muzyka leciała podczas przeglądania poczty lub stron sieci Web, to zastosowanie klasy `Service` będzie jak najbardziej na miejscu. Jednakże w większości przypadków wolimy, aby muzyka zakończyła się w momencie zamknięcia programu, nie ma więc potrzeby, aby implementować klasę `Service`.

Metoda `onPause()` została wprowadzona w formie „wsparcia" dla metody `onResume()`. Android wstrzymuje bieżącą aktywność tuż przed rozpoczęciem kolejnej, więc w przypadku naszej gry po wciśnięciu przycisku *Nowa Gra* aktywność `Sudoku` zostanie wstrzymana, natomiast będziemy świadkami uruchomienia aktywności `Gra`. Również w przypadku naciśnięcia klawisza cofania lub powrotu do ekranu startowego zostanie wywołana metoda `onPause()`. We wszystkich wymienionych tu miejscach chcemy, aby nie grała muzyka tytułowa, zatem wywołujemy `Muzyka.stop()` w metodzie `onPause()`.

Przeprowadźmy podobną operację w aktywności `Gra`:

Sudokuw3/src/org/przyklad/sudoku/Gra.java

```
@Override
protected void onResume() {
    super.onResume();
    Muzyka.play(this, R.raw.gra);
}

@Override
protected void onPause() {
    super.onPause();
    Muzyka.stop(this);
}
```

Ciekawostka o grze Sudoku

Istnieją dziesiątki odmian gry w Sudoku, jednak żadna nie zdołała zdobyć popularności porównywalnej z wersją oryginalną. Jedna z takich modyfikacji umożliwia grę na polu o rozmiarach 16×16 pól, danymi wejściowymi są natomiast cyfry zapisywane w systemie szesnastkowym. Inna wersja, znana jako Gattai 5 lub Samurai Sudoku, zawiera pięć plansz o rozmiarach 9×9 pól, które łączą się ze sobą w narożnikach.

W porównaniu do klasy `Sudoku` wprowadziliśmy tu inny zasób dźwiękowy, `R.raw.gra` (*res/raw/gra.mp3*). Ostatnim fragmentem naszej dźwiękowej układanki jest klasa `Muzyka`, w której będzie zarządzana klasa `MediaPlayer`, wymagana do odtwarzania muzyki:

Sudokuv3/src/org/example/sudoku/Music.java

```
Wiersz 1    package org.przyklad.sudoku;
-
-           import android.content.Context;
-           import android.media.MediaPlayer;
5
-           public class Muzyka {
-               private static MediaPlayer mp = null;
-
-               /** Zatrzymuje stary utwór i rozpoczyna odtwarzanie nowego */
10              public static void play(Context kontekst, int zasob) {
-                   stop(kontekst);
-                   mp = MediaPlayer.create(kontekst, zasob);
-                   mp.setLooping(true);
-                   mp.start();
15              }
-
-               /** Zatrzymuje odtwarzanie muzyki */
-               public static void stop(Context context) {
-                   if (mp != null) {
20                     mp.stop();
-                       mp.release();
-                       mp = null;
-                   }
-               }
25          }
```

Metoda `play()` najpierw wywołuje metodę `stop()`, dzięki czemu wszelkie odtwarzanie bieżącej muzyki zostaje zatrzymane. Następnie za pomocą metody `MediaPlayer.create()` zostaje utworzona nowa instancja klasy `MediaPlayer`, której zostaje przekazany kontekst oraz identyfikator zasobów.

Po utworzeniu odtwarzacza wprowadzamy opcję zapętlenia muzyki i w końcu ją odtwarzamy. Metoda `start()` od razu pozwala aplikacji na przetwarzanie dalszej części kodu.

Rozpoczynająca się od wiersza 18. metoda `stop()` jest bardzo prosta. Po sprawdzeniu, czy rzeczywiście jest dostępna instancja klasy `MediaPlayer`, wywołujemy metody `stop()` i `release()`. O dziwo, metoda `MediaPlayer.stop()` zatrzymuje odtwarzanie muzyki. Z kolei metoda `release()` zwalnia zasoby systemowe powiązane z odtwarzaczem. Ponieważ są to natywne zasoby, nie możemy czekać, aż zwykły proces odzyskiwania pamięci środowiska Java wykona za nas to zadanie. Pozostawienie w spokoju metody `release()` stanowi świetny sposób na niespodziewane zawieszenie aplikacji (nie to, żeby mi się kiedykolwiek coś takiego przydarzyło; czytelnicy powinni jednak o tym pamiętać).

Teraz nadeszła pora na najlepszą część — zagrajmy w Sudoku po wprowadzeniu tych wszystkich zmian. Przetestujmy aplikację w każdy możliwy sposób, na przykład przechodząc pomiędzy różnymi aktywnościami, wciskając przyciski cofania i ekranu startowego na różnych etapach gry, obracając wyświetlacz, i tak dalej. Właściwe zarządzanie cyklem życia stanowi czasami nie lada kłopot, jednak użytkownicy aplikacji docenią włożony w nią wysiłek.

5.4. Przewijamy >>

W niniejszym rozdziale zajęliśmy się tematyką odtwarzania plików audio i wideo za pomocą pakietu Android SDK. Nie omawialiśmy procesu rejestracji multimediów, ponieważ większość programów nie wykorzystuje tej funkcji, jeżeli jednak chcemy stworzyć program będący pod tym względem wyjątkowym, możemy zajrzeć do dokumentacji klasy `MediaRecorder`[7].

W rozdziale 6., „Przechowywanie danych lokalnych" (strona 121), poznamy kilka prostych metod zachowywania danych aplikacji pomiędzy jej wywołaniami. Jeżeli ta wiedza jest nam niepotrzebna, możemy od razu przejść do rozdziału 7., „Podłączenie do świata" (strona 133), aby dowiedzieć się co nieco o dostępie do sieci.

[7] *http://d.android.com/reference/android/media/MediaRecorder.html*

Skorowidz

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

V

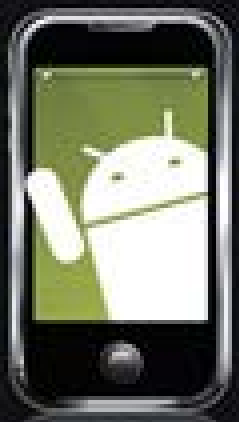
W

X

Y

Z

Ź



Android to system operacyjny dla telefonów komórkowych, który swoją dynamiką wzrostu udziału w rynku przyćmił niejeden produkt. Według badań IDC dla Europy Zachodniej w ciągu niecałego roku Android zwiększył swój udział z 4 do 31 procent! Co jest tego przyczyną? Niezwykle precyzyjnie wykonana i doskonale działająca synchronizacja danych ze wszystkimi najważniejszymi usługami firmy Google, łatwy dostęp do internetu dzięki szybkiej przeglądarce oraz tysiące świetnych aplikacji, zarówno darmowych, jak i płatnych. A skoro mowa o aplikacjach, jak stworzyć własny produkt na tę dynamicznie rozwijającą się platformę? Jak wprowadzić go na rynek i zarobić?

Na te oraz wiele innych pytań związanych z Androidem odpowie Ci ta książka! Na samym wstępie poznasz trochę faktów związanych z powstawaniem samej platformy oraz zobaczysz, jakie nowości oferowały kolejne jej wersje. Wbrew pozorom wiedza ta przyda się przy tworzeniu aplikacji! Po krótkim wstępie historycznym dowiesz się, jak szybko przygotować stanowisko pracy, oraz poznasz kluczowe pojęcia z nim związane. W trakcie lektury nauczysz się projektować interfejs użytkownika, obsługiwać dane wejściowe czy multimedia. Niezwykle ważne są wiadomości zawarte w części trzeciej. Wykorzystanie tych informacji pozwoli Ci na stworzenie nowatorskiej aplikacji. Dostęp do sieci, usług geolokalizacyjnych, baz SQL i trójwymiarowej grafiki pozwoli Ci rozwinąć skrzydła! Książka ta jest długo wyczekiwaną pozycją, omawiającą wszystkie istotne zagadnienia związane z platformą Android, dlatego musisz ją mieć!

- **Nowości w poszczególnych wersjach Androida**
- **Instalacja narzędzi oraz przygotowanie środowiska pracy**
- **Kluczowe pojęcia związane z wytwarzaniem aplikacji dla Androida**
- **Korzystanie z zasobów**
- **Projektowanie interfejsu użytkownika**
- **Wykorzystanie motywów**
- **Tworzenie elementów menu**
- **Grafika dwuwymiarowa oraz trójwymiarowa**
- **Obsługa danych wejściowych**
- **Wykorzystanie multimediów**
- **Przechowywanie danych na urządzeniu mobilnym**
- **Usługi geolokalizacji**
- **Wykorzystanie bazy danych SQLite**
- **Wielodotykowość**
- **Tworzenie aktywnych tapet**
- **Publikowanie aplikacji w serwisie Android Market**

Wykorzystaj potencjał i popularność platformy Android!

helion.pl
helion

Nr katalogowy: 6607

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Cena 57,00 zł

ISBN 978-83-246-3140-7

Informatyka w najlepszym wydaniu