Shane Conder, Lauren Darcey

ADDISON-WESLEY

WYCZERPUJĄCY PRZEWODNIK,
GRUNTOWNIE ZAKTUALIZOWANY
W OPARCIU O NAJNOWSZĄ WERSJĘ ANDROID SDK
I NAJLEPSZE TECHNIKI PROGRAMOWANIA!

Android

WYDANIE II

PROGRAMOWANIE APLIKACJI NA URZĄDZENIA PRZENOŚNE

Helion

Android. Programowanie aplikacji na urządzenia przenośne. Wydanie II

Autorzy: CShane Conder, Lauren Darcey
Tłumaczenie: Piotr Rajca
ISBN: 978-83-246-3349-4
Tytuł oryginału: Android Wireless Application Development (2nd Edition)
Format: 170×230, stron: 820

Wyczerpujący przewodnik, gruntownie zaktualizowany w oparciu o najnowszą wersję Android SDK i najlepsze techniki programowania!

Gdy w 2008 roku Google opublikował Androida, rynkiem telefonów komórkowych wprawdzie nieco zatrzęsło, ale nikt nie spodziewał się, że ta platforma aż tak poważnie zagrozi takim gigantom mobilnych systemów operacyjnych, jak iOS Apple, Windows Mobile, Symbian czy RIM BlackBerry. Od tego czasu każde nowe urządzenie z Androidem miało coraz większe możliwości i było jeszcze bardziej ekscytujące od swoich poprzedników. Minęły zaledwie trzy lata od dnia, kiedy na rynku pojawił się pierwszy telefon z systemem Android - T-Mobile G1, stworzony przez firmę HTC - a system ten już okrzyknięty został najszybciej sprzedającą się platformą dla telefonów przenośnych. To oczywiście nie pozostaje bez echa: potrzeba coraz więcej ludzi specjalizujących się w programowaniu aplikacji na tę fascynującą, darmową i otwartą platformę mobilną.

Oto kompletny podręcznik, zawierający wszystko, co potrzebne do tworzenia, wdrażania i sprzedawania aplikacji na urządzenia przenośne działające pod kontrolą najnowszych wersji Androida. Autorzy – w oparciu o swoje wieloletnie doświadczenie w tworzeniu mobilnych aplikacji – wprowadzą Cię we wszystkie etapy tego procesu: pomysł, projektowanie, pisanie kodu, testowanie, pakowanie i rozpowszechnianie aplikacji. Poznasz doskonale specyfikację platformy Android, podstawowe zasady efektywnego projektowania aplikacji na nią przeznaczonych oraz najlepsze praktyki związane z tworzeniem wygodnych interfejsów użytkownika. Znajdziesz tu także wyczerpujące opisy wszystkich kluczowych interfejsów programistycznych: do obsługi składowania danych, komunikacji sieciowej, obsługi rozmów telefonicznych, usług lokalizacyjnych, multimediów, grafiki 3D oraz opcjonalnych komponentów sprzętowych. Oprócz tego książka została uzupełniona praktycznymi sztuczkami, które pozwolą Ci zaoszczędzić sporo cennego czasu i uniknąć wielu niepotrzebnych pułapek!

Ponadto znajdziesz tu:

- kilka rozdziałów opisujących interfejs API do obsługi technologii związanych z WWW, a także Android NDK, poszerzanie zasięgu aplikacji, zarządzanie użytkownikami, synchronizację danych, tworzenie kopii bezpieczeństwa, zaawansowane metody wprowadzania danych
- wyczerpujące informacje na temat plików manifestu, dostawców treści, projektowania i testowania aplikacji
- prezentację najbardziej aktualnych i interesujących zagadnień, takich jak obsługa komunikacji Bluetooth i gestów, rozpoznawanie mowy, widgety, technologie Live Folders, Live Wallpapers oraz globalne wyszukiwanie
- aktualne informacje na temat generowania grafiki 3D przy użyciu OpenGL ES 2.0
- zagadnienia związane z zapewnianiem zgodności pomiędzy różnymi urządzeniami

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

Spis treści

3

Pierwsza aplikacja na Androida

Aktualnie Czytelnik powinien już posiadać na własnym komputerze zainstalowane i działające środowisko do pisania aplikacji na platformę Androida. Można mieć także nadzieję, że oprócz tego Czytelnik posiada także jakieś urządzenie z Androidem. Nadszedł zatem czas, by napisać swój pierwszy kod na tę platformę.

W tym rozdziale Czytelnik dowie się, jak należy dodawać i tworzyć nowe projekty aplikacji na Androida w środowisku Eclipse oraz jak sprawdzić, czy używane środowisko programistyczne zostało prawidłowo skonfigurowane. Oprócz tego w tym rozdziale Czytelnik napisze i przetestuje swoją pierwszą aplikację na Androida, uruchamiając ją zarówno w emulatorze, jak i na faktycznym urządzeniu.

Testowanie środowiska programistycznego

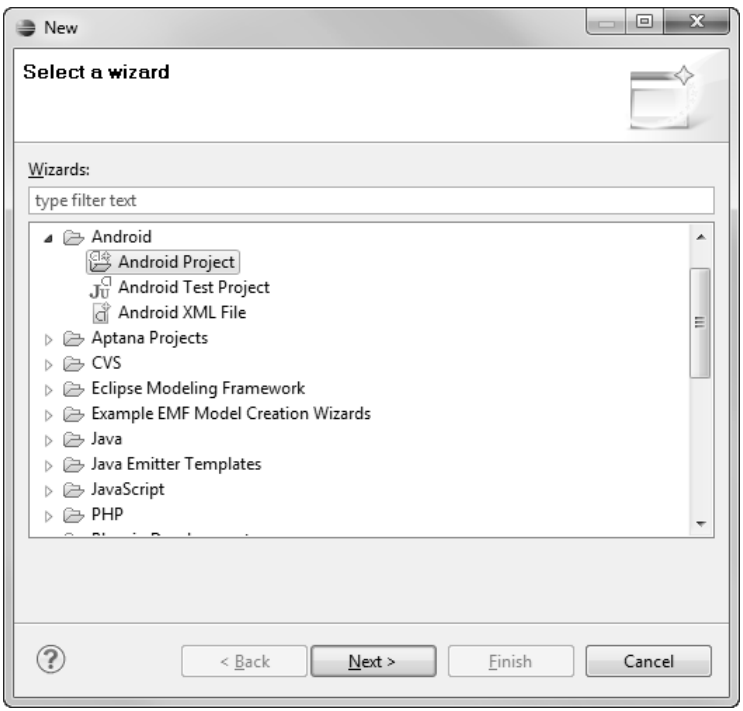
Najlepszym sposobem, by upewnić się, że używane środowisko programistyczne zostało skonfigurowane prawidłowo, jest skorzystanie z jakiejś gotowej aplikacji i przeprowadzenie próby jej uruchomienia. Z łatwością można do tego celu wykorzystać jedną z wielu przykładowych aplikacji dostarczanych wraz z Android SDK i umieszczonych w katalogu */samples*.

Wśród wielu przykładów aplikacji dostarczanych wraz z Android SDK można znaleźć klasyczną grę o nazwie Snake (wąż). Aby ją zbudować i uruchomić, w pierwszej kolejności należy utworzyć w przestrzeni roboczej środowiska Eclipse nowy projekt aplikacji na Androida, stworzyć odpowiedni profil wirtualnego urządzenia (AVD) i przygotować konfigurację uruchomieniową dla projektu. Po prawidłowym skonfigurowaniu wszystkich elementów środowiska będzie można zbudować aplikację i uruchomić ją na emulatorze bądź rzeczywistym urządzeniu.

Dodawanie aplikacji Snake do projektu w przestrzeni roboczej Eclipse

Pierwszą rzeczą, jaką należy zrobić, jest dodanie projektu *Snake* do przestrzeni roboczej Eclipse. W tym celu należy wykonać następujące czynności:

1. Wybrać opcję *File/New/Project*.
2. Rozwinąć grupę *Android*, a następnie wybrać kreatora *Android Project* (patrz rysunek 3.1).



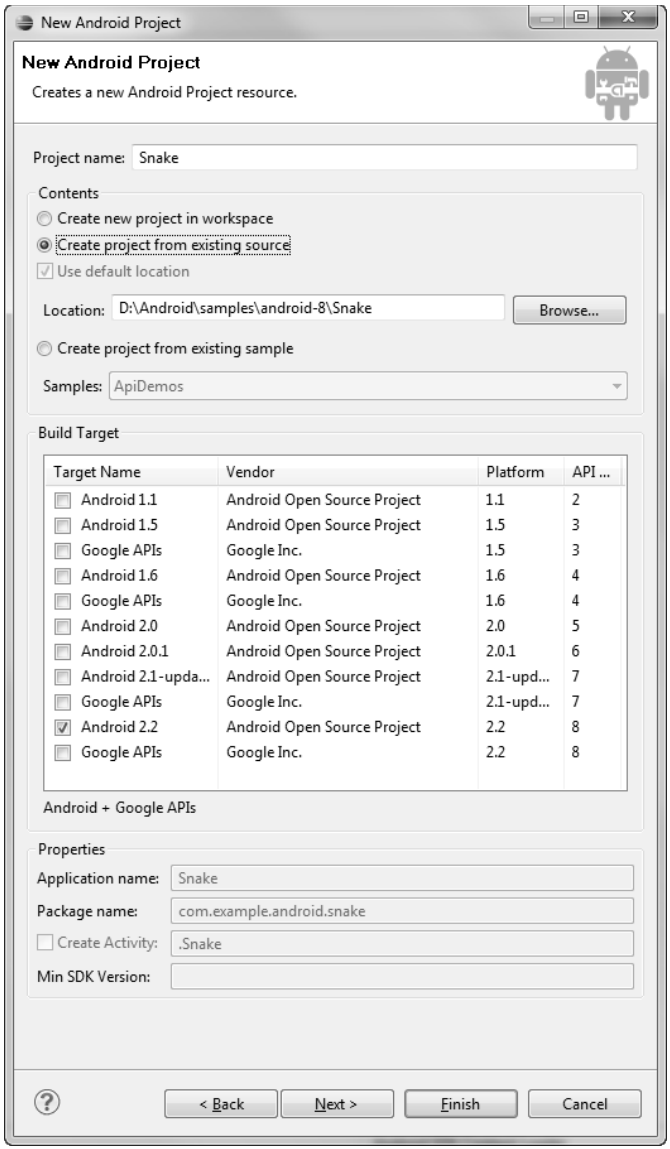
Rysunek 3.1. Tworzenie nowego projektu aplikacji na Androida

Wskazówka

Kiedy już raz wybierzemy kreatora *Android Project*, kolejne projekty będzie można tworzyć, wybierając z menu opcję *File/New/Android Project*.

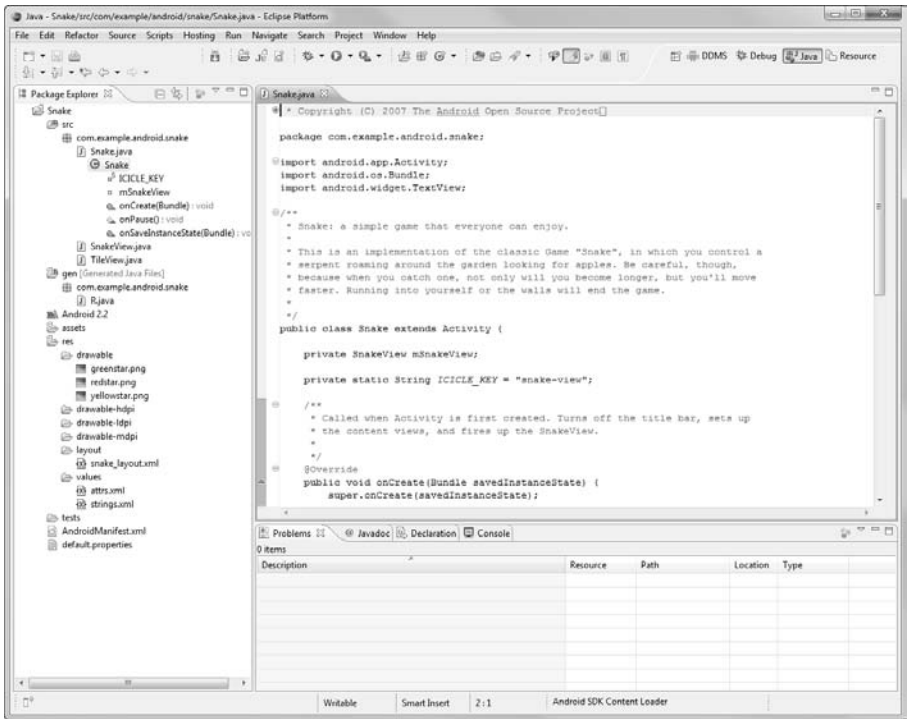
3. W sekcji *Content* zmienić wybraną opcję *Create new project in workspace* na *Create project from existing sources*.
4. Kliknąć przycisk *Browse* i przeglądnąć zawartość katalogu z programami przykładowymi dostarczanymi wraz z Android SDK.

5. Wybrać katalog *Snake*. Wszystkie pola kreatora powinny zostać wypełnione informacjami odczytanymi z pliku manifestu (patrz rysunek 3.2). Dodatkowo warto sprawdzić ustawienie *Build Target* i wybrać w nim tę wersję platformy Android, w jakiej mamy zamiar uruchamiać aplikację.



Rysunek 3.2. Szczegółowe informacje na temat projektu Snake

6. Kliknąć przycisk *Finish*. Kiedy to zrobimy, projekt *Snake* powinien zostać otwarty w środowisku Eclipse (patrz rysunek 3.3).



Rysunek 3.3. Pliki projektu Snake

> **Ostrzeżenie**
>
> Od czasu do czasu może się zdarzyć, że Eclipse wyświetli następujący komunikat o błędzie: *Project „Snake" is missing required source folder: „gen"*[1]. Jeśli tak się zdarzy, wystarczy odnaleźć w projekcie i usunąć plik *R.java* umieszczony w katalogu */gen*. Plik *R.java* jest generowany automatycznie, a po jego usunięciu problem powinien zniknąć.

Tworzenie wirtualnego urządzenia (AVD) na potrzeby projektu Snake

Kolejną czynnością, jaką należy wykonać, jest utworzenie wirtualnego urządzenia, które będzie opisywać typ urządzenia, jakie chcemy symulować w emulatorze podczas uruchamiania aplikacji; można przy tym określić, w jaką wersję platformy Android ma

[1] W projekcie „Snake" brakuje wymaganego katalogu źródłowego: „gen" — *przyp. tłum.*

zostać wyposażone urządzenie. Można określać także różne wielkości ekranów oraz ich orientacje, zażądać, by symulowane urządzenie było wyposażone w kartę SD, oraz podać jej pojemność.

Na potrzeby tego przykładu wystarczy AVD z domyślną wersją platformy Android 2.2. Poniżej opisane zostały czynności, jakie należy wykonać w celu utworzenia prostego urządzenia wirtualnego (AVD):

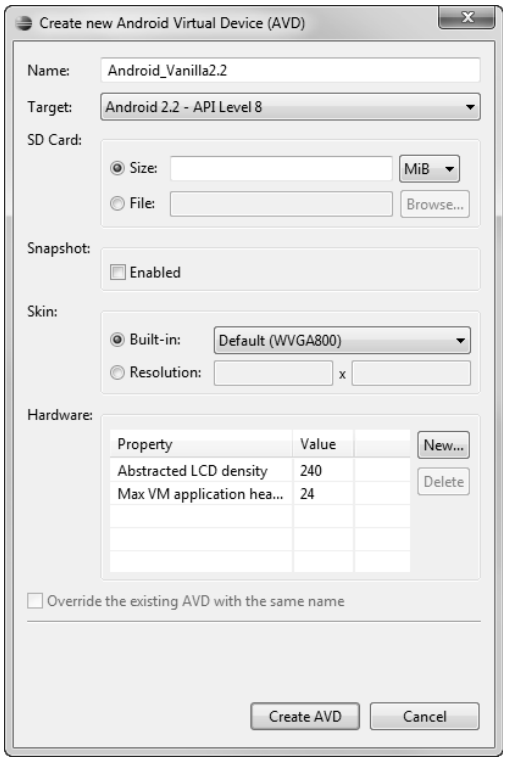
1. Z poziomu Eclipse uruchomić program Android Virtual Device, klikając umieszczony na pasku narzędzi przycisk z niewielką ikoną Androida i strzałką skierowana w dół —  . Jeśli nie można znaleźć tej ikony, to program menedżera można także uruchomić za pośrednictwem opcji dostępnych w menu *Window* środowiska Eclipse.
2. W menu *Virtual Devices* kliknąć przycisk *New*.
3. Podać nazwę tworzonego AVD. Ponieważ mamy zamiar wykorzystać wszystkie domyślne ustawienia, nadamy mu nazwę *Android_Vanilla2.2*.
4. Wybrać urządzenie docelowe. Ponieważ chcemy stworzyć zwyczajne urządzenie z Androidem 2.2, z rozwijalnej listy należy wybrać opcję *Android 2.2*.
5. Określić pojemność karty SD. Można ją podać w kilobajtach lub megabajtach. Obszar ten jest przydzielany z dostępnej przestrzeni na dysku twardym komputera. Warto wybrać jakąś rozsądną wartość, na przykład 1 GB (1024 MB). Jeśli Czytelnikowi brakuje wolnego miejsca na dysku bądź jeśli wiadomo, że nie trzeba będzie testować przechowywania danych aplikacji na karcie SD, to można utworzyć kartę o znacznie mniejszej pojemności, na przykład 64 MB.
6. Wybrać wygląd symulowanego urządzenia, tak zwaną „skórkę” (ang. *skin*). Ta opcja pozwala określać różne rozdzielczości ekranu symulowanego urządzenia. W naszym przypadku zastosujemy skórkę *WVGA800*. Chyba najdokładniej odpowiada ona najbardziej popularnym telefonom z Androidem, takim jak HTC Negus One lub Evo 4G, które właśnie leżą przede mną na biurku.

Ustawienia projektu zostały przedstawione na rysunku 3.4.

Wskazówka

Choć w tej książce skonfigurowaliśmy AVD tak, by emulowane urządzenie miało wysoką rozdzielczość, to jednak Czytelnik może wybrać inne ustawienia, odpowiadające urządzeniu, na jakim zamierza on uruchamiać aplikację.

7. Kliknąć przycisk *Create AVD* i poczekać na wykonanie operacji.
8. Kliknąć przycisk *Finish*. Ponieważ menedżer AVD formatuje pamięć przydzieloną na potrzeby karty SD symulowanego urządzenia, tworzenie wirtualnych urządzeń z kartami pamięci może chwilkę potrwać.



Rysunek 3.4. Tworzenie nowego AVD w środowisku Eclipse

Wskazówka

Wirtualne urządzenia AVD można także tworzyć przy użyciu narzędzi wchodzących w skład Android SDK wybieranych z poziomu wiersza poleceń.

Więcej informacji na temat tworzenia różnych typów urządzeń AVD można znaleźć w dodatku A, „Krótki przewodnik po emulatorze Androida".

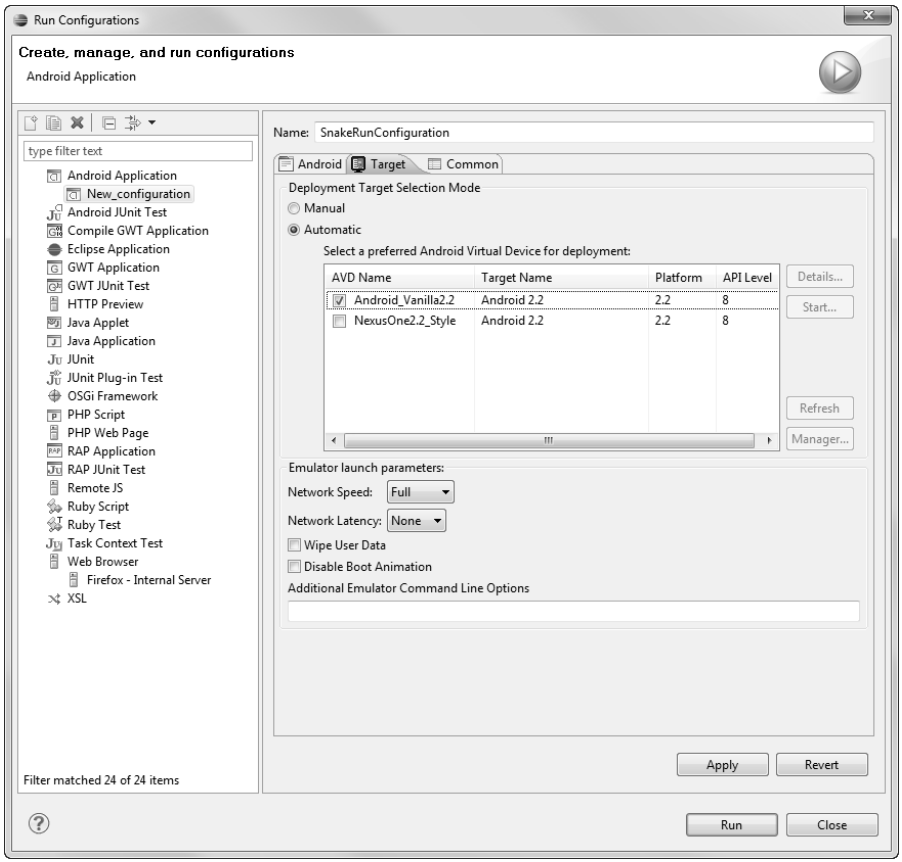
Tworzenie konfiguracji uruchomieniowej projektu Snake

Kolejną czynnością jest utworzenie konfiguracji uruchomieniowej w środowisku Eclipse. Określa ona, w jaki sposób aplikacja Snake będzie budowana i uruchamiana. To właśnie w ramach tej konfiguracji są określane opcje emulatora oraz punkt wejścia aplikacji.

Należy osobno utworzyć konfigurację uruchomieniową oraz konfigurację do debugowania aplikacji. Każda z nich ma niezależne opcje. Oba rodzaje konfiguracji tworzy się przy użyciu opcji dostępnych w menu *Run* (*Run/Run Configuration* oraz *Run/Debug Configuration*).

Oto czynności, jakie należy wykonać, by stworzyć podstawową konfigurację uruchomieniową dla aplikacji Snake:

1. Wybrać opcję *Run/Run Configuration* (lub kliknąć projekt prawym przyciskiem myszy i z wyświetlonego menu kontekstowego wybrać opcję *Run As*).
2. Dwukrotnie kliknąć opcję *Android Application*.
3. Podać nazwę nowej konfiguracji, na przykład `SnakeRunConfiguration` (patrz rysunek 3.5).



Rysunek 3.5. Konfiguracja uruchomieniowa dla projektu Snake, zakładka Target z wybranym urządzeniem AVD

4. Wybrać projekt — kliknąć przycisk *Browse* i wybrać projekt *Snake*.
5. Przejść na zakładkę *Target* i z listy dostępnych AVD wybrać utworzone wcześniej urządzenie *Android_Vanilla2.2* (patrz rysunek 3.5).

Na zakładkach *Target* oraz *Common* można określać także inne opcje, jednak na razie pozostawimy w nich wartości domyślne.

Uruchamianie aplikacji Snake w emulatorze

Teraz możemy już uruchomić aplikację Snake; w tym celu wystarczy wykonać następujące czynności:

1. Rozwinąć menu widoczne przy ikonie *Run As* umieszczonej na pasku narzędzi (przedstawiającej zielone kółko z trójkącikiem).
2. Z menu wybrać utworzoną wcześniej konfigurację uruchomieniową `SnakeRunConfiguration`.
3. Poczekać na uruchomienie emulatora Androida (co może chwilę potrwać).

> **Wskazówka**
>
> Należy się upewnić, że w momencie uruchamiania emulatora do komputera nie jest podłączone żadne urządzenie z Androidem (przy użyciu kabla USB). Domyślnym ustawieniem, zaznaczonym w sekcji *Deployment Target Selection Mode* (tryb wyboru urządzenia docelowego) na zakładce *Target* konfiguracji uruchomieniowej, jest wartość *Automatic*. Oznacza to, że jeśli do komputera będzie podłączone jakieś rzeczywiste urządzenie, to aplikacja Snake zostanie uruchomiona na nim, a nie w emulatorze.

4. Jeśli będzie to konieczne, przesunąć pasek na ekranie z lewa na prawo, by odblokować emulator (patrz rysunek 3.6).
5. Poczekać na uruchomienie aplikacji Snake, pokazanej na rysunku 3.7.

Emulator pozwala na interakcję z aplikacją Snake i faktyczne wypróbowanie tego, jak można się tą grą bawić. W przyszłości będzie ją można ponownie uruchomić w dowolnym momencie, korzystając z listy wszystkich aplikacji (w terminologii Androida nazywanej App Drawer).

Rysunek 3.6. Emulator w trakcie uruchamiania (zablokowany)

Budowanie pierwszej aplikacji na Androida

W końcu nadszedł czas, by napisać pierwszą aplikację na Androida. Zaczniemy od prostej aplikacji typu „Witaj, świecie", którą następnie będziemy rozbudowywać, poznając różne możliwości platformy Android.

> **Wskazówka**
>
> Wiele przykładów przedstawionych w tym rozdziale to fragmenty aplikacji MyFirstAndroidApp. Jej kod źródłowy można znaleźć w przykładach dołączonych do książki, które można pobrać z serwera FTP wydawnictwa Helion, *ftp://ftp.helion.pl/przyklady/andrp2.zip*.

Rysunek 3.7. Gra Snake

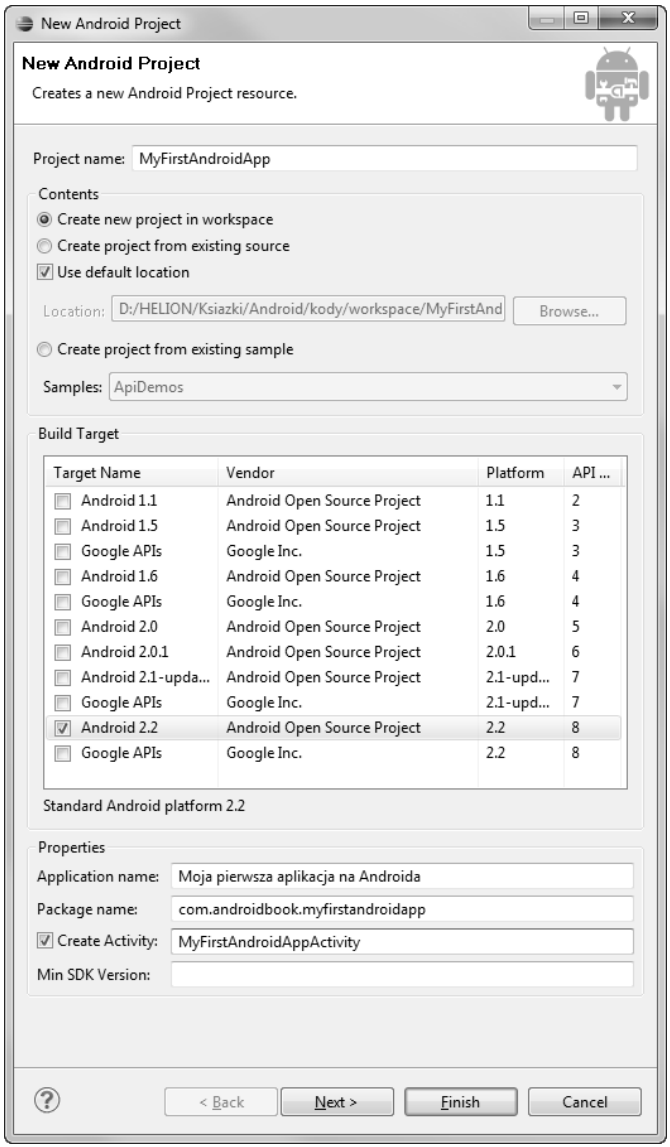
Tworzenie i konfiguracja nowego projektu aplikacji

Sposób tworzenia nowego projektu aplikacji na Androida jest bardo podobny do tego, który został opisany wcześniej przy okazji dodawania do obszaru roboczego Eclipse przykładowej aplikacji Snake.

Pierwszą czynnością, jaką należy wykonać, jest stworzenie nowego projektu i dodanie go do obszaru roboczego Eclipse. Kreator Android Project Wizard tworzy wszystkie pliki wymagane do utworzenia aplikacji na Androida. Poniżej opisane zostały wszystkie czynności, jakie należy wykonać, by utworzyć nowy projekt aplikacji na Androida w środowisku Eclipse:

1. Wybrać z menu opcję *File/New/Android Project* bądź kliknąć przycisk *Android Project* zawierający ikonę przedstawiającą folder z literą „a" oraz znakiem „+" (), umieszczony na pasku narzędzi Eclipse.
2. W polu *Project Name* podać nazwę tworzonego projektu. W naszym przypadku nadamy mu nazwę `MyFirstAndroidApp`.
3. Określić lokalizację plików projektu. Ponieważ mamy zamiar utworzyć nowy projekt, należy zaznaczyć przycisk opcji *Create new project in workspace*. Dodatkowo należy zaznaczyć pole wyboru *Use default location* lub wskazać katalog, w którym chcemy umieścić pliki źródłowe tworzonej aplikacji.
4. Z listy *Build Target* wybrać docelową wersję platformy Android, na jakiej będzie działać tworzona aplikacja. Warto wybrać wersję, która jest zgodna z urządzeniem, jakie posiadamy. W tym przypadku wybierzemy platformę Android 2.2.
5. Określić nazwę aplikacji. Jest to nazwa „przyjazna dla użytkownika", która będzie prezentowana na liście aplikacji w urządzeniu. W naszym przypadku tworzonej aplikacji nadamy nazwę `Moja pierwsza aplikacja na Androida`.
6. Określić nazwę pakietu. W tym przypadku należy podać nazwę określoną zgodnie ze standardowymi konwencjami nazewnictwa przestrzeni nazw pakietów stosowanymi w języku Java. Ponieważ wszystkie przykłady zamieszczone w tej książce są umieszczone w przestrzeni nazw `com.androidbook.*`, zastosujemy nazwę pakietu o następującej postaci: `com.androidbook.myfirstandroidapp`; choć oczywiście Czytelnik może podać inną, własną nazwę pakietu.
7. Zaznaczyć pole wyboru *Create Activity*. Zaznaczenie tej opcji poinformuje kreatora to tym, że należy wygenerować domyślną aktywność (ang. *activity*) pozwalającą na uruchomienie aplikacji. Nasza wersja tej klasy, dziedzicząca po `Activity`, będzie nosić nazwę `MyFirstAndroidAppActivity`.

   Ustawienia nowego projektu zostały przedstawione na rysunku 3.8.
8. W końcu można kliknąć przycisk *Finish*.
9. Aby nasza przykładowa aplikacja była prawidłowo spolonizowana, należy jeszcze zmodyfikować wyświetlane przez nią komunikaty. W tym celu w panelu *Project Explorer* Eclipse należy rozwinąć węzeł projektu, a następnie kliknąć kolejne opcje: *res* oraz *values* i dwukrotnie kliknąć plik *strings.xml*. W wyświetlonym edytorze zasobów należy zaznaczyć opcję *hello* — spowoduje to wyświetlenie z prawej strony dwóch pól tekstowych. W dolnym z nich, *Value*, należy zmienić słowa: „Hello World" na: „Witaj, świecie". W końcu należy zapisać plik zasobów, naciskając kombinację klawiszy *Ctrl+S*. Te ostatnie czynności mogą się wydać Czytelnikowi dosyć tajemnicze, jednak wszystko wyjaśni się w dalszej części książki, a w szczególności w rozdziałach 6. i 7., poświęconych odpowiednio zarządzaniu zasobami oraz wizualnym elementom interfejsu użytkownika aplikacji na Androida.



Rysunek 3.8. Konfiguracja tworzonej aplikacji przy użyciu narzędzia Android Project Wizard

Podstawowe pliki i katalogi aplikacji na Androida

Każda aplikacja przeznaczona na platformę Android posiada pewną grupę podstawowych plików tworzonych i używanych w celu określenia jej funkcjonalności (patrz tabela 3.1). Pliki te są tworzone domyślnie podczas generowania nowej aplikacji.

Tabela 3.1. *Ważne pliki i katalogi projektów aplikacji na Androida*

Plik	Ogólny opis
AndroidManifest.xml	Globalny plik opisujący aplikację. Definiuje on możliwości oraz uprawnienia aplikacji, jak również sposób jej działania.
default.properties	Tworzony automatycznie plik projektu. Określa on docelową wersję platformy Androida, na którą jest tworzona aplikacja, oraz inne opcje związane z jej budowaniem.
katalog *src*	Wymagany folder, w którym są przechowywane wszystkie kody źródłowe tworzonej aplikacji.
src/com.androidbook.myfirstandoridapp/MyFirstAndroidAppActivity.java	Główny plik źródłowy, definiujący punkt wejścia aplikacji.
folder *gen*	Wymagany folder, w którym są przechowywane automatycznie generowane pliki zasobów aplikacji.
gen/com.androidbook.myfirstandroidapp/R.java	Automatycznie wygenerowany plik źródłowy służący do zarządzania zasobami; nigdy nie powinien być edytowany ręcznie.
folder *res*	Wymagany folder, w którym są gromadzone wszystkie zasoby używane przez aplikację. Do tych zasobów zaliczamy: animacje, obrazy i inne pliki przeznaczone do wyświetlania, pliki układów, pliki XML, zasoby danych (takich jak łańcuchy znaków) oraz dowolne inne pliki.
res/drawable-/icon.png*	Foldery zasobów, w których są przechowywane ikony aplikacji dla urządzeń o różnych rozdzielczościach wyświetlaczy.
res/layout/main.xml	Pojedynczy plik układu.
res/values/strings.xml	Plik zasobów aplikacji zawierający łańcuchy znaków.
folder *assets*	Folder, w którym są przechowywane wszystkie „aktywa" aplikacji — czyli dane (zarówno pliki, jak i katalogi), które nie powinny być traktowane i zarządzane jako zasoby.

Oprócz tego w ramach tworzenia nowego projektu na dysku komputera jest także zapisywanych kilka innych plików. Niemniej jednak pliki wymienione w tabeli 3.1 są najważniejsze, a tworząc aplikacje, będziemy ich regularnie używali.

Tworzenie AVD na potrzeby projektu

Kolejnym krokiem jest utworzenie wirtualnego urządzenia, AVD, które w najlepszy sposób będzie opisywać urządzenie, jakie będziemy chcieli symulować podczas testowania i uruchamiania aplikacji. W tym przypadku wykorzystamy to samo AVD, które utworzyliśmy w celu uruchomienia przykładowej aplikacji Snake. AVD opisuje urządzenie, a nie aplikację. Dzięki temu tego samego AVD można używać podczas pisania i testowania wielu aplikacji. Istnieje także możliwość tworzenia podobnych

urządzeń wirtualnych, o tej samej konfiguracji, lecz zawierających różne dane (czyli różne zainstalowane aplikacje oraz różne dane na karcie pamięci).

> **Uwaga**
>
> Więcej informacji na temat tworzenia różnych rodzajów AVD oraz sposobów korzystania z emulatora Androida można znaleźć w dodatku A, „Krótki przewodnik po emulatorze Androida".

Tworzenie konfiguracji uruchomieniowych dla projektu

Kolejną czynnością będzie utworzenie konfiguracji uruchomieniowej oraz konfiguracji testowej służącej do debugowania projektu. Określają one odpowiednio, w jaki sposób oraz w jakich warunkach aplikacja będzie uruchamiana i debugowana. W ramach konfiguracji uruchomieniowej określane są opcje emulatora Androida oraz punkt wejścia do aplikacji.

Oba rodzaje konfiguracji — uruchomieniowa oraz do debugowania — są tworzone niezależnie od siebie i posiadają inne opcje. Zacznijmy od utworzenia konfiguracji uruchomieniowej.

Poniższa lista przedstawia czynności, jakie należy wykonać, by stworzyć konfigurację uruchomieniową dla aplikacji MyFirstAndroidApp:

1. Z menu Eclipse wybrać opcję *Run/Run Configuration* (lub kliknąć projekt prawym przyciskiem myszy i z wyświetlonego menu kontekstowego wybrać opcję *Run As*).
2. Dwukrotnie kliknąć opcję *Android Application*.
3. Określić nazwę konfiguracji — `MyFirstAndroidRunConfig`.
4. Wybrać uruchamiany projekt — kliknąć przycisk *Browse* i w wyświetlonym oknie dialogowym wybrać projekt *MyFirstAndroidApp*.
5. Przejść na zakładkę *Target* i w sekcji *Default Target Selection Mode* zaznaczyć przycisk opcji *Manual*.
6. Kliknąć przycisk *Apply*, a następnie *Close*.

> **Wskazówka**
>
> Jeżeli podczas definiowania konfiguracji uruchomieniowej lub testowej w sekcji *Default Target Configuration Mode* zostanie zaznaczony przycisk opcji *Automatic*, to jeśli tylko do komputera będzie podłączone urządzenie, aplikacja zostanie na nim zainstalowana i uruchomiona. W przeciwnym razie aplikacja zostanie zainstalowana i uruchomiona na emulatorze, na wskazanym wirtualnym urządzeniu AVD. Jeśli jednak zostanie zaznaczony przycisk opcji *Manual*, a nie *Automatic*, to Eclipse zawsze będzie nas pytać, czy: a) chcemy uruchomić aplikację w istniejącym emulatorze; b) chcemy uruchomić aplikację w nowym emulatorze i wskazać AVD, jakiego należy przy tym użyć; c) chcemy uruchomić aplikację na fizycznym urządzeniu (jeśli jest ono podłączone do komputera). Jeżeli jakiś emulator już działa, a następnie zostanie podłączone urządzenie, to jeśli w konfiguracji uruchomieniowej został zaznaczony przycisk opcji *Automatic*, opisywane opcje wyboru także zostaną wyświetlone.

Kolejną czynnością będzie utworzenie konfiguracji testowej, umożliwiającej debugowanie aplikacji. Proces ten jest bardzo podobny do opisanego powyżej. Oto lista czynności, jakie należy wykonać, by stworzyć konfigurację testową dla aplikacji MyFirstAndroidApp:

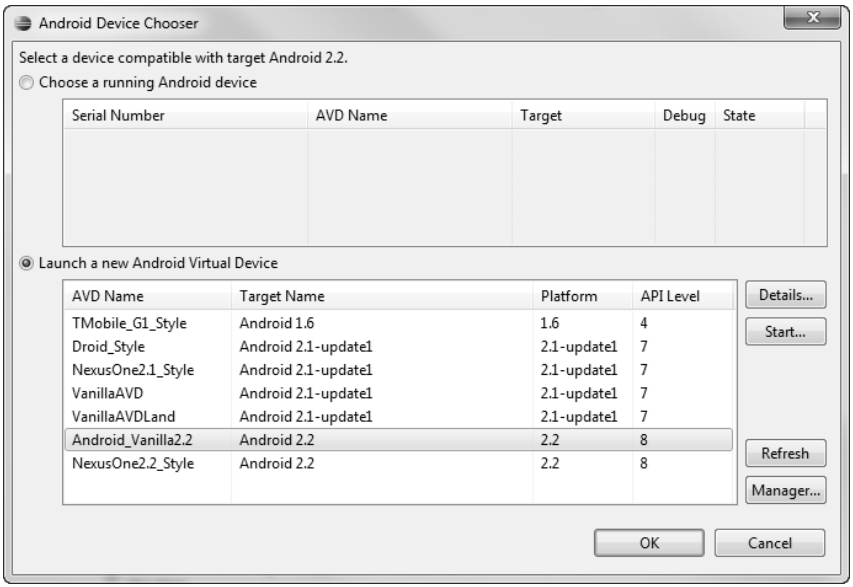
1. Z menu Eclipse wybrać opcję *Run/Debug Configuration* (lub kliknąć projekt prawym przyciskiem myszy i z wyświetlonego menu kontekstowego wybrać opcję *Debug As*).
2. Dwukrotnie kliknąć opcję *Android Application*.
3. Podać nazwę tworzonej konfiguracji — `MyFirstAndroidAppDebugConfig`.
4. Wybrać uruchamiany projekt — kliknąć przycisk *Browse* i w wyświetlonym oknie dialogowym wybrać projekt *MyFirstAndroidApp*.
5. Przejść na zakładkę *Target* i w sekcji *Default Target Selection Mode* zaznaczyć przycisk opcji *Manual*.
6. Kliknąć przycisk *Apply*, a następnie *Close*.

Teraz dysponujemy już konfiguracją, która pozwoli na debugowanie tworzonej aplikacji.

Uruchamianie aplikacji w emulatorze

Teraz możemy już uruchomić aplikację MyFirstAndroidApp; oto, co należy w tym celu zrobić:

1. Kliknąć strzałkę umieszczoną z prawej strony przycisku Run As znajdującego się na pasku narzędzi Eclipse (przedstawiającego zielone kółko z białym trójkącikiem) —  .
2. Z wyświetlonego menu wybrać utworzoną wcześniej konfigurację. (Jeśli nie będzie ona widoczna na liście, to należy wybrać umieszczoną w dolnej części menu opcję *Run Configuration…* i wskazać odpowiednią konfigurację w wyświetlonym oknie dialogowym. W tym przypadku wybrana konfiguracja pojawi się następnym razem w menu).
3. Ponieważ w konfiguracji wybraliśmy ręczny (*Manual*) tryb wyboru urządzenia, teraz zostaniemy poproszeni o wskazanie, na jakim urządzeniu aplikacja ma zostać uruchomiona. W wyświetlonym oknie dialogowym należy zaznaczyć przycisk opcji *Launch a new Android Virtual Device*, a następnie zaznaczyć wirtualne urządzenie, które chcemy uruchomić (patrz rysunek 3.9).
4. Po kliknięciu przycisku *OK* zostanie uruchomiony emulator Androida (może to trochę potrwać).

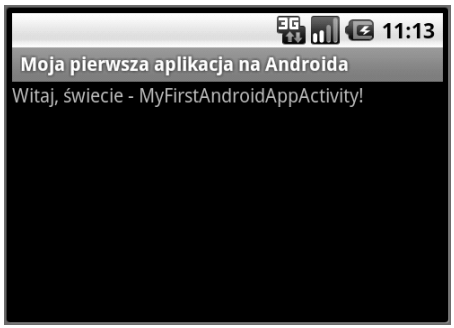


Rysunek 3.9. Ręczne określanie urządzenia, na jakim zostanie uruchomiona aplikacja

Wskazówka

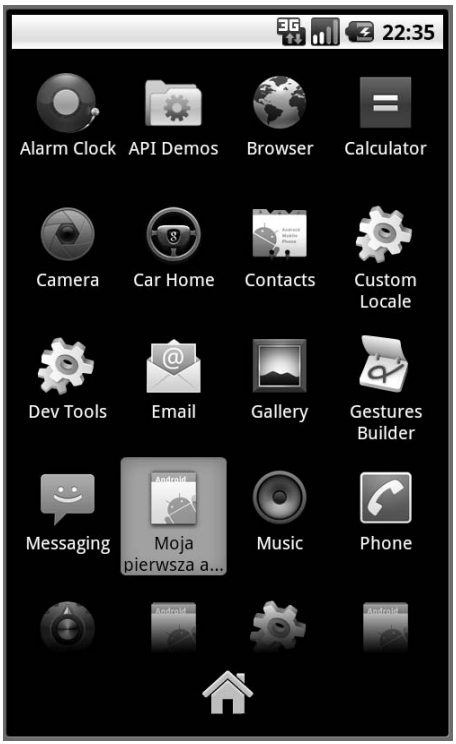
Uruchamianie emulatora może potrwać dosyć długo. Warto zatem pozostawić działający emulator i podłączać się do niego później, w razie takiej konieczności. Narzędzia dostępne w środowisku Eclipse obsługują ponowną instalację i uruchamianie aplikacji, dzięki czemu można zaoszczędzić czas i korzystać z wygody, jaką daje pozostawienie uruchomionego emulatora.

5. Kliknąć przycisk *Menu*, by odblokować emulator.
6. Bezpośrednio potem zostanie uruchomiona aplikacja, jak pokazano na rysunku 3.10.



Rysunek 3.10. Aplikacja Moja pierwsza aplikacja na Androida uruchomiona w emulatorze

7. Kliknąć przycisk *Home* w emulatorze, by zakończyć działanie aplikacji.
8. Wyświetlić listę zainstalowanych aplikacji, klikając ikonę listy aplikacji. Ekran emulatora powinien wyglądać podobnie do tego przedstawionego na rysunku 3.11.



Rysunek 3.11. Program Moja pierwsza aplikacja na Androida wyświetlony na liście aplikacji

9. Kliknąć ikonę *Moja pierwsza aplikacja na Androida*, aby ponownie uruchomić aplikację.

Debugowanie aplikacji w emulatorze

Zanim zajmiemy się kolejnymi zagadnieniami, Czytelnik koniecznie musi się dowiedzieć, jak można debugować aplikacje w emulatorze Androida. Aby mieć okazję do poznania i wypróbowania niektórych spośród dostępnych narzędzi do debugowania, celowo wprowadzimy błąd w kodzie naszej demonstracyjnej aplikacji.

W tym celu w Eclipse otwórzmy plik *MyFirstAndroidApp.java* i dodajmy do niego nową metodę o nazwie `forceError()`. Wywołanie tej metody należy umieścić wewnątrz kodu metody `onCreate()`. Metoda `forceError()` spowoduje zgłoszenie nowego, nieprzechwytywanego wyjątku.

Poniżej przedstawiony został jej kod:

```
public void forceError() {
    if(true) {
        throw new Error("O rany!");
    }
}
```

> **Wskazówka**
>
> Środowisko Eclipse zawiera osobne perspektywy (czyli grupy określonych paneli) służące do tworzenia kodu oraz debugowania aplikacji. Perspektywy można zmieniać, wybierając odpowiednie nazwy wyświetlone na wysokości paska narzędzi, z prawej strony okna środowiska. Perspektywa *Java* zawiera panele przeznaczone do tworzenia kodu oraz poruszania się po projekcie. Z kolei perspektywa *Debug* pozwala na definiowanie nowych punktów wstrzymania, przeglądanie informacji gromadzonych w dzienniku *LogCat* oraz debugowanie aplikacji. Kolejna perspektywa, *Dalvik Debug Monitor Service* (*DDMS*), pozwala monitorować i modyfikować emulator oraz bieżący stan uruchomionego w nim wirtualnego urządzenia.

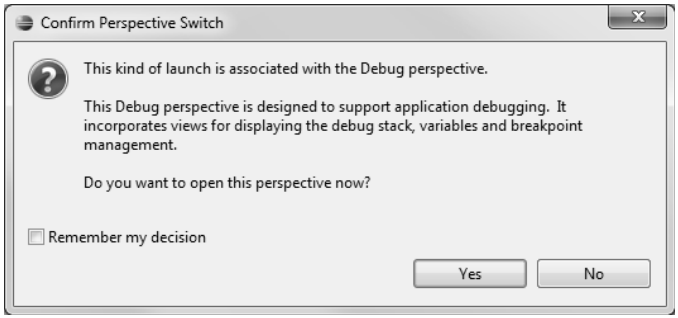
W tym momencie warto uruchomić aplikację i sprawdzić, co się stanie. W tym celu należy w pierwszej kolejności skorzystać z konfiguracji uruchomieniowej (*Run Configuration*). W emulatorze zobaczymy tylko tyle, że aplikacja została nieoczekiwanie zamknięta. Na ekranie emulatora zostanie wyświetlone okno dialogowe dające możliwość wymuszenia zamknięcia aplikacji (zostało ono przedstawione na rysunku 3.12).

Rysunek 3.12. Aplikacja Moja pierwsza aplikacja na Androida zgłaszająca informacje o problemach

Teraz zamknijmy aplikację oraz emulator. Kiedy to zrobimy, będziemy mogli przystąpić do debugowania aplikacji. Można to zrobić, wykonując następujące czynności:

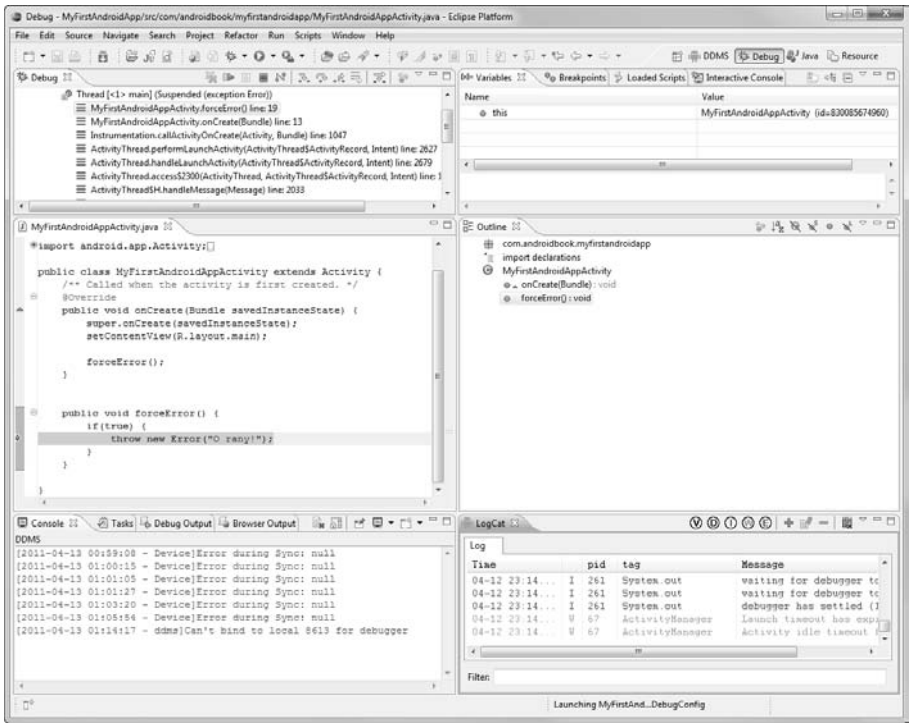
1. Kliknąć strzałkę widoczną z prawej strony przycisku *Debug As* umieszczonego na pasku narzędzi Eclipse (zawiera on ikonę zielonego robaka z czarną strzałką po prawej stronie) —  .
2. Rozwinąć menu i wybrać z niego utworzoną wcześniej konfigurację testową. (Jeśli nie będzie ona widoczna na liście, należy wybrać umieszczoną w dolnej części menu opcję *Debug Configuration…* i wskazać odpowiednią konfigurację w wyświetlonym oknie dialogowym. W tym przypadku wybrana konfiguracja pojawi się następnym razem w menu).
3. Wykonać te same czynności, które zostały przedstawione w opisie sposobu uruchamiania aplikacji — wybrać odpowiednie AVD, poczekać na uruchomienie emulatora i w razie potrzeby go odblokować.

Uruchomienie emulatora oraz dołączenie do niego debugera może chwilę zająć. Jeśli Czytelnik będzie debugował aplikację na Androida po raz pierwszy, zostanie wyświetlonych kilka okien dialogowych, takich jak to przedstawione na rysunku 3.13 — należy je przejrzeć i zamknąć.



Rysunek 3.13. Zmiana perspektywy w celu debugowania aplikacji uruchomionej w emulatorze Androida

W środowisku Eclipse można skorzystać z perspektywy *Debug*, aby ustawiać punkty wstrzymania (ang. *breakpoint*), wykonywać program instrukcja po instrukcji oraz przeglądać informacje zarejestrowane w dzienniku *LogCat*. Tym razem gdy pojawią się problemy w aplikacji, korzystając z debugera, będzie można określić ich przyczynę. Być może w celu skonfigurowania debugera konieczne będzie przejrzenie i zamknięcie kilku okien dialogowych. Jeśli zezwolimy na dalsze wykonywanie aplikacji po zgłoszeniu wyjątku, to wyniki jej działania zostaną wyświetlone w perspektywie *Debug*. Przeglądając informacje wyświetlone w panelu *LogCat*, dowiemy się, że aplikacja została zamknięta z powodu nieobsłużonego wyjątku (patrz rysunek 3.14).



Rysunek 3.14. Debugowanie przykładowej aplikacji Moja pierwsza aplikacja na Androida

Konkretnie rzecz biorąc, na czerwono został wyświetlony błąd `AndroidRuntime`: `java.lang.Error: O rany!`.

Wróćmy do emulatora, klikając przycisk *Force Close*. Spróbujmy ustawić punkt wstrzymania na metodzie `forceError()`. W tym celu wystarczy kliknąć wybrany wiersz kodu prawym przyciskiem myszy i z wyświetlonego menu kontekstowego wybrać opcję *Toggle Breakpoint* (alternatywną metodą jest naciśnięcie kombinacji klawiszy *Ctrl+Shift+B*).

Wskazówka

W środowisku Eclipse można krokowo wykonywać kolejne instrukcje, używając poleceń *Step Into* (*F5*), *Step Over* (*F6*), *Step Return* (*F7*) oraz *Resume* (*F8*).

W systemie Mac OS X może się okazać, że klawisz *F8* jest globalnie skojarzony z pewnymi operacjami. W takim przypadku, aby w Eclipse skorzystać z możliwości ułatwionego wykonywania czynności poprzez naciskanie odpowiedniego klawisza, trzeba będzie zmienić skojarzenia klawiszy z operacjami. W tym celu należy wybrać z menu opcje *Windows/Preferences/General/Keys*, a następnie odnaleźć polecenie, z jakim jest skojarzony klawisz *F8*, i je zmienić. Alternatywnym rozwiązaniem będzie zmiana globalnego skojarzenia wykorzystującego klawisz *F8*. Aby to zrobić w systemie Mac OS X, należy wyświetlić *System Preferences/ Keyboard & Mouse/Keyboard Shortcuts*, a następnie zmienić operację skojarzoną z klawiszem *F8*.

Teraz powinniśmy wrócić do emulatora, ponownie uruchomić w nim naszą przykładową aplikację i zacząć wykonywać ją instrukcja po instrukcji. Przekonamy się, że aplikacja zgłasza wyjątek, który następnie zostanie wyświetlony w panelu *Variable Browser* wchodzącym w skład perspektywy *Debug*. Rozwinięcie tej zmiennej pozwoli wyświetlić zawartość błędu `"O rany!"`.

To doskonały moment, by kilkukrotnie doprowadzić do awarii aplikacji i przyzwyczaić się do przycisków i opcji pozwalających na sterowanie jej wykonywaniem. A skoro już zajmujemy się tymi zagadnieniami, wróćmy do perspektywy *DDMS*. Warto zwrócić uwagę, że emulator udostępnia listę procesów aktualnie działających na telefonie, takich jak `system_process` lub `com.android.phone`. Jeśli uruchomimy aplikację MyFirstAndroidApp, to na tej liście pojawi się opcja `com.androidbook.myfirstandroidapp`. Wystarczy teraz wymusić zamknięcie aplikacji, a odpowiadająca jej pozycja na liście zniknie. Perspektywy *DDMS* można używać do wymuszania zakończenia procesów, sprawdzania istniejących wątków oraz sterty, jak również do przeglądania systemu plików telefonu.

Dodawanie mechanizmów rejestracji do aplikacji

Zanim Czytelnik zacznie badać różne możliwości Android SDK, koniecznie powinien poznać możliwości rejestracji, jest to bowiem nieocenione narzędzie, które można wykorzystywać nie tylko do debugowania, lecz także poznawania Androida. Możliwości rejestracji zapewnia klasa `Log` wchodząca w skład pakietu `android.util`.

Kilka użytecznych metod tej klasy zostało przedstawionych w tabeli 3.2.

Tabela 3.2. *Użyteczne metody klasy android.util.Log*

Metoda	Przeznaczenie
`Log.e()`	Rejestruje błędy.
`Log.w()`	Rejestruje ostrzeżenia.
`Log.i()`	Rejestruje komunikaty informacyjne.
`Log.d()`	Rejestruje komunikaty do debugowania.
`Log.v()`	Rejestruje rozszerzone komunikaty informacyjne.

Wskazówka

Środowisko Eclipse stara się oszczędzać czas programistów — pozwala używać importowanych klas, a następnie dodać odpowiednie instrukcje importu automatycznie. Wystarczy wskazać nazwę klasy myszką i w wyświetlonym okienku dialogowym wybrać opcję *Import 'Log' (android.util)*.

Oprócz tego można także skorzystać z polecenia *Organize imports* (w systemie Windows można je wykonać, naciskając kombinację klawiszy *Ctrl+Shift+O*, a w Mac OS X *Command+Shift+O*), które sprawi, że Eclipse automatycznie doda wszelkie niezbędne instrukcje `import` dla używanych klas i pakietów oraz usunie te, które nie są już potrzebne. Jeśli pojawią się jakieś konflikty nazw, co w przypadku klas o nazwie `Log` zdarza się stosunkowo często, można określić pakiet, do którego należy importowana klasa.

Aby dodać możliwość rejestracji komunikatów do naszej aplikacji MyFirstAndroidApp, wystarczy otworzyć plik *MyFirstAndroidApp.java* w edytorze. W pierwszej kolejności na górze pliku należy dodać odpowiednią instrukcję `import`, która umożliwi korzystanie z klasy `Log`:

```
import android.util.Log;
```

Następnie w klasie `MyFirstAndroidApp` należy zadeklarować stałą łańcuchową, która będzie używana do oznaczania wszystkich rejestrowanych komunikatów. Później będziemy mogli skorzystać z możliwości, jakie daje panel *LogCat* Eclipse, by wyświetlać w nim tylko te komunikaty, które mają naszą etykietę:

```
private static final String DEBUG_TAG = "MyFirstAppLogging";
```

Teraz w kodzie metody `onCreate()` możemy już zarejestrować jakiś komunikat informacyjny:

```
Log.i(DEBUG_TAG, "Informacja o aplikacji MyFirstAndroidApp");
```

Ostrzeżenie

Skoro już wróciliśmy do kodu metody `onCreate()`, to Czytelnik musi pamiętać, by umieścić w komentarzu wywołanie metody `forceError()`, aby aplikacja ponownie zaczęła działać prawidłowo.

Teraz jesteśmy już gotowi do uruchomienia naszej aplikacji. A zatem zapiszmy zmiany wprowadzone w jej kodzie źródłowym i uruchommy ją w emulatorze, korzystając z konfiguracji do debugowania. Należy zwrócić uwagę, że na liście w panelu *LogCat* pojawił się nasz komunikat informacyjny, oznaczony etykietą `MyFirstAppLogging` wyświetloną w kolumnie `tag` (patrz rysunek 3.15).

Rysunek 3.15. Lista komunikatów LogCat przefiltrowana tak, by wyświetlać wyłącznie komunikaty z etykietą MyFirstAppLogging

Wskazówka

Czytelnik może uznać, że warto utworzyć w panelu *LogCat* specjalny filtr, który spowoduje, że będą w nim wyświetlane wyłącznie komunikaty posiadające jego etykietę. W tym celu należy kliknąć ikonę przedstawiającą zielony znak plusa, umieszczoną w prawej górnej części panelu *LogCat*. Następnie należy podać nazwę tworzonego filtra, na przykład `MyFirstApp`, i wpisać wartość `MyFirstAppLogging` w polu `by Log Tag`. Teraz w panelu *LogCat* pojawiła się druga zakładka, prezentująca jedynie komunikaty wybrane przez nas.

Dodawanie obsługi multimediów do aplikacji

Kolejnym testem będzie próba dodania do naszej aplikacji niewielkich „wodotrysków" — konkretnie rzecz biorąc, spróbujemy odtworzyć w niej plik muzyczny zapiany w formacie MP3. Możliwości odtwarzania multimediów w Androidzie zostały zaimplementowane w klasie `MediaPlayer` należącej do pakietu `android.media`.

Obiekt `MediaPlayer` można utworzyć na podstawie istniejących zasobów lub określając adres pliku docelowego w postaci identyfikatora URI. W celu jak największego uproszczenia przykładu zaczniemy od wskazania pliku MP3 przy użyciu identyfikatora URI, czyli obiektu klasy `Uri` należącej do pakietu `android.net`.

Wybrane metody klas `android.media.MediaPlayer` oraz `android.net.Uri` zostały przedstawione w tabeli 3.3.

Tabela 3.3. *Ważne metody klasy MediaPlayer oraz metody do przetwarzania URI*

Metoda	Przeznaczenie
`MediaPlayer.create()`	Tworzy nowy odtwarzacz muzyczny, który będzie operował na wskazanym pliku.
`MediaPlayer.start()`	Rozpoczyna odtwarzanie pliku.
`MediaPlayer.stop()`	Zatrzymuje odtwarzanie.
`MediaPlayer.release()`	Zwalnia zasób używany przez dany obiekt odtwarzacza.
`Uri.parse()`	Tworzy obiekt `Uri` na podstawie prawidłowo sformułowanego adresu URI.

Aby dodać do naszej przykładowej aplikacji możliwość odtwarzania plików MP3, trzeba wprowadzić pewne modyfikacje w pliku *MyFirstAndroidApp.java*. W pierwszej kolejności należy dodać instrukcje `import`, które pozwolą korzystać z klasy `MediaPlayer`:

```
import android.media.MediaPlayer;
import android.net.Uri;
```

Następnie w kodzie klasy `MyFirstAndroidApp` należy zadeklarować składową, w której będzie przechowywany obiekt `MediaPlayer`:

```
private MediaPlayer mp;
```

Kolejną czynnością będzie dodanie metody o nazwie `playMusicFromWeb()` i wywołanie jej wewnątrz metody `onCreate()`. Metoda ta w pierwszej kolejności utworzy obiekt `Uri`, następnie obiekt `MediaPlayer`, a następnie rozpocznie odtwarzanie wskazanego pliku MP3. Jeśli z jakiegokolwiek powodu operacji nie uda się wykonać prawidłowo, to metoda ta zarejestruje w dzienniku odpowiedni komunikat opatrzony naszą utworzoną wcześniej etykietą.

Poniżej przedstawiony został kod metody `playMusicFromWeb()`:

```
public void playMusicFromWeb() {
    try {
        Uri file = Uri.parse("http://www.perlgurl.org/podcast/archives/
        ↪podcasts/PerlgurlPromo.mp3");
        mp = MediaPlayer.create(this, file);
```

```
            mp.start();
        }
        catch (Exception e) {
            Log.e(DEBUG_TAG, "Błąd odtwarzacza", e);
        }
    }
```

I w końcu, w momencie zamykania aplikacji będziemy chcieli usunąć z pamięci wszystkie używane wcześniej obiekty. W tym celu należy przesłonić metodę `onStop()`, a wewnątrz niej zatrzymać odtwarzacz i usunąć z pamięci wszystkie jego zasoby.

> **Wskazówka**
>
> W środowisku Eclipse można kliknąć prawym przyciskiem myszy wewnątrz kodu klasy i z wyświetlonego menu kontekstowego wybrać opcję *Source* (alternatywnym rozwiązaniem jest naciśnięcie kombinacji klawiszy *Alt+Shift+S*). W kolejnym menu należy wybrać opcję *Override/Implement Methods*, a następnie w wyświetlonym oknie dialogowym zaznaczyć pole wyboru przy metodzie `onStop()` i kliknąć przycisk *OK*.

Nasza wersja metody `onStop()` powinna wyglądać w następujący sposób.

```
protected void onStop() {
    if (mp != null) {
       mp.stop();
       mp.release();
    }
    super.onStop();
}
```

Kiedy teraz uruchomimy aplikację MyFirstAndroidApp w emulatorze (i jeśli dodatkowo będziemy dysponowali połączeniem z internetem, by pobrać plik MP3 określony przez podany w kodzie identyfikator URI), aplikacja zacznie odtwarzać plik muzyczny. Po zamknięciu aplikacji odtwarzacz `MediaPlayer` jest zatrzymywany, a jego zasoby prawidłowo usuwane z pamięci.

Dodawanie do aplikacji usług lokalizacyjnych

Nasza aplikacja już wie, jak się przywitać, ale nie wie, gdzie się znajduje. A zatem nadarza się chyba dobra okazja, by zapoznać się z niektórymi podstawowymi możliwościami pobierania współrzędnych GPS.

Tworzenie AVD wyposażonego w interfejsy API firmy Google

Aby móc pobawić się nieco usługami lokalizacyjnymi oraz skorzystać z możliwości integracji aplikacji z mapami, konieczne będzie wykorzystanie niektórych aplikacji firmy Google dostępnych na telefonach z Androidem — konkretnie rzecz biorąc, chodzi o aplikację Google Maps. Aby móc z niej korzystać, musimy utworzyć następne wirtualne urządzenie. To nowe AVD powinno mieć dokładnie takie same parametry i ustawienia jak nasze pierwsze AVD, Android_Vanilla2.2, z jednym wyjątkiem — w jego przypadku z listy

Target należy wybrać odpowiednik interesującego nas poziomu API (którym jest API Level 8) przygotowany przez Google (czyli opcję *Google Apis (Google Inc.) — API Level 8*). Temu nowemu AVD możemy nadać nazwę na przykład *Android_with_GoogleAPIs_2.2*.

Konfiguracja lokalizacji w emulatorze

Po utworzeniu nowego AVD wyposażonego w obsługę interfejsów API firmy Google konieczne jest zamknięcie aktualnie działającego emulatora. Następnie ponownie należy rozpocząć debugowanie naszej testowej aplikacji, jednak tym razem wybierając nowe AVD.

Emulator nie jest wyposażony w GPS, więc konieczne jest ręczne podanie współrzędnych określających jego początkowe położenie. W tym celu należy uruchomić emulator w trybie debugowania, wykorzystując przy tym AVD obsługujące interfejsy API Google'a, a następnie wykonać następujące czynności:

W emulatorze:

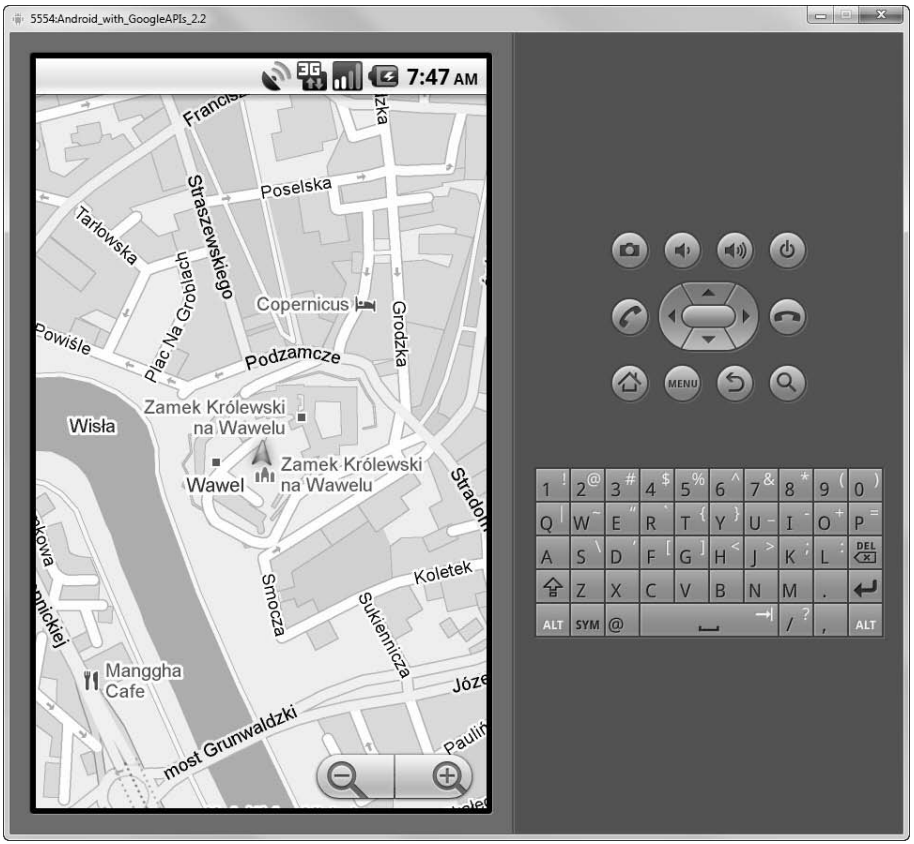
1. Nacisnąć przycisk *Home*, by wrócić do głównego ekranu.
2. Otworzyć listę aplikacji i uruchomić aplikację *Maps*.
3. Kliknąć przycisk *Menu*.
4. Wybrać opcję *Moja lokalizacja* (wyglądającą jak celownik z kółkiem w środku).

W Eclipse:

1. Kliknąć perspektywę *DDMS* umieszczoną w prawym górnym wierzchołku okna.
2. Z lewej strony okna Eclipse widoczny będzie panel *Emulator Control*. Należy go przewinąć w dół, tak by pojawiła się sekcja *Location Controls*.
3. W polach *Longitude* i *Latitude* wpisać odpowiednio długość i szerokość geograficzną. (Warto zwrócić uwagę, że ich kolejność jest zamieniona w stosunku do zwyczajowego sposobu podawania współrzędnych geograficznych).
4. Kliknąć przycisk *Send*.

> **Wskazówka**
>
> Aby określić konkretne współrzędne geograficzne, można wyświetlić w przeglądarce stronę *http://maps.google.pl*, po czym odnaleźć interesujące nas miejsce, kliknąć je prawym przyciskiem myszy i wybrać opcję *Co tu jest* z menu kontekstowego. Współrzędne wybranego miejsca pojawią się w polu tekstowym nad mapą. Na przykład odszukajmy Wzgórze Wawelskie, a po jego kliknięciu w polu tekstowym nad mapą zostaną wyświetlone współrzędne: *50,054005,19,935154*, czyli szerokość geograficzna wynosi *50,054005*, a długość *19,935154*.

Jeśli teraz wrócimy do emulatora, zauważymy, że aplikacja Google Maps pokazuje lokalizację, której współrzędne podaliśmy w Eclipse. Na ekranie telefonu w emulatorze powinna być widoczna mapa Krakowa, pokazana na rysunku 3.16.



Rysunek 3.16. Określanie położenia Wzgórza Wawelskiego w emulatorze

Jak widać, emulator wskazuje teraz symulowane położenie.

Odnajdowanie ostatniego znanego położenia

Aby dodać obsługę lokalizacji do naszej demonstracyjnej aplikacji MyFirstAndroidApp, trzeba będzie wprowadzić kilka zmian w pliku *MyFirstAndroidApp.java*. Przede wszystkim konieczne będzie dodanie odpowiednich instrukcji `import`:

```
import android.location.Location;
import android.location.LocationManager;
```

Następnie należy zadeklarować w klasie nową metodę — `getLocation()` — i wywołać ją w metodzie `onCreate()`. Metoda będzie pobierać ostatnią znaną lokalizację telefonu i wyświetlać ją w formie komunikatu zapisywanego w dzienniku. Jeśli z jakichkolwiek powodów operacji nie uda się wykonać prawidłowo, w dzienniku zostanie zapisana informacja o błędzie.

Poniżej pokazano, jak powinien wyglądać kod metody `getLocation()`:

```
public void getLocation() {
    try {
        LocationManager locMgr = (LocationManager)
            getSystemService(LOCATION_SERVICE);
        Location recentLoc = locMgr.
            getLastKnownLocation(LocationManager.GPS_PROVIDER);
        Log.i(DEBUG_TAG, "lokalizacja: " + recentLoc.toString());
    }
    catch (Exception e) {
        Log.e(DEBUG_TAG, "Nie udało się określić położenia", e);
    }
}
```

Dodatkowo nasza aplikacja wymaga odpowiednich uprawnień, by móc korzystać z możliwości określania położenia geograficznego. Uprawnienia te należy zarejestrować w pliku *AndroidManifest.xml*. Oto, co należy zrobić, by dodać do aplikacji uprawnienia pozwalające na korzystanie z usług lokalizacyjnych:

1. Dwukrotnie kliknąć plik *AndroidManifest.xml*.
2. Przejść na zakładkę *Permissions*.
3. Kliknąć przycisk *Add* i wybrać opcję *Uses Permission*.
4. W prawym panelu wybrać opcję `android.permission.ACCESS_FINE_LOCATION`.
5. Zapisać plik.

Jeśli teraz uruchomimy naszą przykładową aplikację w emulatorze, zarejestruje ona w dzienniku komunikat ze współrzędnymi, które przesłaliśmy wcześniej do emulatora; jego treść będzie można przeczytać w panelu *LogCat* w Eclipse.

Debugowanie aplikacji na fizycznym urządzeniu

Czytelnik opanował już sposoby uruchamiania aplikacji przy wykorzystaniu emulatora. Nadszedł zatem czas, by spróbować uruchomić ją na prawdziwym telefonie. W tym celu w pierwszej kolejności trzeba będzie oznaczyć aplikację jako przeznaczoną do debugowania. Stosowne informacje należy podać w pliku *AndroidManifest.xml*. Aby wprowadzić te zmiany, należy wykonać następujące operacje:

1. Dwukrotnie kliknąć plik *AndroidManifest.xml*.
2. Przejść na zakładkę *Application*.
3. Atrybutowi *Debuggable* przypisać wartość *True*.
4. Zapisać plik.

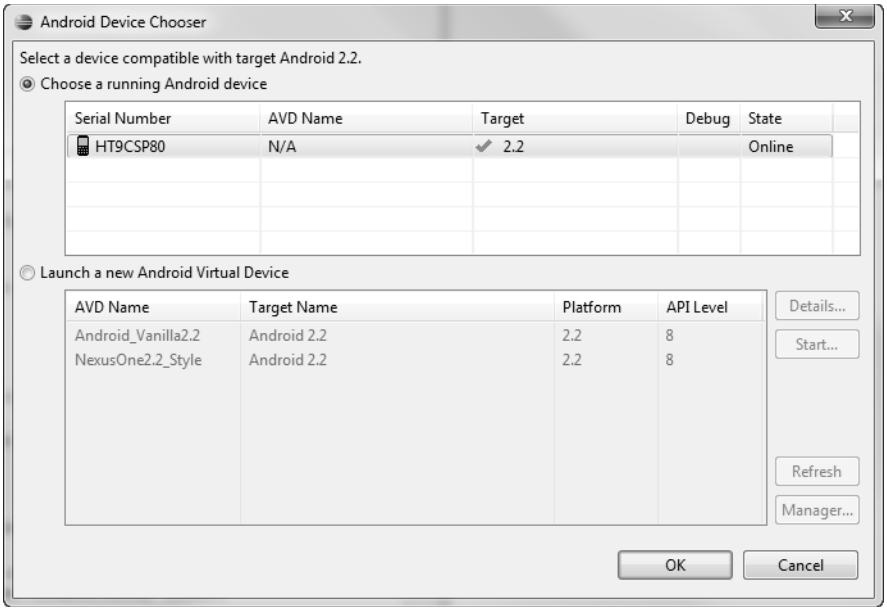
Innym rozwiązaniem jest wprowadzenie analogicznych zmian bezpośrednio w kodzie pliku *AndroidManifest.xml*, a konkretnie w atrybucie `android:debuggable`:

```
<application ... android:debuggable="true">
```

Jeśli zapomnimy przypisać atrybutowi `debuggable` wartości `true`, telefon będzie pokazywać okienko dialogowe z komunikatem o oczekiwaniu na podłączenie debugera

aż do momentu naciśnięcia przycisku *Force Close* i wprowadzenia odpowiednich zmian w pliku manifestu aplikacji.

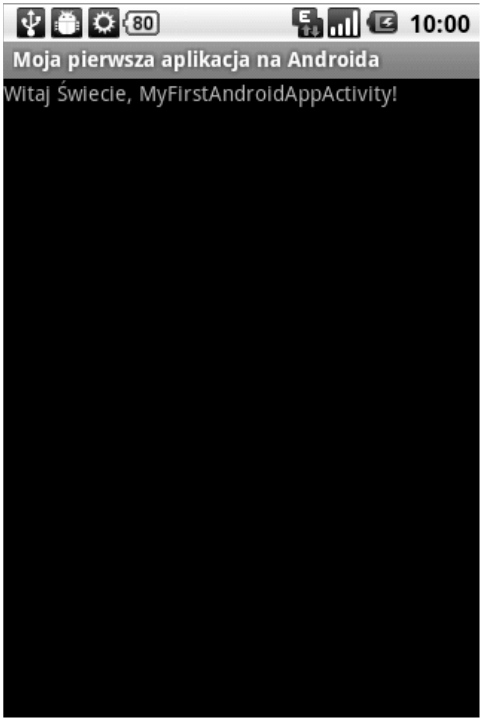
Teraz można już podłączyć telefon do komputera przy użyciu kabla USB i spróbować ponownie uruchomić aplikację (korzystając z konfiguracji uruchomieniowej lub testowej). Ponieważ w obu konfiguracjach ustawiliśmy ręczny tryb wyboru urządzenia, powinniśmy zobaczyć podłączone urządzenie jako opcję na liście działających urządzeń — *Choose a running Android device* (widocznej na rysunku 3.17).



Rysunek 3.17. Lista umożliwiająca wybór urządzenia, prezentująca telefon podłączony do komputera przy użyciu kabla USB

Teraz wystarczy wybrać telefon jako docelowe urządzenie, na którym chcemy uruchomić aplikację. W efekcie aplikacja zostanie skopiowana na telefon i uruchomiona tak samo jak wcześniej na emulatorze. Zakładając, że wcześniej zezwoliliśmy także na debugowanie aplikacji w telefonie, będziemy mogli analizować jej działanie tak samo jak na emulatorze. O tym, że telefon aktywnie używa połączenia w trybie debugowania, można się przekonać po ikonie „androidowego robaka” () wyświetlonej na pasku powiadomień telefonu. Rysunek 3.18 przedstawia zrzut aplikacji uruchomionej na prawdziwym telefonie.

Debugowanie aplikacji na telefonie odbywa się dokładnie tak samo jak w emulatorze, choć różni się od niego pod kilkoma względami. Nie można korzystać z kontrolek emulatora, by przesyłać do telefonu na przykład SMS-y lub informacje o położeniu geograficznym; można natomiast wykonywać analogiczne czynności — czyli napisać SMS lub włączyć odbiornik GPS, by określić położenie.



Rysunek 3.18. Aplikacja MyFirstAndroidApp uruchomiona na rzeczywistym telefonie

Podsumowanie

W tym rozdziale pokazano, jak można tworzyć, budować, uruchamiać i debugować aplikacje na Androida przy wykorzystaniu środowiska Eclipse. Zaczęliśmy od przetestowania środowiska programistycznego, wykorzystując do tego aplikację przykładową dostarczoną wraz Android SDK, a następnie stworzyliśmy zupełnie nową aplikację. Czytelnik dowiedział się także, jak wprowadzać proste modyfikacje w aplikacji, i zobaczył kilka fascynujących możliwości Androida, które zostaną bardziej szczegółowo opisane w dalszej części książki.

W kilku kolejnych rozdziałach przedstawione zostaną bardziej szczegółowe informacje na temat definiowania możliwości i działania aplikacji przy użyciu pliku manifestu oraz opisany cykl działania aplikacji. Oprócz tego znajdą się w nich również zagadnienia związane z zarządzaniem zasobami, które będą używane w aplikacji, takimi jak obrazy oraz łańcuchy znaków.

Zasoby i inne źródła informacji

Sterowniki USB dla systemu Windows:

http://developer.android.com/sdk/win-usb.html

Poradnik dla programistów aplikacji na Androida:

http://developer.android.com/guide/developing/device.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

Z

Ż

WYCZERPUJĄCY PRZEWODNIK, GRUNTOWNIE ZAKTUALIZOWANY W OPARCIU O NAJNOWSZĄ WERSJĘ ANDROID SDK I NAJLEPSZE TECHNIKI PROGRAMOWANIA!

Android

PROGRAMOWANIE APLIKACJI NA URZĄDZENIA PRZENOŚNE

Gdy w 2008 roku Google opublikował Androida, rynkiem telefonów komórkowych wprawdzie nieco zatrzęsło, ale nikt nie spodziewał się, że ta platforma aż tak poważnie zagrozi takim gigantom mobilnych systemów operacyjnych, jak iOS Apple, Windows Mobile, Symbian czy RIM BlackBerry. Od tego czasu każde nowe urządzenie z Androidem miało coraz większe możliwości i było jeszcze bardziej ekscytujące od swoich poprzedników. Minęły zaledwie trzy lata od dnia, kiedy na rynku pojawił się pierwszy telefon z systemem Android — T-Mobile G1, stworzony przez firmę HTC — a system ten już okrzyknięty został najszybciej sprzedającą się platformą dla telefonów przenośnych. To oczywiście nie pozostaje bez echa: potrzeba coraz więcej ludzi specjalizujących się w programowaniu aplikacji na tę fascynującą, darmową i otwartą platformę mobilną.

Oto kompletny podręcznik, zawierający wszystko, co potrzebne do tworzenia, wdrażania i sprzedawania aplikacji na urządzenia przenośne działające pod kontrolą najnowszych wersji Androida. Autorzy — w oparciu o swoje wieloletnie doświadczenie w tworzeniu mobilnych aplikacji — wprowadzą Cię we wszystkie etapy tego procesu: pomysł, projektowanie, pisanie kodu, testowanie, pakowanie i rozpowszechnianie aplikacji. Poznasz doskonale specyfikację platformy Android, podstawowe zasady efektywnego projektowania aplikacji na nią przeznaczonych oraz najlepsze praktyki związane z tworzeniem wygodnych interfejsów użytkownika. Znajdziesz tu także wyczerpujące opisy wszystkich kluczowych interfejsów programistycznych: do obsługi składowania danych, komunikacji sieciowej, obsługi rozmów telefonicznych, usług lokalizacyjnych, multimediów, grafiki 3D oraz opcjonalnych komponentów sprzętowych. Oprócz tego książka została uzupełniona praktycznymi sztuczkami, które pozwolą Ci zaoszczędzić sporo cennego czasu i uniknąć wielu niepotrzebnych pułapek!

Ponadto znajdziesz tu:

- kilka rozdziałów opisujących interfejs API do obsługi technologii związanych z WWW, a także Android NDK, poszerzanie zasięgu aplikacji, zarządzanie użytkownikami, synchronizację danych, tworzenie kopii bezpieczeństwa, zaawansowane metody wprowadzania danych
- wyczerpujące informacje na temat plików manifestu, dostawców treści, projektowania i testowania aplikacji
- prezentację najbardziej aktualnych i interesujących zagadnień, takich jak obsługa komunikacji Bluetooth i gestów, rozpoznawanie mowy, widgety, technologie Live Folders, Live Wallpapers oraz globalne wyszukiwanie
- aktualne informacje na temat generowania grafiki 3D przy użyciu OpenGL ES 2.0
- zagadnienia związane z zapewnianiem zgodności pomiędzy różnymi urządzeniami

Shane Conder ma szerokie doświadczenia programistyczne. Zaprojektował i napisał wiele komercyjnych aplikacji na Androida, iPhone'a, BREW, BlackBerry, Palm oraz środowiska J2ME i Windows Mobile; część z nich została zainstalowana na milionach telefonów działających na całym świecie.

Lauren Darcey jest kierowniczką do spraw technicznych oraz wyznacza kierunki działalności w firmie programistycznej zajmującej się technologiami mobilnymi (w tym Androidem, iPhone'em, BlackBerry czy J2ME) i działalnością konsultingową. Dysponując ponaddwudziestoletnim doświadczeniem, Lauren uznawana jest za niekwestionowany autorytet w dziedzinie architektury aplikacji oraz tworzenia komercyjnych aplikacji mobilnych.

ISBN 978-83-246-3349-4

Cena 149,00 zł