Błyskawicznie opanuj środowisko produkcyjne dla Androida!

Android 2
Tworzenie aplikacji

Sayed Hashimi • Satya Komatineni • Dave MacLean

Jak rozpocząć przygodę z systemem Android?

Jak korzystać z usług geolokalizacyjnych?

Jak tworzyć trójwymiarową grafikę, korzystającą z biblioteki OpenGL?

Android 2. Tworzenie aplikacji

Autorzy: Sayed Hashimi, Satya Komatineni, Dave MacLean
Tłumaczenie: Krzysztof Sawka
ISBN: 978-83-246-2754-7
Tytuł oryginału: Pro Android 2
Format: 158×235, stron: 704

Błyskawicznie opanuj środowisko produkcyjne dla Androida!

- Jak rozpocząć przygodę z systemem Android?
- Jak korzystać z usług geolokalizacyjnych?
- Jak tworzyć trójwymiarową grafikę, korzystającą z biblioteki OpenGL?

Jakiś czas temu mogło się wydawać, że rynek systemów operacyjnych dla smartfonów jest podzielony i nie ma na nim miejsca dla żadnych debiutantów. Jednak najwięksi gracze tego segmentu byli w dużym błędzie. Pojawił się Android. Nowatorski, przyjazny, atrakcyjny oraz w najwyższym stopniu zintegrowany z usługami Google'a system operacyjny dla telefonów. Fakt, że był firmowany przez jedną z najbardziej znanych firm – Google – niewątpliwie dał mu ogromną siłę przebicia. Liczba jego fanów rośnie wręcz w tempie geometrycznym!

Błyskawicznie powstały niezliczone aplikacje dla tego systemu. Codziennie zastępy programistów tworzą nowe, jeszcze lepsze programy. Dzisiaj ich liczba jest szacowana na grubo ponad 80 tysięcy! Teraz i Ty będziesz mógł dołączyć do grona ludzi tworzących aplikacje na tę platformę. Niniejsza książka stanowi kompletny przewodnik po najnowszej wersji systemu Android. Na samym początku dowiesz się, jak skonfigurować środowisko produkcyjne, oraz poznasz strukturę i elementy składowe aplikacji dla Androida. Kolejne rozdziały to same smakowite kąski dla każdego programisty – tworzenie interfejsu użytkowników i animacji dwu- oraz trójwymiarowych. Nauczysz się także efektywnie korzystać z zasobów i usług lokalizacyjnych, tworzyć widgety oraz podłączać się do zasobów sieci Internet. Książka ta jest idealną pozycją dla każdego, kto chce rozpocząć przygodę z tworzeniem aplikacji dla telefonów komórkowych.

- Przygotowanie środowiska pracy
- Podstawowe składniki systemu Android
- Wykorzystanie zasobów, dostawców treści oraz intencji
- Tworzenie interfejsu użytkownika – używanie kontrolek
- Zastosowanie menedżerów układu graficznego
- Praca z menu
- Prezentowanie animacji dwuwymiarowych
- Geokodowanie w Androidzie
- Wykorzystanie usług geolokalizacyjnych
- Zastosowanie modułu HttpClient
- Wykorzystanie biblioteki OpenGL do tworzenia grafiki trójwymiarowej
- Zarządzanie preferencjami
- Tworzenie widgetów ekranu początkowego
- Usługi wyszukiwania
- Korzystanie ze sklepu Android Market

Twórz aplikacje na jedną z najbardziej dynamicznie rozwijających się platform!

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

Spis treści

ROZDZIAŁ

2

Pierwsze koty za płoty

W poprzednim rozdziale omówiliśmy historię Androida oraz zarysowaliśmy koncepcje, które zostaną omówione w dalszej części książki. W tym momencie Czytelnik prawdopodobnie może chcieć już zająć się kodem. Rozpoczniemy od przedstawienia elementów potrzebnych do tworzenia aplikacji w środowisku Android SDK oraz od przygotowania tego środowiska. Następnie szczegółowo przeanalizujemy aplikację „Hello World!" oraz rozłożymy na czynniki pierwsze nieco bardziej złożony fragment kodu. W dalszej kolejności objaśnimy cykl życia aplikacji w Androidzie, a na końcu poświęcimy chwilę tematowi wyszukiwania błędów w aplikacji za pomocą narzędzi AVD (ang. *Android Virtual Devices* — wirtualne urządzenia Androida).

Do tworzenia aplikacji przeznaczonych dla Androida wymagane jest posiadanie zestawu JDK (ang. *Java SE Development Kit* — zestaw do projektowania w środowisku Java SE), środowiska Android SDK oraz środowiska projektowego. Inaczej mówiąc, można tworzyć aplikacje za pomocą prymitywnego edytora tekstowego, ale podczas pisania książki używaliśmy powszechnie dostępnego środowiska IDE Eclipse. Android SDK wymaga zestawu JDK w wersji co najmniej 5 (korzystaliśmy z JDK 6) oraz środowiska Eclipse w wersji nie wcześniejszej niż 3.3 (używaliśmy wersji Eclipse 3.5, inaczej zwanej Galileo). Mieliśmy zainstalowane środowisko Android SDK 2.0.

Żeby ułatwić sobie życie, można zainstalować narzędzia ADT (ang. *Android Development Tools* — narzędzia projektowe Androida). Jest to wtyczka środowiska Eclipse, umożliwiająca tworzenie aplikacji przeznaczonych do Androida w środowisku IDE Eclipse. W istocie wszystkie przykłady w tej książce zostały zaprojektowane w środowisku Eclipse za pomocą narzędzi ADT.

Konfigurowanie środowiska

Żeby móc tworzyć aplikacje na Androida, należy stworzyć środowisko projektowe. W tym podrozdziale zajmiemy się omówieniem procesu pobierania aplikacji JDK 6, środowiska Eclipse, zestawu Android SDK oraz dodatku ADT. Pomożemy także skonfigurować środowisko Eclipse tak, aby można było w nim tworzyć aplikacje na Androida.

Środowisko Android SDK jest kompatybilne z systemami Windows (Windows XP, Windows Vista oraz Windows 7), Mac OS X (jedynie z procesorami Intel) oraz Linux (również wyłącznie z procesorami Intel). W tym rozdziale omówimy proces konfigurowania środowiska we wszystkich wymienionych rodzajach systemów (w przypadku Linuksa jedynie dla wariantu Ubuntu). W kolejnych rozdziałach nie będziemy się zajmować różnicami pomiędzy poszczególnymi platformami.

Pobieranie zestawu JDK 6

Pierwszym potrzebnym składnikiem jest zestaw Java Development Kit. Środowisko Android SDK wymaga co najmniej wersji 5 zestawu JDK; przykłady w książce były tworzone w obecności wersji 6. Dla systemów Windows aplikacja JDK 6 jest dostępna na oficjalnej stronie firmy Sun (*http://java.sun.com/javase/downloads/*) — należy ją pobrać i zainstalować. Wystarczy edycja standardowa aplikacji JDK, jej wersje Bundle nie są wymagane. Zestaw JDK dla systemu Mac OS X znaleźć można w witrynie Apple (*http://developer.apple.com/java/download/*), należy stamtąd wybrać plik do odpowiedniej wersji systemu i zainstalować go. Żeby zainstalować JDK w systemie Linux, należy otworzyć okno terminalu i wpisać następujące polecenie:

```
sudo apt-get install sun-java6-jdk
```

Zostanie zainstalowana aplikacja JDK oraz wszystkie wymagane, dodatkowe składniki, takie jak środowisko JRE (ang. *Java Runtime Environment* — środowisko uruchomieniowe Java).

Następnie należy skonfigurować zmienną środowiskową `JAVA_HOME`, tak żeby wskazywała folder instalacyjny JDK. Na komputerze z zainstalowanym systemem Windows XP można tego dokonać, otwierając menu *Start*, klikając prawym przyciskiem myszy ikonę *Mój Komputer* i wybierając z menu opcję *Właściwości*, przechodząc do zakładki *Zaawansowane* i klikając przycisk *Zmienne środowiskowe*. Następnie należy kliknąć przycisk *Nowa*, żeby dodać zmienną, lub *Edycja*, żeby naprawić istniejącą zmienną. Wartość zmiennej `JAVA_HOME` będzie wyglądała mniej więcej następująco: *C:\Program Files\Java\jdk1.6.0_18*. W systemach Windows Vista oraz Windows 7 droga do okna *Zmienne środowiskowe* wygląda nieco inaczej. Trzeba wybrać menu *Start*, prawym przyciskiem myszy kliknąć ikonę *Komputer* i wybrać z menu opcję *Właściwości*, wybrać łącze *Zaawansowane ustawienia systemu*, a następnie użyć przycisku *Zmienne środowiskowe…*. Kolejne czynności wymagane do ustanowienia lub zmiany zmiennej środowiskowej `JAVA_HOME` są identyczne jak w systemie Windows XP. W systemie Mac OS X zmienną środowiskową `JAVA_HOME` konfiguruje się w pliku *.profile*, umieszczonym w katalogu *HOME*. Należy utworzyć lub edytować ten plik i dodać następującą linijkę:

```
export JAVA_HOME=ścieżka_do_katalogu_JDK
```

gdzie w miejscu `ścieżka_do_katalogu_JDK` będzie prawdopodobnie `/Library/Java/Home`. W systemie Linux należy edytować plik *.profile* oraz dodać taką samą linijkę, jak w przypadku systemu Mac OS X, z tym, że docelową ścieżką, którą należy dodać, najprawdopodobniej będzie `/usr/lib/jvm/java-6-sun`.

Pobieranie środowiska Eclipse 3.5

Po zainstalowaniu pakietu JDK można pobrać środowisko Eclipse IDE for Java Developers (edycja dla Java EE nie jest wymagana; będzie działać, lecz zajmuje o wiele więcej miejsca i zawiera elementy, których nie będziemy potrzebować). Przykłady przygotowane dla tej książki zostały napisane w obecności środowiska Eclipse 3.5 (na systemie Windows).

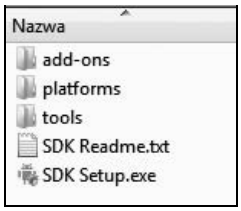
Wszystkie wersje Eclipse dostępne są pod adresem *http://www.eclipse.org/downloads/*. Pliki środowiska są skompresowane w formacie *.zip*, można je wypakować w dowolnym miejscu. Najprościej jest wypakować je do partycji *C:*, co spowoduje utworzenie katalogu *C:\Eclipse*. Można w nim znaleźć plik wykonywalny *eclipse.exe*. W przypadku systemu Mac OS X pliki można rozpakować do katalogu *Applications*, a w Linuksie — do katalogu *HOME*. We wszystkich przypadkach plik wykonywalny środowiska Eclipse zostaje umieszczony w utworzonym folderze.

Podczas pierwszego uruchomienia środowiska Eclipse pojawi się prośba o określenie ścieżki do przestrzeni roboczej. Żeby nie komplikować sobie zbytnio życia, można wpisać tak prosty adres, jak na przykład *C:\android*. Jeżeli komputer jest współużytkowany przez kilka osób, dobrze jest umieścić folder przestrzeni roboczej gdzieś w katalogu swojego profilu.

Pobieranie zestawu Android SDK

Zasadniczym składnikiem tworzenia aplikacji na Androida jest środowisko programistyczne Android SDK. W zestawie tym umieszczony został emulator, zatem nie jest wymagane posiadanie urządzenia przenośnego z systemem Android do projektowania aplikacji. Tak naprawdę wszystkie przykłady z książki były pisane w systemie Windows XP.

Zestaw Android SDK jest dostępny pod adresem *http://developer.android.com/sdk*. Podobnie jak w przypadku środowiska Eclipse jest on skompresowany w formie pliku *.zip*, zatem należy go wypakować do wybranej lokacji. W systemach Windows można umieścić te pliki na przykład w partycji *C:*, a po rozpakowaniu powinien pojawić się folder, nazwany *android-sdk-windows* lub podobnie, w którym będą obecne pliki, przedstawione na rysunku 2.1. W przypadku systemu Mac OS X oraz Linux zestaw Android SDK można wypakować do katalogu *HOME*.



Rysunek 2.1. Zawartość folderu Android SDK

Środowisko Android SDK zawiera katalog narzędzi, który warto umieścić w zmiennej systemowej `PATH`. Dodajmy go teraz lub upewnijmy się, że jest właściwie umieszczony. Przy okazji ułatwimy sobie pracę, dodając także katalog *bin* zestawu JDK. W systemie Windows należy wrócić do opisanego powyżej okna zmiennych środowiskowych. Następnie trzeba edytować zmienną `PATH`, dodać na końcu średnik (;), wpisać ścieżkę do folderu *tools* Androida SDK i, po kolejnym średniku, umieścić wpis `%JAVA_HOME%\bin`. Wystarczy teraz kliknąć przycisk *OK*. W przypadku systemów Mac OS X oraz Linux należy edytować plik *.profile* i dodać ścieżkę do folderu *tools* oraz parametr `$JAVA_HOME/bin` do zmiennej `PATH`. Powinno to wyglądać mniej więcej tak:

```
export PATH=$PATH:$HOME/android-sdk-linux_x86/tools:$JAVA_HOME/bin
```

W dalszej części książki pojawią się momenty, gdy trzeba będzie uruchamiać pewne programy z wiersza poleceń. Są one częścią środowiska JDK lub Android SDK. Dzięki umieszczeniu ich w zmiennej systemowej `PATH` nie będzie trzeba wpisywać pełnej ścieżki do nich,

jednak do uruchomienia tych programów konieczne jest uruchomienie „okna narzędzi". W następnych rozdziałach będziemy korzystać z takiego okna narzędzi. Najprostszym sposobem jego utworzenia w systemie Windows jest kliknięcie menu *Start/Wyszukaj*, a następnie wpisanie `cmd` w polu tekstowym i kliknięcie przycisku *OK*. W systemie Mac OS X należy wybrać aplikację *Terminal* w folderze *Applications* z poziomu menedżera plików *Finder* lub z poziomu *Dock*. W systemie Linux aplikacja *Terminal* znajduje się w menu *Applications/Accessories*.

Jeszcze jedna rzecz dotycząca różnic pomiędzy platformami: w pewnej chwili może być wymagana znajomość adresu IP stacji roboczej. W systemie Windows należy uruchomić wiersz poleceń i wpisać komendę `ipconfig`. Wśród wyników będzie widniał wpis dotyczący IPv4 (lub podobnie), a obok zostanie wyświetlony adres IP danego komputera. Wygląda on mniej więcej tak: `192.168.1.25`. W systemach Mac OS X oraz Linux należy uruchomić wiersz poleceń i wpisać `ifconfig`. Adres IP jest umieszczony obok wpisu `inet addr`. Może też być widoczne połączenie sieciowe przy nazwie *localhost* lub *lo*. Adres IP tego połączenia to `127.0.0.1`. Jest to specjalny typ połączenia sieciowego, wykorzystywany przez system operacyjny, i nie ma nic wspólnego z adresem IP stacji roboczej. Należy poszukać wiersza, w którym widoczny jest inny adres IP.

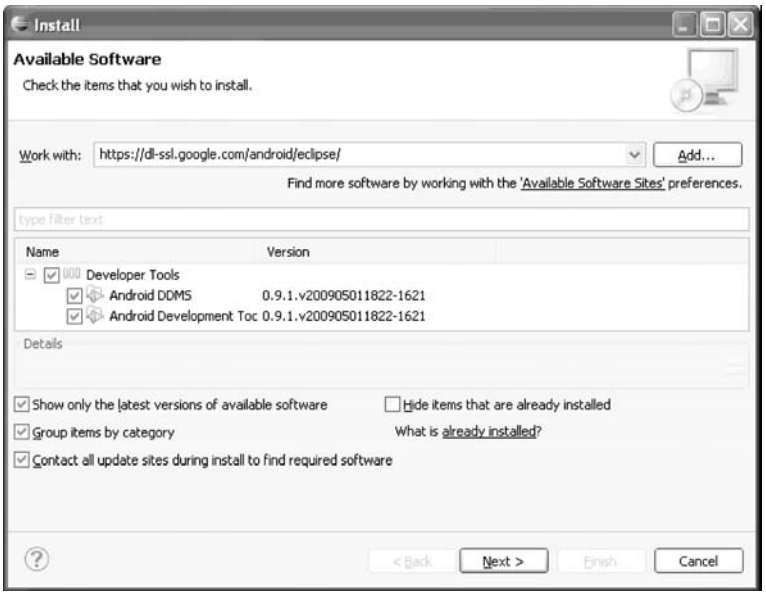
Instalowanie narzędzi ADT

Teraz należy zainstalować narzędzia ADT, wtyczkę środowiska Eclipse usprawniającą tworzenie aplikacji na Androida. Dokładniej mówiąc, narzędzia ADT łączą się ze środowiskiem Eclipse, dzięki czemu można tworzyć i testować aplikacje przeznaczone na platformę Android oraz wyszukiwać w nich błędy. Żeby zainstalować tę wtyczkę, należy skorzystać z funkcji *Install New Software…*, dostępnej w aplikacji Eclipse. Informacje dotyczące aktualizacji wtyczek można znaleźć pod poniższymi instrukcjami. Żeby zainstalować narzędzia ADT, należy uruchomić środowisko Eclipse i wykonać następujące czynności:

1. W pasku narzędzi wybierz opcję *Help*, a następnie kliknij opcję *Install New Software…*. W poprzednich wersjach aplikacji Eclipse była ona nazwana *Software Updates*.
2. Zaznacz pole tekstowe *Work with…*, wpisz *https://dl-ssl.google.com/android/eclipse/* i naciśnij klawisz *Enter*/*Return*. Aplikacja połączy się z witryną i wyświetli listę, pokazaną na rysunku 2.2.
3. Powinien pojawić się węzeł *Developer Tools*, posiadający dwie podrzędne kategorie: *Android DDMS* oraz *Android Development Tools*. Zaznacz węzeł nadrzędny oraz upewnij się, że elementy podrzędne są również zaznaczone, a następnie kliknij przycisk *Next*. Prawdopodobnie numery wersji będą wyższe niż przedstawione na rysunku, ale nie szkodzi.
4. Pojawi się okno potwierdzenia instalacji wtyczek. Kliknij *Next*.
5. Na następnym ekranie zostaniesz poproszony o przejrzenie licencji narzędzi ADT, a także licencji dotyczących narzędzi potrzebnych do zainstalowania wtyczki. Przejrzyj licencje, zaznacz opcję *I accept the terms od license agreements* i kliknij przycisk *Finish*.

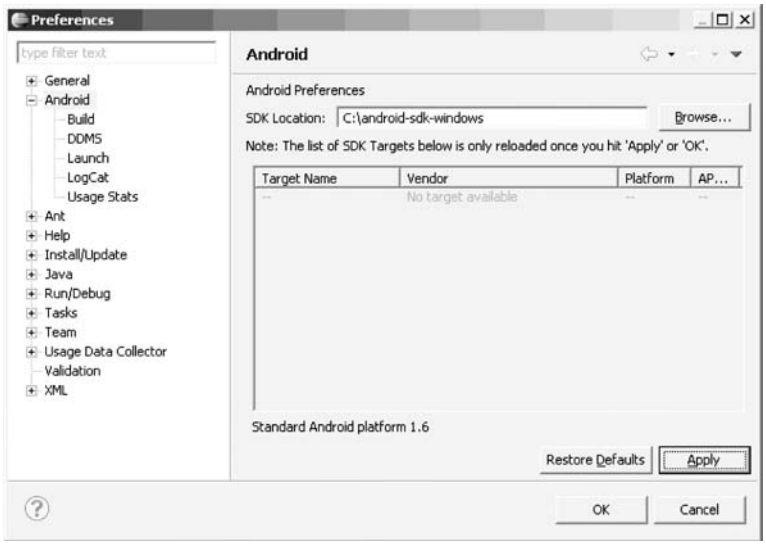
W tym momencie aplikacja Eclipse zacznie pobierać i instalować narzędzia ADT. Żeby nowe wtyczki pojawiły się w oknie Eclipse, należy ponownie uruchomić aplikację.

W przypadku posiadania starszej wersji narzędzi ADT należy otworzyć menu *Help* i wybrać opcję *Check for Updates*. Powinna zostać wyświetlona aktualna wersja wtyczek ADT, których instalacja przebiega od punktu 3. powyższych instrukcji.



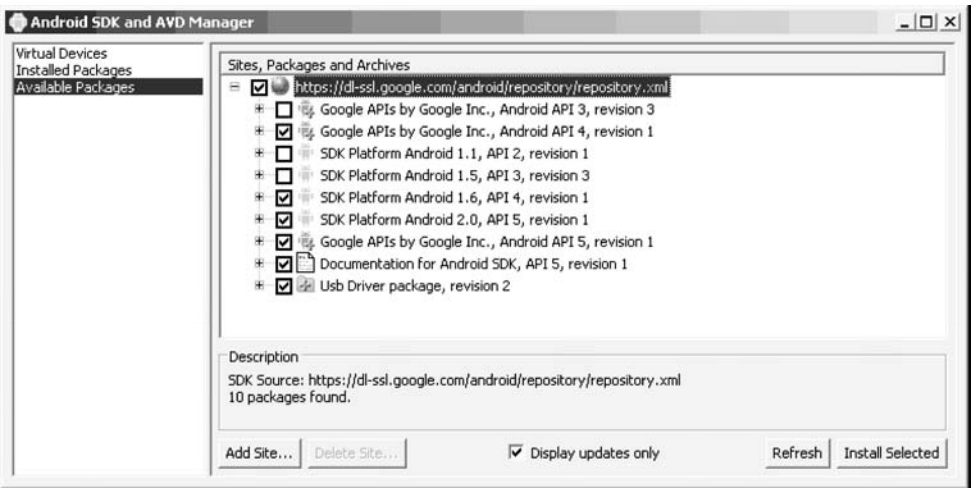
Rysunek 2.2. Instalacja narzędzi ADT za pomocą funkcji Install New Software w środowisku Eclipse

Ostatnim etapem aktywacji narzędzi ADT w obrębie środowiska Eclipse jest odniesienie ich do zestawu Android SDK. W tym celu należy otworzyć menu *Window* i wybrać opcję *Preferences* (w systemie Mac OS X opcja ta jest dostępna w menu *Eclipse*). W oknie dialogowym Preferences należy wybrać węzeł *Android* i wpisać ścieżkę katalogu Android SDK (rysunek 2.3), a następnie kliknąć przycisk *Apply*. Może pojawić się w międzyczasie okno dialogowe, w którym można zaznaczyć opcję wysyłania do firmy Google statystyk dotyczących wykorzystania programu Android SDK. Wybór należy do Czytelnika. Teraz wystarczy kliknąć *OK*, żeby zamknąć okno Preferences.



Rysunek 2.3. Powiązanie narzędzi ADT z zestawem Android SDK

Świeżo zainstalowana wersja pakietu Android SDK nie posiada żadnych wersji platform. Gdyby było inaczej, byłyby one widoczne w zakładce Android (rysunek 2.3) po wskazaniu jego lokalizacji. Instalowanie platform jest bardzo proste. W programie Eclipse należy wybrać *Window/Android SDK and AVD Manager*, kliknąć element *Available Packages*, zaznaczysz adres źródła *https://dl-ssl.google.com/android/repository/repository.xml*, a następnie określić platformy oraz dodatki, które mają zostać zainstalowane (na przykład *Android 2.0*). Zostało to zaprezentowane na rysunku 2.4.



Rysunek 2.4. Dodawanie platform do środowiska Android SDK

Teraz wystarczy kliknąć przycisk *Install Selected*. Trzeba zatwierdzić każdy element, zaznaczając opcję *Accept*[1]. Wybrane pakiety i dodatki zostaną pobrane i wdrożone do środowiska Eclipse przez wtyczkę ADT. Dodatki *Google APIs* służą do projektowania aplikacji wykorzystujących Google Maps. Istnieje możliwość podejrzenia zainstalowanych dodatków po kliknięciu opcji *Installed Packages*, widocznej na rysunku 2.4 w lewym górnym rogu ekranu.

Już niemal nadszedł czas na zapoznanie się z pierwszą aplikacją na Androida — najpierw jednak musimy przejrzeć podstawowe pojęcia odnoszące się do aplikacji tworzonych na ten system.

Przedstawienie podstawowych składników

Szkielet każdej aplikacji zawiera pewne kluczowe składniki, z którymi muszą zapoznać się projektanci, zanim zaczną pisać programy oparte na tym szkielecie. Na przykład do napisania aplikacji w środowisku J2EE (Java 2 Platform Enterprise Edition) wymagana jest znajomość technologii JSP (JavaServer Pages) oraz serwletów. Podobnie w przypadku aplikacji pisanych na platformę Android — należy znać pojęcia aktywności, widoków, intencji, dostawców treści, usług oraz przeznaczenie pliku *AndroidManifest.xml*. W tym podrozdziale omówimy krótko każde z tych pojęć, przedstawiając bardziej szczegółowe informacje w kolejnych rozdziałach.

[1] Lub zaakceptować wszystkie jednocześnie, zaznaczając *Accept All — przyp. tłum*

Widok

Widoki są elementami interfejsu użytkownika tworzącymi jego podstawowe bloki budulcowe. Wykorzystują model hierarchiczny oraz posiadają zaimplementowane informacje, dzięki którym są poprawnie wyświetlane. Widok może przybrać kształt przycisku, etykiety, pola tekstowego oraz wielu innych składników interfejsu UI. Jeżeli ktoś wie, czym są widoki w platformach J2EE oraz Swing, szybko pojmie widoki w Androidzie.

Aktywność

Aktywność jest pojęciem interfejsu użytkownika. Aktywność przeważnie jest reprezentacją pojedynczego ekranu aplikacji. Zazwyczaj zawarty jest w niej przynajmniej jeden widok, ale niekoniecznie tak musi być. Poza tym inne elementy mogą lepiej przedstawiać aktywność niezawierającą widoku (zostanie to omówione w ustępie „Usługa").

Intencja

Generalnie intencja definiuje „intencję", „zamiar" wykonania jakiejś pracy. W terminie tym mieści się kilka pojęć, więc najlepszym sposobem na jego zrozumienie jest wykorzystanie intencji w praktyce. Intencje są wykorzystywane w następujących celach:

- nadawanie komunikatu,
- uruchamianie usługi,
- rozpoczynanie aktywności,
- wyświetlanie strony WWW lub listy kontaktów,
- wybieranie lub odbieranie połączenia telefonicznego.

Intencje nie zawsze są inicjowane przez aplikację — są także wykorzystywane przez system do powiadamiania aplikacji o określonych zdarzeniach (na przykład o otrzymaniu wiadomości tekstowej).

Intencje można podzielić na jawne oraz niejawne. Jeżeli zostanie wyraźnie określone, że adres URL ma być widoczny, system automatycznie zdecyduje, jaki składnik będzie dotyczył intencji. Istnieje także możliwość określenia konkretnej informacji, w jaki sposób powinna być potraktowana intencja. Intencje luźno łączą działanie z jego uchwytem.

Dostawca treści

Współdzielenie danych pomiędzy aplikacjami urządzenia przenośnego jest powszechnie stosowaną praktyką. Android definiuje więc standardowy mechanizm dzielenia danych (takich jak listy kontaktów) wśród aplikacji bez konieczności odsłaniania podstawowych magazynów, struktury oraz implementacji. Dzięki dostawcom treści można ujawniać dane oraz pozwalać jednym aplikacjom korzystać z zasobów innych programów.

Usługa

Usługi Androida podobne są do usług obecnych w systemie Windows lub na innych platformach — są to procesy działające w tle, przeznaczone do trwania przez długi okres czasu. W Androidzie są zdefiniowane dwa rodzaje usług: usługi lokalne oraz usługi zdalne. Usługi lokalne są elementami dostępnymi wyłącznie dla aplikacji je obsługującej. Z drugiej strony, usługi zdalne są przeznaczone dla innych aplikacji, łączących się z nimi w sposób zdalny.

Przykładem usługi jest składnik, wykorzystywany przez aplikację pocztową do sprawdzania, czy pojawiły się nowe wiadomości. Usługa ta jest lokalna, jeżeli nie jest używana przez inne aplikacje znajdujące się w urządzeniu. Jeżeli korzysta z niej kilka usług, można ją zaimplementować w formie usługi zdalnej. Jak zostanie wyjaśnione w rozdziale 8., jest to związane z różnicą pomiędzy funkcjami `startService ()` oraz `bindService ()`.

Istnieje możliwość stosowania istniejących usług, jak również pisania własnych za pomocą rozszerzania klasy `Service`.

AndroidManifest.xml

Plik *AndroidManifest.xml*, podobny do pliku *web.xml* w świecie J2EE, określa zawartość oraz zachowanie aplikacji. Na przykład znajduje się w nim lista aktywności oraz usług danej aplikacji, a także uprawnienia wymagane do jej uruchomienia.

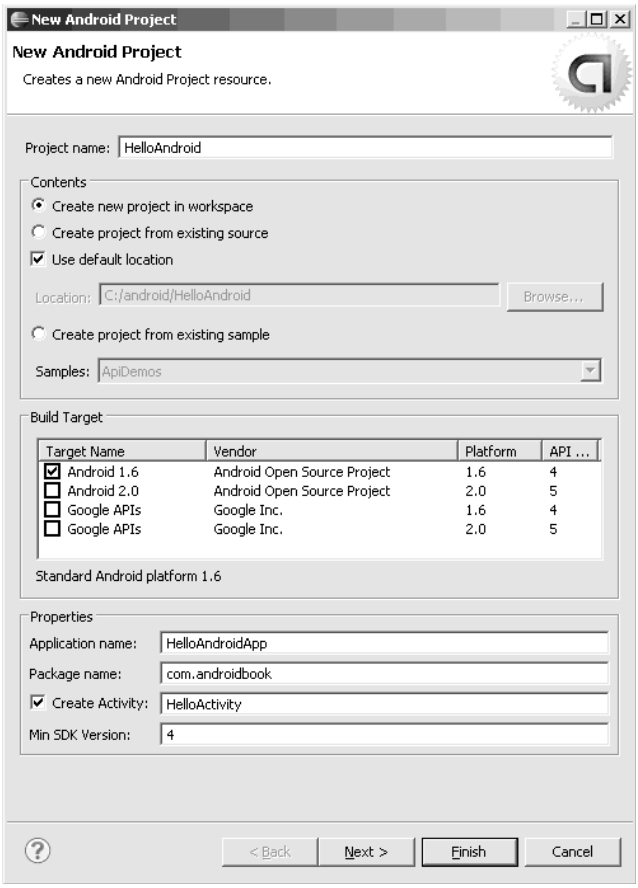
Urządzenia AVD

Urządzenie AVD (ang. *Android Virtual Device* — wirtualne urządzenie Androida) pozwala programistom na przetestowanie aplikacji bez konieczności połączenia się z rzeczywistym urządzeniem. Można tworzyć różne konfiguracje urządzeń AVD, zdolne do emulowania różnych modeli istniejących telefonów.

Witaj Świecie!

Teraz możemy rozpocząć pisanie pierwszej aplikacji na Androida. Na początek stworzymy prosty program „Hello World!". Szkielet aplikacji zbudujemy w następujący sposób:

1. Uruchom środowisko *Eclipse* i wybierz *File/New/Project…*. W oknie dialogowym New Project otwórz węzeł *Android*, a następnie wybierz opcję *Android Project*, po czym kliknij przycisk *Next*. Ujrzysz okno New Android Project, zaprezentowane na rysunku 2.5. Być może dostęp do projektu Android został umieszczony w menu *New*, dzięki czemu można nieco szybciej otwierać nowe projekty. Jeżeli istnieje taka możliwość, możesz również skorzystać z przycisku *New Android Project* w pasku narzędzi.
2. Wpisz, zgodnie z rysunkiem 2.5, nazwę projektu `HelloAndroid`, nazwę aplikacji `HelloAndroidApp`, nazwę pakietu `com.androidbook` oraz `HelloActivity` jako nową aktywność. Należy pamiętać, że w przypadku prawdziwej aplikacji warto podawać przemyślaną nazwę, gdyż będzie się ona pojawiać w pasku tytułowym programu. Warto również zauważyć, że domyślna lokalizacja projektu związana jest z lokalizacją przestrzeni roboczej środowiska Eclipse. W naszym przypadku przestrzenią roboczą jest *C:\android*, a kreator nowego projektu dodaje w niej folder z nazwą projektu, zatem ostatecznie otrzymujemy *C:\android\HelloAndroid*. Dzięki wartości `4` w polu *Min SDK Version* Android „wie", że aplikacja wymaga co najmniej wersji 1.6 systemu operacyjnego.
3. Kliknij przycisk *Finish*, dzięki czemu narzędzia ADT wygenerują szkielet projektu. Teraz otwórz plik *HelloActivity.java* w folderze *src* i zmodyfikuj metodę `onCreate ()` w następujący sposób:



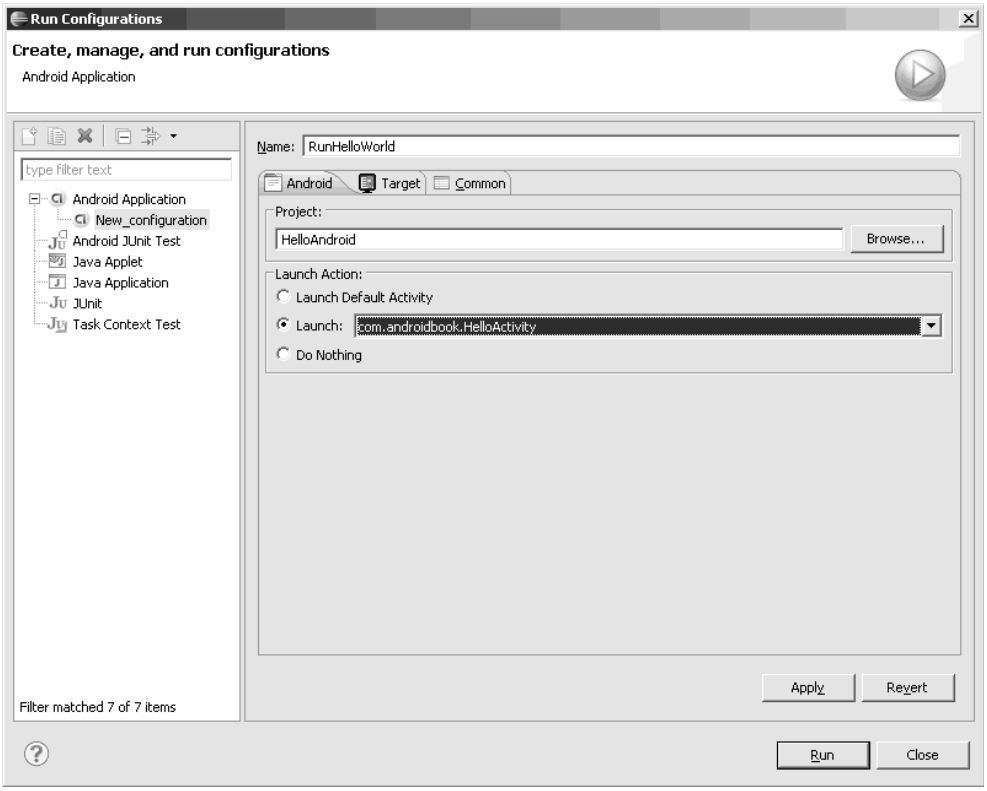
Rysunek 2.5. Okno kreatora New Android Project

```
/** Called when activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /** tworzy deklarację widoku TextView i wyświetla napis „Witaj Świecie!" */
        TextView tv = new TextView(this);
        tv.setText("Witaj Świecie!");
        /** przyłącza widok treści do deklaracji widoku TextView */
        setContentView(tv);
    }
```

Program Eclipse powinien automatycznie dodać instrukcję *import* dla klasy `android.widget.TextView`. Żeby zobaczyć wszystkie klasy, należy kliknąć znak +, widniejący przy pierwszej instrukcji *import*. Jeżeli ta instrukcja nie zostanie automatycznie dodana, należy wprowadzić ją samodzielnie. Teraz wystarczy zapisać plik *HelloActivity.java*.

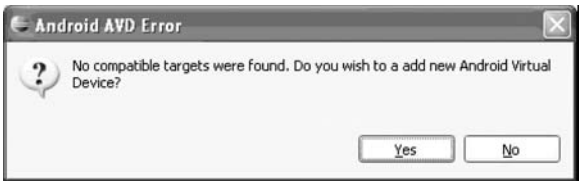
Żeby uruchomić aplikację, należy utworzyć konfigurację uruchomieniową środowiska Eclipse; będzie też potrzebne wirtualne urządzenie, na którym aplikacja zostanie przetestowana. Szybko opiszemy wymagane czynności, a następnie zajmiemy się bardziej szczegółowo urządzeniami AVD. Konfigurację uruchomieniową tworzy się w następujący sposób:

1. Wybierz opcję *Run/Run Configurations…*.
2. W oknie dialogowym *Run Configurations* kliknij dwukrotnie opcję *Android Application*, znajdującą się w panelu po lewej stronie. Kreator utworzy nową konfigurację nazwaną *New Configuration*.
3. Zmień nazwę tej konfiguracji na `RunHelloWorld`.
4. Kliknij przycisk *Browse…* i zaznacz projekt *HelloAndroid*.
5. W części ekranu nazwanej Launch Action zaznacz opcję *Launch* i wybierz z rozwijanej listy *com.androidbook.HelloActivity*. Ekran powinien wyglądać podobnie, jak na rysunku 2.6.

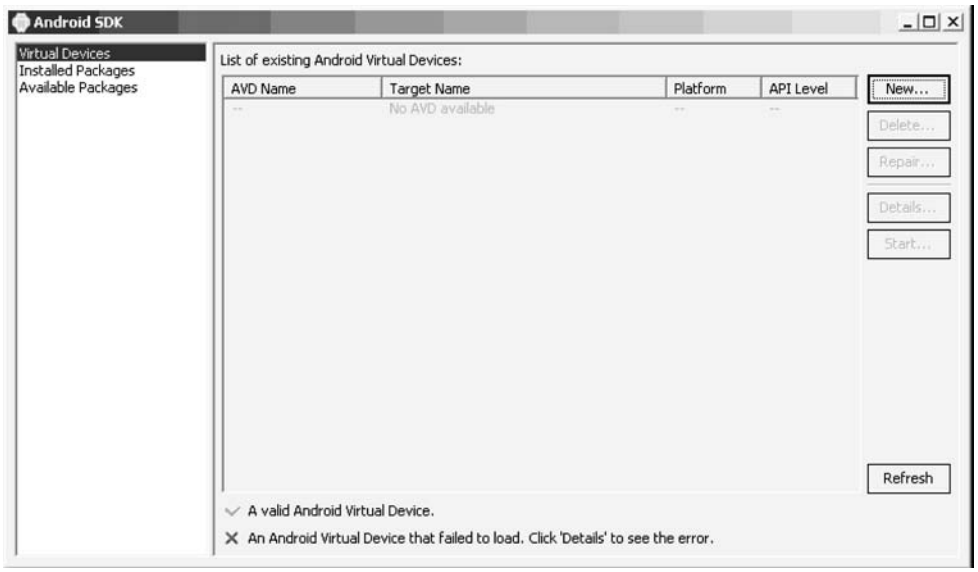


Rysunek 2.6. Tworzenie konfiguracji uruchomieniowej środowiska Eclipse, pozwalającej na uruchomienie aplikacji „Witaj Świecie!"

6. Kliknij *Apply*, a następnie *Run*. Już niemal gotowe. Środowisko Eclipse jest gotowe do uruchomienia aplikacji, ale potrzebuje jeszcze urządzenia, na którym zostanie ona sprawdzona. Pojawi się okno z ostrzeżeniem, jak na rysunku 2.7, że nie zostały znalezione kompatybilne urządzenia. Kliknij *Yes*, żeby stworzyć własne urządzenie.
7. Zostaniesz przeniesiony do ekranu wyświetlającego listę dostępnych urządzeń AVD (rysunek 2.8). Zwróć uwagę, że mamy do czynienia z tym samym oknem, co przedstawione na rysunku 2.4. Musisz tu dodać urządzenie pasujące do aplikacji. Kliknij przycisk *New…*.



Rysunek 2.7. Program Eclipse ostrzega, że nie istnieją kompatybilne urządzenia, oraz pyta, czy stworzyć własne urządzenie

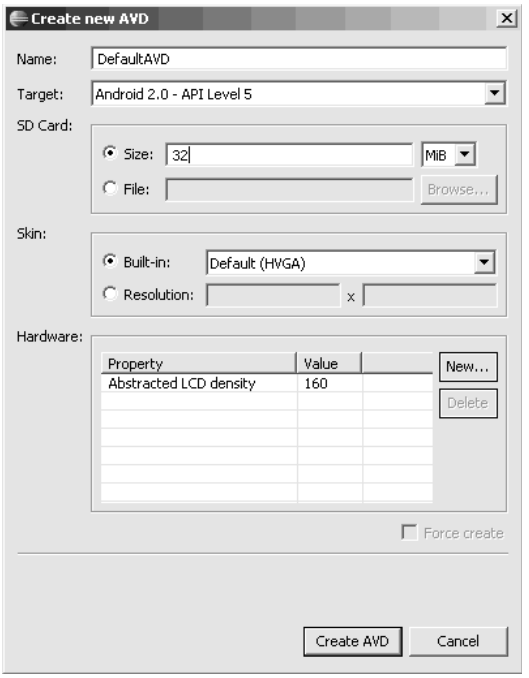


Rysunek 2.8. Okno listy istniejących urządzeń AVD

8. Wypełnij pola w oknie Create AVD, tak jak zostało pokazane na rysunku 2.9. Podaj nazwę urządzenia *DefaultAVD*, jako docelowy system wybierz *Android 2.0.1 — API Level 6* z listy *Target*, ustaw rozmiar pamięci karty na 32 MB i pozostaw bez zmian domyślną skórę urządzenia. Kliknij *Create AVD*. Program Eclipse powiadomi Cię o udanym utworzeniu urządzenia AVD. Kliknij *OK*, aby zamknąć okno środowiska Android SDK.

Uwaga! Wybraliśmy nowszą wersję środowiska SDK dla naszego urządzenia AVD, ale równie dobrze można skorzystać ze starszej wersji. Urządzenia AVD oparte na nowszym zestawie SDK współpracują z aplikacjami napisanymi w starszym środowisku programistycznym. Odwrotna możliwość nie wchodzi oczywiście w rachubę: aplikacja wymagająca nowszego środowiska SDK nie zadziała na urządzeniu AVD opartym na starszej wersji SDK.

9. Teraz wybierz utworzone urządzenie AVD z listy. Zwróć uwagę, że po kliknięciu przycisku *Refresh* lista będzie odświeżana. Kliknij *OK*.
10. Zostanie uruchomiona Twoja pierwsza aplikacja na emulatorze!



Rysunek 2.9. Konfigurowanie wirtualnego urządzenia AVD

Uwaga! Emulacja rozruchu urządzenia może zająć emulatorowi kilka minut. Gdy system operacyjny zostanie załadowany, powinna pojawić się aplikacja *HelloAndroidApp* na wirtualnym ekranie, co zostało przedstawione na rysunku 2.10. Należy też mieć świadomość, że emulator uruchamia w tle również inne aplikacje, więc co jakiś czas może wyskakiwać informacja o błędzie lub ostrzeżenie. Jeżeli pojawi się komunikat o błędzie, zazwyczaj można go zignorować i kazać emulatorowi przejść do kolejnego etapu rozruchu. Na przykład jeżeli pojawi się informacja „`application abc is not responding`" („aplikacja abc przestała odpowiadać"), można albo zaczekać na jej uruchomienie, albo po prostu zmusić emulator do jej zamknięcia. Zasadniczo warto poczekać i pozwolić, żeby emulator uruchomił się bez błędów.

Wiadomo już, w jaki sposób stworzyć nową aplikację w Androidzie oraz jak ją uruchomić na emulatorze. Teraz przyjrzymy się uważniej urządzeniom AVD, po czym zagłębimy się w świat artefaktów oraz struktury aplikacji Androida.

Wirtualne urządzenia AVD

Wirtualne urządzenie Androida (ang. *Android Virtual Device* — AVD) reprezentuje konfigurację wybranego modelu urządzenia. Na przykład można utworzyć urządzenie AVD symbolizujące telefon starszego rodzaju, działający zgodnie z wersją 1.5 środowiska SDK oraz posiadający kartę SD 32 MB. Cała koncepcja oparta jest na możliwości tworzenia urządzeń AVD obsługujących tworzone aplikacje oraz emulowania tych urządzeń w celu projektowania i testowania aplikacji. Definiowanie (oraz zmienianie) urządzeń AVD jest bardzo łatwym procesem do przeprowadzenia oraz umożliwia błyskawiczne testowanie aplikacji

Rysunek 2.10. Aplikacja HelloAndroidApp uruchomiona na emulatorze

w różnych konfiguracjach. W poprzednim podrozdziale został przedstawiony sposób tworzenia urządzenia AVD w środowisku Eclipse. Można stworzyć większą ilość urządzeń AVD, klikając *Window/Android SDK and AVD Manager*, a następnie wybierając węzeł *Virtual Devices* w panelu po lewej stronie ekranu. Poniżej został także opisany sposób tworzenia tych urządzeń z poziomu wiersza poleceń.

Do stworzenia urządzenia AVD wykorzystywany jest plik wsadowy *android*, umieszczony w katalogu *tools* (*c:\android-sdk-windows\tools*). Dzięki temu plikowi możliwe jest także zarządzanie utworzonymi urządzeniami AVD. Można je na przykład przeglądać oraz przenosić. Spis poleceń dostępnych dzięki plikowi *android* zostaje wyświetlony po wpisaniu w wierszu polecenia *android –help*. Na razie stwórzmy urządzenie AVD.

Pliki urządzeń AVD domyślnie są przechowywane w katalogu profilu użytkownika (na wszystkich platformach), w folderze *.android\AVD*. Znajduje się tam urządzenie AVD, stworzone do uruchomienia aplikacji „Hello World!". Istnieje również możliwość przeniesienia (lub edytowania) urządzeń AVD do innej lokalizacji. Stwórzmy teraz folder, w którym będzie przechowywany obraz naszego urządzenia AVD, na przykład *c:\avd*. Kolejnym etapem jest uruchomienie pliku *android* w celu wygenerowania urządzenia AVD. Należy otworzyć wiersz poleceń i wpisać następującą komendę (Czytelnik wprowadza własną ścieżkę, na końcu której przechowywane będą pliki AVD, oraz wartość argumentu *t* — w oparciu o wersję zainstalowanego środowiska SDK):

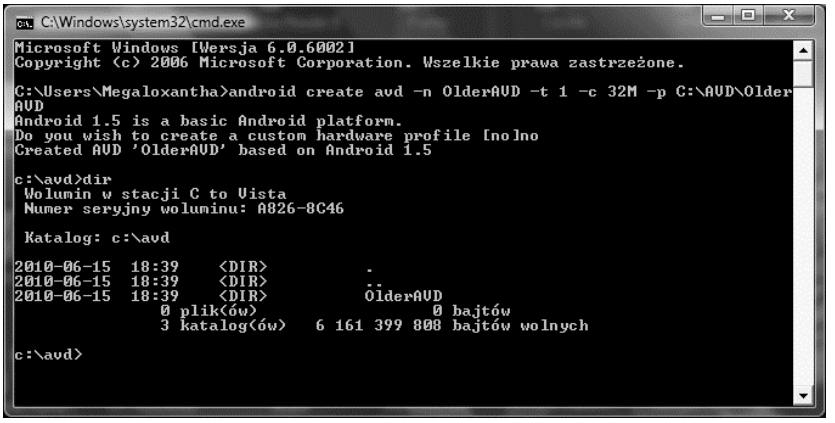
```
android create avd -n OlderAVD -t 2 -c 32M -p C:\AVD\OlderAVD\
```

W tabeli 2.1 zostały objaśnione parametry narzędzia *android*.

Tabela 2.1. Parametry przypisane do pliku *android.bat*

Argument/polecenie	Opis
`create avd`	Polecenie utworzenia urządzenia AVD.
`n`	Nazwa urządzenia AVD.
`t`	Wersja środowiska SDK. Wartość *1* oznacza Android 1.1, *2* to Android 1.5, *3* reprezentuje Android 1.6 itd.
`c`	Pojemność karty SD wyrażona w bajtach. Wartość *K* oznacza kilobajty, a *M* — megabajty.
`p`	Ścieżka tworzonego urządzenia. Ten argument nie jest wymagany.

Wykonanie powyższego polecenia spowoduje wygenerowanie pliku urządzenia AVD; powinien zostać wyświetlony ekran, podobny do pokazanego na rysunku 2.11. Warto zwrócić uwagę, że po wpisaniu polecenia *create avd* system zapyta, czy utworzyć niestandardowy profil sprzętowy. Na razie wpiszmy *No*, dobrze jest jednak wiedzieć, że po udzieleniu odpowiedzi *Yes* stanie się dostępnych wiele opcji konfiguracji urządzenia AVD, takich jak rozmiar ekranu, obecność aparatu i tak dalej.



Rysunek 2.11. Ekran wynikowy utworzenia urządzenia AVD za pomocą pliku android.bat

Nawet jeżeli została określona alternatywna ścieżka dla pliku *OlderAVD* w programie *android.bat*, istnieje także kopia pliku *OlderAVD.ini* w folderze macierzystym *.android\AVD*. Jest to przemyślane działanie, dzięki któremu po kliknięciu *Window/Android SDK and AVD Manager* w środowisku Eclipse dostępne będą wszystkie urządzenia AVD, także utworzone w wierszu poleceń.

Spójrzmy ponownie na rysunek 2.5. W przypadku aplikacji „Hello World!" wybraliśmy system Android 1.6, co jest równoznaczne z wybraniem co najmniej wersji 4 środowiska Android SDK. Jeżeli zostanie wybrana wersja systemu Android 1.5 (przy założeniu, że została zainstalowana), wartość minimalnej wersji środowiska SDK wyniesie 3, dla Androida 2.0 będzie miała wartość 5.

Należy również mieć na uwadze, że wybranie któregoś z interfejsów *Google API* z listy *SDK Target* da dostęp do funkcji korzystania z map w aplikacji, podczas gdy wybranie interfejsu *Android 1.5* lub późniejszego nie zapewni takiej możliwości. W wersjach środowiska SDK

wcześniejszych niż 1.5 klasy powiązane z mapami były umieszczone w pliku *android.jar*, jednak od tamtego czasu zostały przeniesione do oddzielnego pliku *maps.jar*. Po wybraniu jednego z interfejsów Google API domyślną wartością Min SDK Version będzie 5 (dla systemu Android 2.0) lub 4 (Android 1.6) i tak dalej, natomiast narzędzia ADT umieszczą plik *maps.jar* w projekcie. Innymi słowy, jeżeli aplikacja będzie wymagała klas związanych z mapami, należy wybrać jedną z wersji Google API na liście SDK Target. Nadal trzeba będzie umieścić wpis dotyczący korzystania z map (`<uses-libraryandroid:name="com.google.android.maps"/>`) w pliku *AndroidManifest.xml*. Szczegółowe informacje na ten temat można znaleźć w rozdziale 7.

Poznanie struktury aplikacji Androida

Chociaż różne aplikacje Androida będą się różniły rozmiarami oraz złożonością, ich struktura będzie podobna. Na rysunku 2.12 została przedstawiona struktura stworzonej niedawno aplikacji „Hello World!".

Rysunek 2.12. Struktura aplikacji „Witaj Świecie!"

Aplikacje na Androida składają się z elementów niezbędnych oraz opcjonalnych. W tabeli 2.2 wymienione zostały składniki aplikacji tworzonej na Androida.

Jak zostało pokazane w tabeli 2.2, aplikacja systemu Android składa się z trzech zasadniczych elementów: deskryptora aplikacji, zbioru zasobów oraz kodu źródłowego aplikacji. Jeżeli zignorować na chwilę plik *AndroidManifest.xml*, można zauważyć prostotę aplikacji: zostaje umieszczona logika biznesowa w formie kodu, a cała reszta to zasoby. Taka nieskomplikowana struktura przypomina szkielet aplikacji J2EE, w którym zasobom odpowiadają strony JSP, logice biznesowej — serwlety, a odpowiednikiem pliku *AndroidManifest.xml* jest plik *web.xml*.

Tabela 2.2. Elementy składowe aplikacji systemu Android

Element składowy	Opis	Wymagany?
AndroidManifest.xml	Plik deskryptora aplikacji. Są w nim zdefiniowane aktywności, dostawcy usług, usługi oraz adresaci intencji, czyli elementy związane z daną aplikacją. Można w nim również zadeklarować uprawnienia wymagane przez aplikację, a także przydzielić określone uprawnienia dla innych aplikacji, korzystających z usług danego programu. Ponadto może być tu zamieszczona instrumentacja, wykorzystywana do testowania danej aplikacji lub innych programów.	Tak
src	Folder przechowujący kod źródłowy aplikacji.	Tak
assets	Luźny zbiór plików i folderów.	Nie
res	Folder zawierający zasoby aplikacji. Jest to folder nadrzędny wobec węzłów *drawable*, *anim*, *layout*, *menu*, *values*, *xml* oraz *raw*.	Tak
drawable	Folder mieszczący w sobie pliki obrazów lub deskryptorów obrazów używanych przez aplikację.	Nie
anim	W folderze tym są umieszczone pliki deskryptora napisane w języku XML, opisujące animacje wykorzystywane przez aplikację.	Nie
layout	Mieszczą się tu widoki aplikacji. Bardziej opłaca się tworzenie widoków poprzez deskryptory języka XML, niż poprzez pisanie kodu.	Nie
menu	Folder zawierający pliki deskryptorów list menu aplikacji.	Nie
values	Przechowywane są w nim pozostałe zasoby wykorzystywane przez aplikację. Wszystkie znajdujące się tu zasoby są opisane za pomocą deskryptorów XML. Przykładowymi zasobami mogą być ciągi znaków, style oraz kolory.	Nie
xml	Znajdują się tu dodatkowe pliki XML wykorzystywane przez aplikację.	Nie
raw	Folder z dodatkowymi danymi — prawdopodobnie nieopisanymi w języku XML — wymaganymi przez aplikację.	Nie

Można również porównać modele projektowania w środowiskach J2EE oraz Android. W przypadku J2EE widoki są budowane za pomocą języka znaczników. W Androidzie wykorzystano tę samą filozofię, ale stosowanym językiem jest XML. Jest to korzystne rozwiązanie, gdyż nie ma konieczności wplatania widoku do głównego kodu; można zmieniać wygląd i zachowanie aplikacji poprzez edytowanie znaczników.

Należy również pamiętać o kilku ograniczeniach dotyczących zasobów. Po pierwsze, Android obsługuje jedynie liniową listę plików, znajdującą się w predefiniowanych plikach umieszczonych w folderze *res*. Na przykład nie widzi zagnieżdżonych folderów, znajdujących się w katalogu *layout* (tak samo w przypadku pozostałych folderów podrzędnych do folderu *res*). Po drugie, istnieją pewne podobieństwa pomiędzy folderem *assets* oraz folderem *raw*, umieszczonym w katalogu *res*. W obydwu katalogach mogą być przechowywane nieskompresowane pliki, ale dane znajdujące się w folderze *raw* są uznawane za zasoby, a w folderze *assets* już nie. Zatem pliki z katalogu *raw* będą zlokalizowane, dostępne poprzez identyfikatory zasobów i tak dalej. Jednak informacje znajdujące się w katalogu *assets* są traktowane jako dane ogólnego przeznaczenia, pozbawione ograniczeń oraz obsługi zasobów. Warto zwrócić na to uwagę, gdyż pozbawienie danych znajdujących się w katalogu *assets* miana zasobów umożliwia stworzenie własnej hierarchii plików i folderów w jego wnętrzu (więcej informacji na temat zasobów znajduje się w rozdziale 3.).

Uwaga! Dosyć wyraźnie widać, że w Androidzie jest całkiem często stosowany język XML. Powszechnie wiadomo, że jest to dość rozbudowany język, rodzi się zatem pytanie: czy ma sens korzystanie z niego, gdy celem jest urządzenie posiadające ograniczone zasoby? Okazuje się, że kod XML, używany podczas projektowania aplikacji, jest w rzeczywistości kompilowany do kodu binarnego przy użyciu narzędzia AAPR (ang. *Android Asset Packaging Tool* — narzędzie pakowania zasobów Androida). Zatem podczas instalowania aplikacji na urządzeniu pliki są konwertowane i przechowywane w formie kodu binarnego. Podczas uruchomienia plik jest odczytywany w tej formie i nie jest konwertowany ponownie do pliku XML. Ta metoda łączy zalety obydwu technologii — można pracować z językiem XML i nie martwić się o ilość cennych zasobów urządzenia.

Analiza aplikacji Notepad

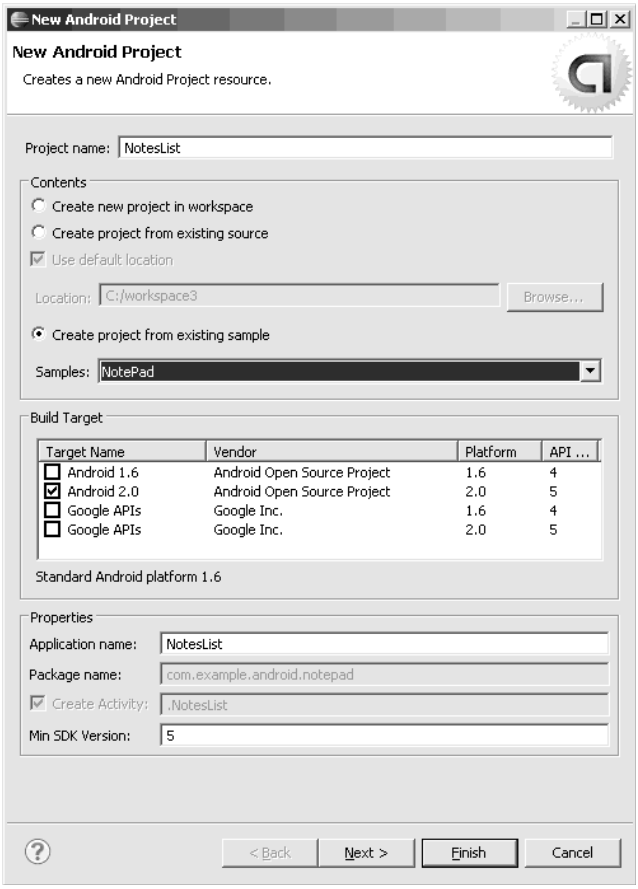
Do tej pory nie tylko pokazaliśmy, w jaki sposób stworzyć oraz uruchomić w emulatorze nową aplikację na Androida, lecz staraliśmy się, żeby Czytelnik zrozumiał elementy jej struktury. Teraz przyjrzymy się aplikacji Notepad, umieszczonej w pakiecie Android SDK. Jej poziom złożoności plasuje się pomiędzy naszą aplikacją „Hello World!" a w pełni rozwiniętą aplikacją na Androida, zatem analiza jej składników pozwoli niejako pojąć realny proces projektowania w środowisku SDK.

Wczytanie oraz uruchomienie aplikacji Notepad

W tym ustępie wyjaśnimy, w jaki sposób załadować aplikację Notepad w środowisku Eclipse i uruchomić ją na emulatorze. Przed rozpoczęciem należy wiedzieć, że w aplikacji Notepad zaimplementowano kilka różnych opcji. Użytkownik może na przykład stworzyć nową notatkę, edytować istniejącą notatkę, usunąć ją, przejrzeć listę notatek i tak dalej. Kiedy aplikacja zostanie uruchomiona, nie będzie w niej żadnych zapisanych notatek, więc użytkownik ujrzy pustą listę. Po wciśnięciu przycisku *Menu* zostanie wyświetlona lista czynności, wśród nich opcja dodania nowej notatki. Po utworzeniu nowego pliku można go edytować lub usunąć za pomocą odpowiedniej opcji.

Żeby wczytać przykładową aplikację Notepad w środowisku Eclipse, należy wykonać następujące czynności:

1. Uruchom program Eclipse.
2. Otwórz *File/New/Project*.
3. W oknie dialogowym New Project wybierz *Android/Android Project* i kliknij *Next*.
4. W następnym oknie wpisz *NotesList* jako nazwę projektu, wybierz opcję *Create project from existing sample*, następnie zaznacz pole *Android 2.0.1* na liście *Build Target*, a z rozwijanej listy wybierz aplikację *Notepad*. Zwróć uwagę, że jest ona umiejscowiona w folderze *platforms\android-2.0\samples* pakietu Android SDK, który wcześniej pobrałeś. Po wybraniu tej aplikacji zostanie automatycznie odczytany plik *AndroidManifest.xml* i zostaną wypełnione pozostałe pola w tym oknie dialogowym (rysunek 2.13).



Rysunek 2.13. Tworzenie aplikacji Notepad

5. Kliknij przycisk *Finish*.

Teraz powinna być dostępna aplikacja *Notepad* w środowisku Eclipse. Jeżeli zostaną wyświetlone jakieś informacje o problemach związanych z tym projektem, można spróbować użyć opcji *Clean* z menu *Project*, aby je wyczyścić. Żeby uruchomić aplikację, można utworzyć aplikację uruchomieniową (podobnie jak to zrobiliśmy przy okazji programu „Hello World!") lub wystarczy kliknąć prawym przyciskiem ikonę projektu, wybrać opcję *Run As*,

a następnie *Android Application*. Zostanie uruchomiony emulator i zainstalowana na nim aplikacja. Po wczytaniu emulatora (można poznać, że zostanie wczytany po wyświetleniu daty i godziny na środku jego ekranu) wystarczy wcisnąć przycisk *Menu*, żeby została wyświetlona aplikacja *Notepad*. Aby się z nią zaznajomić, można po niej pomyszkować przez kilka minut.

Rozłożenie kodu na czynniki pierwsze

Przyjrzyjmy się teraz strukturze aplikacji (rysunek 2.14).

Rysunek 2.14. Struktura aplikacji Notepad

Jak widać, program zawiera kilka plików *.java*, obrazów *.png*, trzy widoki (w folderze *layout*) oraz plik *AndroidManifest.xml*. Gdyby to była aplikacja wiersza poleceń, należałoby szukać pliku, w którym jest umieszczona metoda *Main*. Zatem co jest odpowiednikiem metody *Main* w Androidzie?

W środowisku Android jest definiowana początkowa aktywność, zwana także aktywnością szczytowego poziomu. Jeżeli przyjrzeć się zawartości pliku *AndroidManifest.xml*, można tam znaleźć jednego dostawcę oraz trzy aktywności. Aktywność *NotesList* wyznacza filtr intencji dla akcji *android.intent.action.MAIN*, a także dla kategorii *android.intent.category.LAUNCHER*.

Po uruchomieniu aplikacji Androida zostaje ona wczytana przez urządzenie i jest odczytywany plik *AndroidManifest.xml*. Zostają wyszukane i uruchomione aktywności posiadające filtr intencji, który składa się z aktywności *MAIN* oraz kategorii *LAUNCHER*, tak jak pokazano poniżej.

```
<intent-filter>
   <action android:name="android.intent.action.MAIN" />
   <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Po odnalezieniu właściwej aktywności urządzenie musi powiązać ją z rzeczywistą klasą. Dokonuje tego poprzez połączenie nazwy głównego pakietu z nazwą aktywności, w naszym wypadku będzie to *com.example.android.notepad.NotesList* (listing 2.1).

Listing 2.1. Plik AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.android.notepad"
>
   <application android:icon="@drawable/app_notes"
      android:label="@string/app_name"
   >
      <provider android:name="NotePadProvider"
         android:authorities="com.google.provider.NotePad"
      />
      <activity android:name="NotesList" android:label="@string/title_notes_list">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
         <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <action android:name="android.intent.action.EDIT" />
            <action android:name="android.intent.action.PICK" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
         </intent-filter>
         <intent-filter>
            <action android:name="android.intent.action.GET_CONTENT" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
         </intent-filter>
      </activity>
...
</manifest>
```

Nazwa głównego pakietu aplikacji jest zdefiniowana jako atrybut elementu `<manifest>` w pliku *AndroidManifest.xml*, a każda aktywność posiada atrybut nazwy.

Po określeniu początkowej aktywności zostaje ona uruchomiona oraz ulega wywołaniu metoda `onCreate()`. Przyjrzyjmy się elementowi `NotesList.onCreate()`, przedstawionemu w listingu 2.2.

Listing 2.2. Metoda onCreate

```
public class NotesList extends ListActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);

         setDefaultKeyMode(DEFAULT_KEYS_SHORTCUT);
         Intent intent = getIntent();
         if (intent.getData() == null) {
            intent.setData(Notes.CONTENT_URI);
         }

         getListView().setOnCreateContextMenuListener(this);

         Cursor cursor = managedQuery(getIntent().getData(),PROJECTION, null, null,
               Notes.DEFAULT_SORT_ORDER);

          SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
          ↪R.layout.noteslist_item, cursor, new String[] { Notes.TITLE },
new int[] { android.R.id.text1 });
         setListAdapter(adapter);
   }
}
```

Aktywności w Androidzie są przeważnie uruchamiane przez intencje, a także przez inne aktywności. Metoda `onCreate()` sprawdza, czy intencja bieżącej aktywności zawiera dane (notatki). Jeżeli nie zawiera, zostaje ustanowiony identyfikator URI, dzięki któremu zostają pobrane dane. W rozdziale 3. zademonstrujemy, że Android uzyskuje dostęp do danych poprzez dostawców treści korzystających z identyfikatorów URI. W tym przypadku identyfikator URI dostarcza wystarczająco wiele informacji, żeby pobrać dane z bazy danych. Stała `Notes.CONTENT_URI` jest zdefiniowana jako element `static final` w pliku *Notepad.java*:

```
public static final Uri CONTENT_URI =
      Uri.parse("content://" + AUTHORITY + "/notes");
```

Klasa `Notes` znajduje się wewnątrz klasy `Notepad`. Na razie wystarczy wiedzieć, że identyfikator URI przedstawiony powyżej każe dostawcy treści pobrać wszystkie notatki. Gdyby identyfikator ten wyglądał następująco:

```
public static final Uri CONTENT_URI =
      Uri.parse("content://" + AUTHORITY + "/notes/11");
```

to używany dostawca treści zwróciłby notatkę posiadającą identyfikator o wartości `11`. Temat dostawców treści oraz identyfikatorów URI zostanie poruszony w rozdziale 3.

Klasa `NotesList` jest dopełnieniem klasy `ListActivity`, definiującej sposób wyświetlania danych w postaci listy. Składniki listy są zarządzane poprzez wewnętrzną klasę `ListView` (element interfejsu UI), wyświetlającą notatki w oknie listy. Po wstawieniu identyfikatora URI do intencji danej aktywności aktywność ta zgłasza gotowość do zbudowania kontekstowego menu dla notatek. Jeżeli Czytelnik starał się poznać tę aplikację, zauważył zapewne, że w zależności od wybranego elementu wyświetlane jest menu kontekstowe. Jeśli na przykład zostanie zaznaczona notatka, zostaną wyświetlone opcje *Edit note* oraz *Edit title*. Jeżeli notatka nie zostanie zaznaczona, dostępna będzie opcja *Add note*.

Widzimy następnie, że aktywność wykonuje zarządzaną kwerendę, w wyniku czego pojawia się kursor. Poprzez zarządzaną kwerendę mamy na myśli, że Android będzie zarządzał przywołanym kursorem. Jako część tego procesu zarówno aplikacja, jak i aktywność nie muszą martwić się pozycjonowaniem kursora, jego wczytywaniem lub usunięciem z pamięci w przypadku wczytania lub usunięcia aplikacji z pamięci. Interesujące są parametry elementu `managedQuery()`, opisane w tabeli 2.3.

Tabela 2.3. Parametry elementu Activity.managedQuery()

Parametr	Typ danych	Opis
`URI`	`Uri`	Identyfikator URI dostawcy treści
`projection`	`String[]`	Zwracana kolumna (nazwy kolumn)
`selection`	`String`	Opcjonalna klauzula *where*
`selectionArgs`	`String[]`	Wybierane argumenty w przypadku, gdy kwerenda zawiera znaki zapytania
`sortOrder`	`String`	Kolejność sortowania zestawu wynikowego

Elementy `managedQuery()` oraz bliźniaczy `query()` omówimy w dalszej części tego podrozdziału oraz w rozdziale 3. Na razie istotna jest informacja, że kwerendy w Androidzie zwracają dane tabelaryczne. Parametr *projection* pozwala określić interesujące nas kolumny. Można także ograniczyć wynikowy zestaw oraz go posortować za pomocą klauzuli sortowania, używanej w języku SQL (na przykład *asc* lub *desc*). Należy zauważyć także, że kwerenda w Androidzie musi zwrócić kolumnę o nazwie `_ID`, żeby móc obsługiwać wyświetlanie pojedynczych rekordów. Ponadto należy znać typ danych zwracanych przez dostawcę treści — czy kolumna zawiera dane typu `string`, `int`, `binary` i tak dalej.

Po wykonaniu kwerendy zwrócony kursor jest przekazywany konstruktorowi elementu `SimpleCursorAdapter`, przekształcającemu rekordy zestawu danych w elementy interfejsu użytkownika (`ListView`). Przyjrzyjmy się bliżej parametrom przekazywanym do konstruktora elementu `SimpleCursorAdapter`:

```
      SimpleCursorAdapter adapter =
   new SimpleCursorAdapter(this, R.layout.noteslist_item,
cursor, new String[] { Notes.TITLE }, new int[] { android.R.id.text1 });
```

W szczególności zwróćmy uwagę na drugi parametr: identyfikator widoku reprezentującego elementy w metodzie `ListView`. Jak się będzie można przekonać w rozdziale 3., Android zawiera automatycznie generowaną klasę użytkową, zawierającą odniesienia do zasobów projektu. Jest to tak zwana klasa `R`, gdyż mieści się w pliku *R.java*. Podczas kompilowania projektu narzędzie AAPT tworzy klasę `R` z zasobów umieszczonych w folderze *res*. Na przykład można umieścić wszystkie zasoby składające się z ciągów znaków w folderze *values*, a narzędzie AAPT wygeneruje identyfikator `public static` dla każdego z tych zasobów. Generalnie Android obsługuje w ten sposób wszystkie zasoby. Na przykład w konstruktorze elementu `SimpleCursorAdapter` aktywność `NotesList` przekazuje identyfikator widoku umożliwiającego wyświetlanie elementu listy notatek. Dzięki tej klasie użytkowej nie ma potrzeby umieszczania zasobów wewnątrz głównego kodu oraz uzyskuje się możliwość sprawdzania odniesień w trakcie kompilacji. Inaczej mówiąc, jeżeli zasób zostanie usunięty, klasa R straci do niego odniesienie i żaden kod powiązany z tym zasobem nie zostanie skompilowany.

Przyjrzyjmy się kolejnej koncepcji Androida, o której wspomnieliśmy nieco wcześniej: metodzie `onListItemClick()` w klasie `NotesList` (listing 2.3).

Listing 2.3. Metoda onListItemClick

```
@Override
   protected void onListItemClick(ListView l, View v, int position, long id) {
      Uri uri = ContentUris.withAppendedId(getIntent().getData(), id);

      String action = getIntent().getAction();
      if (Intent.ACTION_PICK.equals(action) ||
Intent.ACTION_GET_CONTENT.equals(action)) {
         setResult(RESULT_OK, new Intent().setData(uri));
      } else {
         startActivity(new Intent(Intent.ACTION_EDIT, uri));
      }
   }
```

Metoda `onListItemClick()` jest wywoływana po zaznaczeniu notatki przez użytkownika. Jest ona przykładem aktywności uruchamiającej inną aktywność. Po zaznaczeniu notatki metoda ta tworzy identyfikator URI poprzez dodanie identyfikatora danej notatki do bazowego identyfikatora URI. Zostaje on następnie przekazany metodzie `startActivity()` wraz z nową intencją. Użycie metody `startActivity()` jest jednym ze sposobów uruchomienia aktywności: aktywność zostaje rozpoczęta, jednak po jej zakończeniu nie zostaje wyświetlony raport z wynikami. Inną możliwością uruchomienia aktywności jest użycie metody `startActivityForResult()`. Za jej pomocą można rozpocząć aktywność i zarejestrować wywoływanie zwrotne po jej zakończeniu. Można zastosować metodę `startActivity ↪ForResult()` na przykład w przypadku uruchomienia aktywności służącej do zaznaczania kontaktu, gdy ten kontakt ma być dostępny po zakończeniu aktywności.

W tym momencie można zacząć zastanawiać się, jak wygląda interakcja użytkownika względem aktywności. Na przykład: jeżeli uruchomiona aktywność uruchamia następną aktywność, a *ta* z kolei uruchamia inną aktywność (i tak dalej), to z którą aktywnością pracuje użytkownik? Czy może kontrolować jednocześnie wszystkie aktywności, czy może jest ograniczony do jednej? Okazuje się, że aktywności posiadają zdefiniowany cykl życia. Są one utrzymywane w stosie aktywności, na którego szczycie znajduje się uruchomiona aktywność. Jeżeli aktywność uruchomi inną aktywność, pierwsza uruchomiona aktywność przesunie się w dół stosu, a nowa zostanie umieszczona na jego szczycie. Aktywności znajdujące się na niższych poziomach stosu mogą znajdować się w stanie wstrzymania lub zatrzymania. Wstrzymana aktywność jest częściowo lub całkowicie widoczna dla użytkownika; aktywność zatrzymana jest dla niego niewidoczna. System może usunąć ze stosu wstrzymane lub zatrzymane aktywności, jeżeli trzeba będzie zwolnić miejsce na zasoby.

Przejdźmy teraz do trwałości danych. Notatki tworzone przez użytkownika zapisywane są w rzeczywistej bazie danych urządzenia. Ściślej mówiąc, magazynem notatek programu Notepad jest baza danych SQLite. Wcześniej wspomniana metoda `managedQuery()` służy do określania danych w bazie danych poprzez dostawcę treści. Prześledźmy, w jaki sposób identyfikator URI, dostarczony metodzie `managedQuery()`, powoduje wykonanie kwerendy wobec bazy SQLite. Przypomnijmy, że identyfikator URI przekazany metodzie `managedQuery()` wygląda następująco:

```
public static final Uri CONTENT_URI =
Uri.parse("content://" + AUTHORITY + "/notes");
```

Identyfikatory URI treści zawsze przybierają następującą formę: `content://`, następnie uprawnienie (`AUTHORITY`), a na końcu segment ogólny (zależny od kontekstu). Ponieważ identyfikator URI nie zawiera rzeczywistych informacji, w jakiś sposób musi wpływać na wykonanie kodu generującego dane. Jaki jest związek pomiędzy tym identyfikatorem a kodem? W jaki sposób odniesienie URI wpływa na kod produkujący informacje? Czy identyfikator URI jest usługą HTTP lub sieciową? Okazuje się, że identyfikator URI, a dokładniej jego część związana z uprawnieniami, jest skonfigurowany w pliku *AndroidManifest.xml* jako dostawca treści:

```
<provider android:name="NotePadProvider"
      android:authorities="com.google.provider.NotePad"/>
```

Kiedy Android trafi na identyfikator URI, który należy przeanalizować, skupia się na jego części związanej z uprawnieniami i sprawdza klasę `ContentProvider` skonfigurowaną dla tych uprawnień. Aplikacja Notepad posiada klasę `NotePadProvider`, umieszczoną w pliku *AndroidManifest.xml*, skonfigurowaną dla uprawnienia *com.google.provider.NotePad*. Na listingu 2.4 został przedstawiony niewielki wycinek tej klasy.

Listing 2.4. Klasa NotePadProvider

```
public class NotePadProvider extends ContentProvider
{

   @Override
   public Cursor query(Uri uri, String[] projection, String selection,
 String[] selectionArgs,String sortOrder) {}

   @Override
   public Uri insert(Uri uri, ContentValues initialValues) {}

   @Override
   public int update(Uri uri, ContentValues values, String where,
 String[] whereArgs) {}

   @Override
   public int delete(Uri uri, String where, String[] whereArgs) {}

   @Override
   public String getType(Uri uri) {}

   @Override
   public boolean onCreate() {}


   private static class DatabaseHelper extends SQLiteOpenHelper {}

   @Override
      public void onCreate(SQLiteDatabase db) {}

      @Override
      public void onUpgrade(SQLiteDatabase db,
```

```
int oldVersion, int newVersion) {
         //...
      }
   }
}
```

Klasa `NotePadProvider` rozszerza funkcjonalność klasy `ContentProvider`. Ta druga klasa definiuje sześć abstrakcyjnych metod, z których cztery są operacjami CRUB (ang. *Create, Read, Update, Delete* — tworzenie, odczyt, aktualizacja, usuwanie). Pozostałe dwie metody to `onCreate()` oraz `getType()`. Metoda `onCreate()` jest wywoływana podczas pierwszego utworzenia dostawcy treści. Dzięki metodzie `getType()` zostaje dostarczony typ MIME dla zestawu wyników (po przeczytaniu rozdziału 3. stanie się zrozumiałe działanie typów MIME).

Innym interesującym składnikiem klasy `NotePadProvider` jest wewnętrzna klasa `Database ↪Helper`, stanowiąca rozwinięcie klasy `SQLiteOpenHelper`. Rolą obydwu klas jest inicjacja, otwieranie oraz zamykanie bazy danych aplikacji Notepad, a także wykonywanie innych operacji bazodanowych. Co ciekawe, klasa `DatabaseHelper` składa się wyłącznie z kilku linijek kodu (listing 2.5), podczas gdy większość pracy wykonuje implementacja klasy `SQLiteOpen ↪Helper`.

Listing 2.5. Klasa DatabaseHelper

```
private static class DatabaseHelper extends SQLiteOpenHelper {

   DatabaseHelper(Context context) {
      super(context, DATABASE_NAME, null, DATABASE_VERSION);
   }

   @Override
   public void onCreate(SQLiteDatabase db) {
      db.execSQL("CREATE TABLE " + NOTES_TABLE_NAME + " ("
         + Notes._ID + " INTEGER PRIMARY KEY,"
         + Notes.TITLE + " TEXT,"
         + Notes.NOTE + " TEXT,"
         + Notes.CREATED_DATE + " INTEGER,"
         + Notes.MODIFIED_DATE + " INTEGER"
         + ");");
   }

   //...
}
```

Jak zostało przedstawione na listingu 2.5, metoda `onCreate()` generuje tabelę aplikacji Notepad. Należy zwrócić uwagę, że konstruktor klasy wywołuje konstruktora superklasy za pomocą nazwy tabeli. Superklasa wywoła metodę `onCreate()` jedynie w wypadku, gdy taka tablica nie istnieje w bazie danych. Warto również zauważyć, że jedną z kolumn w tabeli aplikacji Notepad jest `_ID`, omówiona kilka stron wyżej.

Przyjrzyjmy się teraz jednej z operacji CRUD: metodzie `insert()` (listing 2.6).

Listing 2.6. Metoda insert()

```
//…
SQLiteDatabase db = mOpenHelper.getWritableDatabase();
      long rowId = db.insert(NOTES_TABLE_NAME, Notes.NOTE, values);
      if (rowId > 0) {
         Uri noteUri = ContentUris.withAppendedId(
NotePad.Notes.CONTENT_URI, rowId);
         getContext().getContentResolver().notifyChange(noteUri, null);
         return noteUri;
      }
```

Metoda `insert()` wykorzystuje swoje wewnętrzne wystąpienie `DatabaseHelper`, żeby uzyskać dostęp do bazy danych, a następnie wstawia rekord notatek. Zwrócony identyfikator krotki zostaje następnie dołączony do identyfikatora URI, a taki nowy identyfikator zostaje zwrócony aplikacji wywołującej.

Czytelnik do tej pory powinien już być zaznajomiony ze strukturą aplikacji Androida. Poruszanie się w aplikacji Notepad, a także innych przykładowych programach, nie powinno sprawiać problemów. Dobrym pomysłem jest uruchomienie przykładowych aplikacji i zapoznanie się z ich działaniem. Zajmijmy się teraz ogólnym cyklem życia aplikacji dla systemu Android.

Badanie cyklu życia aplikacji

Cykl życia aplikacji stworzonej na Androida jest ściśle zarządzany przez system w oparciu o potrzeby użytkownika, dostępne zasoby i tak dalej. Użytkownik może chcieć otworzyć na przykład przeglądarkę internetową, ale ostatecznie to system decyduje, czy aplikacja zostanie uruchomiona. Chociaż system jest głównym zarządcą, postępuje zgodnie z pewnymi zdefiniowanymi oraz logicznymi wytycznymi, pozwalającymi określić, czy aplikacja ma zostać wczytana, wstrzymana lub zatrzymana. Jeżeli użytkownik korzysta aktualnie z aktywności, system wyznaczy tej aplikacji wysoki priorytet. Z drugiej strony, jeżeli aktywność nie jest widoczna, a system zdecyduje, że należy zamknąć aplikację w celu zwolnienia zasobów, to zostanie zamknięty program posiadający mniejszy priorytet.

Porównajmy to z cyklem życia aplikacji sieciowych, stworzonych w środowisku J2EE. Są one w sposób luźny zarządzane przez kontener, w którym są uruchomione. Na przykład aplikacja może zostać usunięta z pamięci w przypadku, gdy jest ona bezczynna przez określony czas. Generalnie jednak kontener nie będzie umieszczał aplikacji w pamięci oraz usuwał jej stamtąd w oparciu o obciążenie oraz (lub) dostępność zasobów. Zazwyczaj dostępna jest wystarczająca ilość zasobów, żeby było jednocześnie uruchomionych wiele zasobów. W przypadku Androida zasoby są bardziej ograniczone, więc system musi posiadać kontrolę oraz władzę nad aplikacjami.

Uwaga! W Androidzie każda aplikacja jest uruchomiona w oddzielnym procesie, posiadającym własną wirtualną maszynę. W ten sposób zostaje zapewnione środowisko chronionej pamięci. Ponadto poprzez przydzielenie aplikacji do indywidualnych procesów system może określać ich priorytet. Na przykład uruchomiony w tle proces wykonujący zadanie znacznie pochłaniające zasoby procesora nie może blokować przychodzącego połączenia telefonicznego.

Koncepcja cyklu życia aplikacji jest logiczna, jednak podstawowa struktura aplikacji systemu Android komplikuje sprawę. Gwoli ścisłości, architektura aplikacji jest zorientowana na składniki oraz integrację. Dzięki temu uzyskiwane jest bogate doświadczenie użytkownika, możliwość bezproblemowego wielokrotnego korzystania z aplikacji oraz łatwość jej integracji, jednak przed menedżerem cyklu życia stoi bardzo skomplikowane zadanie.

Rozważmy typowy scenariusz. Użytkownik rozmawia z kimś przez telefon i musi otworzyć wiadomość e-mail, żeby odpowiedzieć na zadane przez rozmówcę pytanie. Otwiera ekran główny, aplikację pocztową, klika adres łącza do witryny, zawierającej poszukiwaną wiadomość i przytacza jej fragment ze strony internetowej. W takim przypadku wymagane są cztery aplikacje: ekranu głównego, telefonu, pocztowa oraz przeglądarka. Użytkownik w sposób ciągły może przenosić się pomiędzy aplikacjami, jednak w tle system zapisuje oraz przywraca ich stan. Przykładowo po kliknięciu adresu łącza w wiadomości e-mail system zapisuje metadane uruchomionej aktywności tej wiadomości, zanim przekaże aktywności przeglądarki dane potrzebne do przekierowania na adres URL. Tak naprawdę system zapisuje metadane każdej aktywności przed uruchomieniem następnej, dzięki czemu może do niej wrócić (na przykład gdy użytkownik wraca do poprzedniej strony). Jeżeli wystąpi problem z ilością pamięci, zostanie zamknięty proces wykonujący aktywność, a w razie konieczności zostanie wznowiony.

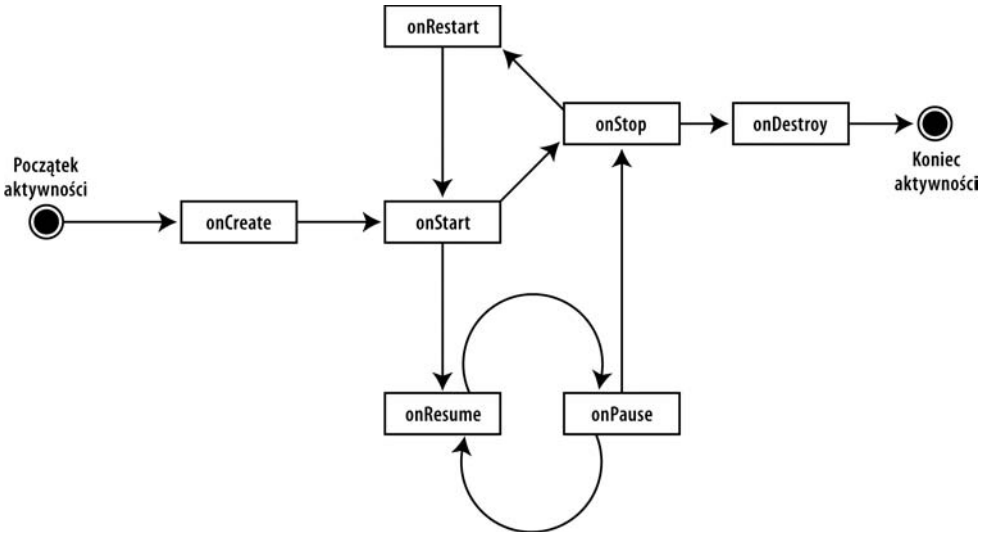
System Android jest wrażliwy na cykl życia aplikacji oraz jej elementów składowych. Zatem, żeby stworzyć stabilną aplikację, należy zrozumieć zdarzenia cyklu życia oraz nauczyć się nimi posługiwać. Procesy korzystające z danej aplikacji oraz jej składników natrafiają na różnorodne zdarzenia cyklu życia i istnieje możliwość zaimplementowania wywołań zwrotnych, zajmujących się zmianami ich stanu. Na początek warto zapoznać się z wywołaniami cyklu życia aktywności (listing 2.7).

Listing 2.7. Metody cyklu życia aktywności

```
protected void onCreate(Bundle savedInstanceState);
protected void onStart();

protected void onRestart();
protected void onResume();
protected void onPause();
protected void onStop();
protected void onDestroy();
```

W listingu 2.7 zostały wypisane metody, które są wywoływane podczas cyklu życia aktywności. Dla stworzenia stabilnej struktury aplikacji istotne jest zrozumienie, kiedy dana metoda jest wywoływana przez system. Nie wszystkie metody muszą być implementowane. Jeżeli zostaną użyte wszystkie wywołania, należy również stworzyć analogiczne wersje dla superklas. Na rysunku 2.15 zostały pokazane przejścia pomiędzy stanami aktywności.

System może uruchamiać oraz zatrzymywać aktywności w zależności od tego, co się w nim dzieje. Metoda `onCreate()` jest wywoływana podczas pierwszego utworzenia aktywności. Po tej metodzie zawsze pojawia się metoda `onStart()`, jednak nie jest to regułą w przeciwnym kierunku, gdyż metoda ta może zostać wywołana w przypadku zatrzymania aplikacji (za pomocą metody `onStop()`). Po wywołaniu metody `onStart()` aktywność nie jest jeszcze widziana przez użytkownika. Po metodzie `onStart()` wywoływana jest metoda `onResume()` w momencie, gdy aktywność znajduje się na pierwszym planie i jest dostępna dla użytkownika. To właśnie teraz użytkownik bezpośrednio korzysta z aplikacji.



Rysunek 2.15. Zmiany stanów aktywności

Kiedy użytkownik zdecyduje się przenieść do innej aktywności, system przywoła metodę `onPause()` dla opuszczanej aktywności. Z tego miejsca może zostać wywołana metoda `onResume()` lub `onStop()`. Ta pierwsza metoda jest wywoływana na przykład wtedy, gdy użytkownik przywróci aktywność na pierwszy plan. Jeżeli stanie się ona niewidoczna dla użytkownika, wywołana zostanie metoda `onStop()`. Jeżeli po tym wywołaniu aktywność zostanie przywrócona na pierwszy plan, nastąpi przywołanie metody `onRestart()`. Jeżeli aktywność znajduje się w stosie używanych aktywności, lecz jest niewidoczna dla użytkownika, a system postanowi ją zakończyć, wywołana zostanie metoda `onDestroy()`.

Omówiony model stanów aktywności może wydawać się skomplikowany, jednak umieszczenie wszystkich metod nie jest konieczne. Tak naprawdę najczęściej będą wykorzystywane metody `onCreate()` oraz `onPause()`. Pierwsza metoda służyć będzie do tworzenia interfejsu UI danej aktywności. W tej metodzie dane będą wiązane z widgetami, a procedury obsługi zdarzeń — z elementami interfejsu użytkownika. Metoda `onPause()` wykorzystywana jest w przypadku konieczności przechowywania istotnych danych w magazynie aplikacji. Jest to ostatnia bezpieczna metoda, wywoływana przed zamknięciem aplikacji. Metody `onStop()` oraz `onDestroy()` nie zawsze są wywoływane, więc nie należy na nie liczyć w przypadku tworzenia szczególnie ważnych programów.

Jakie wnioski powinny się nasuwać z powyższych wywodów? System zarządza aplikacją i może w każdej chwili uruchomić, zatrzymać lub przywrócić każdy z jej składników. Chociaż składniki te są kontrolowane przez system, nie są one całkowicie oddzielone od aplikacji. Innymi słowy, jeżeli system uruchomi aktywność w aplikacji, można liczyć na kontekst aplikacji w tej aktywności. Na przykład nie jest rzadkością posiadanie zmiennych globalnych, współdzielonych przez aktywności w aplikacji. Taką zmienną globalną tworzy się poprzez napisanie rozszerzenia klasy `android.app.Application`, a następnie inicjowanie jej w metodzie `onCreate()` (listing 2.8). Aktywności oraz inne składniki aplikacji będą uzyskiwały dostęp do tych odniesień bez obaw, że nie zostaną uruchomione.

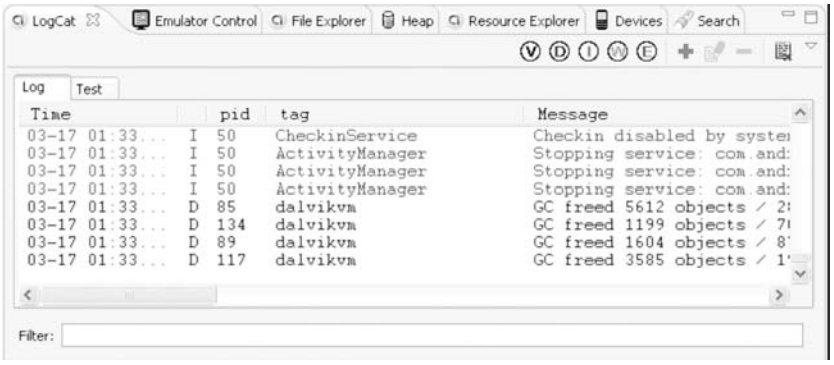
Listing 2.8. Rozszerzenie klasy Application

```
public class MyApplication extends Application
{
   // zmienna globalna
   private static final String myGlobalVariable;

   @Override
   public void onCreate()
   {
      super.onCreate();
      //... tutaj następuje inicjacja zmiennych globalnych
      myGlobalVariable = loadCacheData();
   }

   public static String getMyGlobalVariable() {
      return myGlobalVariable;
   }
}
```

W następnym podrozdziale zaopatrzymy Czytelnika w pancerz usprawniający tworzenie aplikacji na Androida: omówimy usuwanie błędów z programu.

Usuwanie błędów w aplikacji

Po napisaniu kilku linijek pierwszej aplikacji wiele osób z pewnością zacznie się zastanawiać, czy będzie możliwe przeprowadzenie sesji usuwania błędów podczas korzystania z aplikacji w emulatorze. Zestaw Android SDK został zaopatrzony w wiele aplikacji pozwalających na sprawdzanie aplikacji pod kątem błędów. Są one zintegrowane ze środowiskiem Eclipse (rysunek 2.16).



Rysunek 2.16. Narzędzia do usuwania błędów, które można wykorzystać podczas tworzenia aplikacji

Jednym z takich narzędzi jest LogCat. Aplikacja ta wyświetla wiadomości dziennika, tworzone podczas korzystania z klas `android.util.Log`, `System.out.println`, wyjątków i tak dalej. Podczas gdy klasa `System.out.println` działa i informacje są wyświetlane w oknie LogCat, do wyświetlenia komunikatów z aplikacji należy użyć klasy `android.util.Log`. Są w niej zdefiniowane znajome metody informacyjne, ostrzeżeń oraz błędów, które można filtrować w oknie LogCat. Przykładem polecenia *Log* jest:

```
Log.v("string TAG", "Ta wiadomość zostanie zapisana w dzienniku");
```

Pakiet SDK zawiera również eksplorator plików, umożliwiający oglądanie oraz przenoszenie plików do urządzenia, nawet jeśli jest ono emulatorem.

Narzędzia stają się dostępne po wybraniu perspektywy *Debug* w środowisku Eclipse. Można także pojedynczo uruchamiać narzędzia w perspektywie Java, klikając opcje *Window/Show View/Other/Android*.

Istnieje także możliwość dokładnego śledzenia aplikacji za pomocą klasy `android.os.Debug`, zawierającej metodę rozpoczęcia śledzenia (`Debug.startMethodTracing()`) oraz zakończenia śledzenia (`Debug.stopMethodTracing()`). W urządzeniu (lub w emulatorze) zostanie utworzony plik śledzenia. Można go następnie skopiować do stacji roboczej i obserwować dane wyjściowe znacznika za pomocą narzędzia *traceview*, znajdującego się w katalogu *tools* zestawu SDK. Pozostałe narzędzia zostaną omówione w następnych rozdziałach.

Podsumowanie

W tym rozdziale zademonstrowaliśmy, w jaki sposób należy skonfigurować środowisko projektowe do tworzenia aplikacji na platformę Android. Opisaliśmy podstawowe elementy budulcowe interfejsu API Androida, a także wprowadziliśmy pojęcia widoków, aktywności, intencji, dostawców treści oraz usług. W dalszej części przeanalizowaliśmy strukturę aplikacji Notepad pod kątem wspomnianych już bloków budulcowych oraz składników aplikacji. Następnie omówiliśmy istotę cyklu życia aplikacji pisanych na Androida. Na końcu wspomnieliśmy o narzędziach do usuwania błędów zestawu Android SDK, zintegrowanych ze środowiskiem Eclipse.

A teraz wprowadzimy podstawy projektowania na platformę Android. Następny rozdział został poświęcony dostawcom treści, zasobom oraz intencjom.