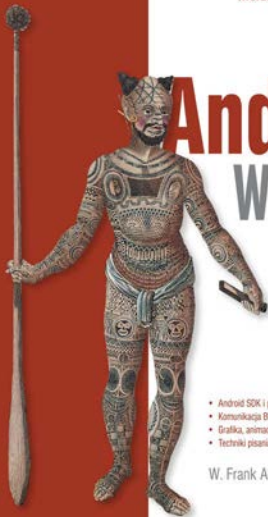


Rozbudowany podręcznik  
tworzenia niesamowitych aplikacji  
dla systemu Android!



# Android

## W AKCJI

WYDANIE II

- Android SDK i programowanie aplikacji WWW
- Komunikacja Bluetooth i przetwarzanie danych z czujników
- Grafika, animacja i multimedia w Androidzie
- Techniki pisania natywnych aplikacji w języku C

W. Frank Ableson, Robi Sen, Chris King

## » Idź do

- Spis treści
- Przykładowy rozdział
- Skorowidz

## » Katalog książek

- Katalog online
- Zamów drukowany katalog

## » Twój koszyk

- Dodaj do koszyka

## » Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

## » Czytelnia

- Fragmenty książek online

## » Kontakt

Helion SA  
ul. Kościuszki 1c  
44-100 Gliwice  
tel. 32 230 98 63  
e-mail: helion@helion.pl  
© Helion 1991–2011

## Android w akcji. Wydanie II

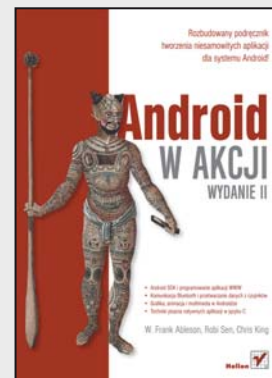
Autorzy: W. Frank Ableson, Robi Sen, Chris King

Tłumaczenie: Paweł Gonera

ISBN: 978-83-246-3380-7

Tytuł oryginału: [Android in Action](#)

Format: 168×237, stron: 624



### Rozbudowany podręcznik tworzenia niesamowitych aplikacji dla systemu Android!

- Android SDK i programowanie aplikacji WWW
- Komunikacja Bluetooth i przetwarzanie danych z czujników
- Grafika, animacja i multimedia w Androidzie
- Techniki pisania natywnych aplikacji w języku C

Skoro zwróciłeś uwagę właśnie na tę książkę, zapewne dobrze wiesz, czym jest Android i co potrafi – teraz przyszła pora, abyś sprawdził go także w akcji! Oto doskonała książka dla wszystkich programistów, którym marzy się tworzenie własnych aplikacji dla robiącego oszałamiającą karierę systemu. Choć ta książka nie jest przeznaczona dla początkujących, zawiera wszystkie informacje potrzebne osobom, dla których Android jest całkowicie nowym środowiskiem. Można w niej znaleźć instrukcje niezbędne do szybkiego zorientowania się w architekturze tej platformy oraz sposobie jej działania, co pozwoli sprawnie rozpocząć pracę w tym środowisku. Pozostałe rozdziały to już czysta frajda programowania!

Od czego zaczniesz zabawę z Androidem? Dowiesz się, jak budować aplikacje dla tego systemu od najmniejszych cegiełek aż po ekrany, dodawać funkcje telefoniczne i wykorzystywać bibliotekę OpenGL ES do tworzenia złożonej grafiki 2D oraz 3D. Następnie poznasz zasady tworzenia większych aplikacji oraz techniki pisania aplikacji w języku C, także z użyciem Android Native Development Kit. Opanujesz potężne narzędzie Android SDK oraz budowanie aplikacji dla WebKit z użyciem HTML 5, a nawet nauczysz się rozszerzać lub zastępować wbudowane funkcje Androida na podstawie użytecznych i intrygujących przykładów.

- Wprowadzenie do systemu Android
- Środowisko programowania
- Komponenty interfejsu użytkownika, w tym View i Layout
- Metody przechowywania i odczytywania lokalnych danych
- Sieci oraz usługi sieciowe, Bluetooth, sensory i widżety aplikacji
- Przegląd podstawowych zagadnień związanych z telefonią
- Powiadomienia i alarmy
- Grafika i animacja w Androidzie
- Korzystanie z funkcji multimedialnych Androida
- Usługi oparte na lokalizacji
- Integracja kontaktów z platformy społecznościowej
- Aplikacja wspomagająca serwisantów
- Budowanie aplikacji Android w języku C
- Tworzenie stron WWW dla systemu Android
- Strategie lokalizowania aplikacji
- Android Native Development Kit oraz korzystanie z SDK oraz AVD Manager

**Wkrocz wreszcie do akcji i zacznij tworzyć własne aplikacje dla Androida!**

# *Spis treści*

<b>Wprowadzenie</b>	<b>13</b>
<b>Wprowadzenie do pierwszego wydania</b>	<b>15</b>
<b>Podziękowania</b>	<b>17</b>
<b>O książce</b>	<b>19</b>
<b>O ilustracji na okładce</b>	<b>25</b>
<b>Część I. Czym jest Android? Zarys ogólny</b>	<b>27</b>
<b>1. Wprowadzenie do systemu Android</b>	<b>29</b>
1.1. Platforma Android	30
1.2. Omówienie rynku Androida	32
1.2.1. Operatorzy telefonii komórkowej	32
1.2.2. Android kontra zaawansowane telefony	32
1.2.3. Android kontra smartfony	34
1.2.4. Android kontra Android	35
1.2.5. Licencjonowanie Androida	35
1.3. Warstwy systemu Android	36
1.3.1. Budowanie na bazie jądra Linux	38
1.3.2. Praca w maszynie wirtualnej Dalvik	39
1.4. Intencje w programowaniu dla Androida	39
1.4.1. Wykorzystywanie intuicyjnego interfejsu użytkownika	40
1.4.2. Sposób działania intencji	40

1.5.	Cztery rodzaje komponentów Androida .....	43
1.5.1.	Klasa Activity .....	44
1.5.2.	Klasa Service .....	45
1.5.3.	Klasa BroadcastReceiver .....	46
1.5.4.	Klasa ContentProvider .....	49
1.6.	Budowa pliku AndroidManifest.xml .....	50
1.7.	Odwzorowanie aplikacji na procesy .....	52
1.8.	Tworzenie aplikacji dla systemu Android .....	53
1.9.	Podsumowanie .....	57
<b>2.</b>	<b>Środowisko programowania dla systemu Android</b> .....	<b>59</b>
2.1.	Wprowadzenie do Android SDK .....	60
2.1.1.	Podstawowe pakiety systemu Android .....	61
2.1.2.	Pakiety opcjonalne .....	62
2.2.	Przegląd środowiska programowania .....	62
2.2.1.	Perspektywa Java .....	64
2.2.2.	Perspektywa DDMS .....	65
2.2.3.	Narzędzia wiersza poleceń .....	68
2.3.	Budowanie aplikacji dla systemu Android w Eclipse .....	71
2.3.1.	Kreator projektu aplikacji Android .....	72
2.3.2.	Kod źródłowy przykładowej aplikacji Android .....	72
2.3.3.	Pakowanie aplikacji .....	79
2.4.	Użycie emulatora systemu Android .....	80
2.4.1.	Konfiguracja środowiska emulowanego .....	81
2.4.2.	Testowanie aplikacji w emulatorze .....	85
2.5.	Debugowanie aplikacji .....	86
2.6.	Podsumowanie .....	88
	<b>Część II. Ćwiczenia z Android SDK</b> .....	<b>89</b>
<b>3.</b>	<b>Interfejs użytkownika</b> .....	<b>91</b>
3.1.	Tworzenie aktywności .....	93
3.1.1.	Tworzenie klasy Activity .....	94
3.1.2.	Przedstawiamy cykl życia aktywności .....	99
3.2.	Praca z widokami .....	103
3.2.1.	Przegląd wspólnych widoków .....	103
3.2.2.	Korzystanie z ListView .....	105
3.2.3.	Wielowątkowość z użyciem klas Handler i Message .....	109
3.2.4.	Tworzenie własnych widoków .....	111
3.2.5.	Przedstawiamy układy .....	113
3.2.6.	Obsługa fokusu .....	115
3.2.7.	Przechwytywanie zdarzeń .....	116
3.3.	Użycie zasobów .....	117
3.3.1.	Obsługiwane typy zasobów .....	117
3.3.2.	Odwolywanie się do zasobów w kodzie Java .....	118
3.3.3.	Definiowanie widoków i układów w zasobach XML .....	120
3.3.4.	Wartości zewnętrzne .....	122
3.3.5.	Tworzenie animacji .....	125
3.4.	Przedstawiamy plik AndroidManifest.xml .....	126
3.5.	Podsumowanie .....	128

<b>4.</b>	<b>Intencje i usługi</b>	<b>129</b>
4.1.	Obsługa aplikacji Wyszukiwarka restauracji z użyciem intencji	130
4.1.1.	Definiowanie intencji	131
4.1.2.	Wywołania jawne i niejawne	131
4.1.3.	Dodawanie łączy zewnętrznych do aplikacji Wyszukiwarka restauracji	132
4.1.4.	Wyszukiwanie celu dla intencji	135
4.1.5.	Wykorzystanie aktywności dostępnych w Androidzie	137
4.2.	Sprawdzanie pogody z użyciem własnych URI	138
4.2.1.	Oferowanie własnych URI	138
4.2.2.	Użycie niestandardowego URI	140
4.3.	Sprawdzanie pogody za pomocą obiektu BroadcastReceiver	142
4.3.1.	Rozgłaszanie intencji	142
4.3.2.	Tworzenie odbiornika	143
4.4.	Budowanie usługi prognozy pogody	144
4.5.	Komunikacja WeatherAlertService z innymi aplikacjami	148
4.5.1.	Język definicji interfejsu	148
4.5.2.	Binder oraz Parcelable	150
4.5.3.	Udostępnianie zdalnego interfejsu	151
4.5.4.	Łączenie z usługą	152
4.5.5.	Uruchamianie i dołączanie usługi	155
4.5.6.	Cykl życia usługi	156
4.6.	Podsumowanie	157
<b>5.</b>	<b>Zapisywanie i odczytywanie danych</b>	<b>159</b>
5.1.	Użycie właściwości	160
5.1.1.	Wykorzystanie obiektu SharedPreferences	160
5.1.2.	Uprawnienia dostępu do właściwości	163
5.2.	Użycie systemu plików	166
5.2.1.	Tworzenie plików	166
5.2.2.	Odczyt z plików	167
5.2.3.	Pliki jako surowe zasoby	168
5.2.4.	Zasoby plików XML	169
5.2.5.	Zapis na karcie SD	171
5.3.	Zapisywanie danych w bazie danych	174
5.3.1.	Budowanie i wykorzystanie bazy danych	175
5.3.2.	Wykorzystanie programu sqlite3	179
5.4.	Użycie klas ContentProvider	180
5.4.1.	Użycie istniejącej klasy ContentProvider	181
5.4.2.	Tworzenie dostawcy treści	182
5.5.	Podsumowanie	188
<b>6.</b>	<b>Sieci oraz usługi sieciowe</b>	<b>189</b>
6.1.	Przegląd zagadnień sieciowych	191
6.1.1.	Podstawy sieci	191
6.1.2.	Serwery i klienci	194
6.2.	Sprawdzanie stanu sieci	195
6.3.	Komunikacja poprzez gniazdo serwera	196
6.4.	Wykorzystanie HTTP	199
6.4.1.	Proste żądania HTTP i java.net	199
6.4.2.	Zaawansowana obsługa HTTP za pomocą HttpClient	201
6.4.3.	Tworzenie klasy pomocniczej dla wywołań HTTP i HTTPS	203

6.5.	Usługi sieciowe .....	209
6.5.1.	POX — połączenie HTTP i XML .....	209
6.5.2.	REST .....	211
6.5.3.	SOAP czy nie SOAP — oto jest pytanie .....	215
6.6.	Podsumowanie .....	216
<b>7.</b>	<b>Telefonia</b> .....	<b>217</b>
7.1.	Przegląd podstawowych zagadnień związanych z telefonią .....	218
7.1.1.	Poznajemy GSM .....	219
7.1.2.	Poznajemy CDMA .....	220
7.2.	Dostęp do danych telefonii .....	221
7.2.1.	Odczyt właściwości telefonu .....	222
7.2.2.	Pozyskiwanie informacji o stanie telefonu .....	224
7.3.	Interakcja z telefonem .....	226
7.3.1.	Użycie intencji do nawiązywania połączeń .....	226
7.3.2.	Użycie narzędzi związanych z numerami telefonicznymi .....	227
7.3.3.	Przechwytywanie połączeń wychodzących .....	229
7.4.	Obsługa wiadomości — SMS .....	230
7.4.1.	Wysyłanie wiadomości SMS .....	231
7.4.2.	Odbieranie wiadomości SMS .....	233
7.5.	Podsumowanie .....	235
<b>8.</b>	<b>Powiadomienia i alarmy</b> .....	<b>237</b>
8.1.	Korzystanie z Toast .....	238
8.1.1.	Tworzenie aplikacji SMS korzystającej z klasy Toast .....	238
8.1.2.	Odbieranie wiadomości SMS .....	239
8.2.	Wprowadzenie do powiadomień .....	242
8.2.1.	Klasa Notification .....	243
8.2.2.	Powiadamianie użytkownika o wiadomości SMS .....	244
8.3.	Wprowadzenie do alarmów .....	247
8.3.1.	Przykład użycia alarmu .....	248
8.3.2.	Użycie powiadomień z alarmami .....	252
8.4.	Podsumowanie .....	253
<b>9.</b>	<b>Grafika i animacja</b> .....	<b>255</b>
9.1.	Rysowanie grafiki w systemie Android .....	256
9.1.1.	Rysowanie przy użyciu XML .....	257
9.1.2.	Przegląd figur rysowanych za pomocą XML .....	259
9.2.	Tworzenie animacji za pomocą API graficznego .....	261
9.2.1.	Animacja poklatkowa w Androidzie .....	261
9.2.2.	Programowe tworzenie animacji .....	263
9.3.	Wprowadzenie do OpenGL dla systemów wbudowanych .....	267
9.3.1.	Tworzenie kontekstu OpenGL .....	268
9.3.2.	Rysowanie prostokąta za pomocą OpenGL ES .....	272
9.3.3.	Tworzenie trójwymiarowych kształtów i powierzchni za pomocą OpenGL ES .....	275
9.4.	Podsumowanie .....	279
<b>10.</b>	<b>Multimedia</b> .....	<b>281</b>
10.1.	Wprowadzenie do multimediiów oraz OpenCORE .....	282
10.2.	Odtwarzanie dźwięków .....	283
10.3.	Odtwarzanie wideo .....	285

10.4. Przechwytywanie mediów .....	287
10.4.1. Obsługa kamery .....	287
10.4.2. Zapisywanie dźwięku .....	292
10.5. Zapisywanie wideo .....	295
10.6. Podsumowanie .....	301
<b>11. Lokalizacja</b> .....	<b>303</b>
11.1. Symulowanie położenia w emulatorze .....	305
11.1.1. Wysyłanie współrzędnych z perspektywy DDMS .....	305
11.1.2. Format GPS Exchange .....	307
11.1.3. Keyhole Markup Language z Google Earth .....	309
11.2. Użycie klas <code>LocationManager</code> i <code>LocationProvider</code> .....	311
11.2.1. Dostęp do danych lokalizacji za pomocą <code>LocationManager</code> .....	312
11.2.2. Użycie klasy <code>LocationProvider</code> .....	314
11.2.3. Odbieranie informacji o lokalizacji za pomocą klasy <code>LocationListener</code> .....	315
11.3. Korzystanie z map .....	318
11.3.1. Dziedziczenie po <code>MapActivity</code> .....	318
11.3.2. Użycie <code>MapView</code> .....	319
11.3.3. Umieszczanie danych na mapie za pomocą <code>Overlay</code> .....	322
11.4. Zamiana miejsc na adresy za pomocą klasy <code>Geocoder</code> .....	325
11.5. Podsumowanie .....	327
<b>Część III. Aplikacje dla systemu Android</b> .....	<b>329</b>
<b>12. Aplikacja wspomagająca serwisantów</b> .....	<b>331</b>
12.1. Projektowanie rzeczywistej aplikacji Android .....	333
12.1.1. Podstawowe wymagania aplikacji .....	333
12.1.2. Zarządzanie danymi .....	334
12.1.3. Architektura aplikacji i integracji .....	335
12.2. Określanie przebiegów w aplikacji .....	336
12.2.1. Określanie procesów w aplikacji .....	337
12.2.2. Lista plików źródłowych .....	338
12.2.3. Plik <code>AndroidManifest.xml</code> aplikacji serwisanta mobilnego .....	340
12.3. Kod źródłowy aplikacji .....	341
12.3.1. Aktywność <code>Splash</code> .....	341
12.3.2. Właściwości używane przez aktywność <code>FieldService</code> .....	343
12.3.3. Implementacja aktywności <code>FieldService</code> .....	344
12.3.4. Ustawienia .....	346
12.3.5. Zarządzanie danymi zleceń .....	348
12.4. Kod źródłowy dla zarządzania zleceniami .....	355
12.4.1. Aktywność <code>RefreshJobs</code> .....	355
12.4.2. Zarządzanie zleceniami — aktywność <code>ManageJobs</code> .....	359
12.4.3. Obsługa zlecenia w aktywności <code>ShowJob</code> .....	362
12.4.4. Przechwytywanie podpisu w aktywności <code>CloseJob</code> .....	366
12.5. Kod serwera .....	372
12.5.1. Interfejs użytkownika dyspozytora .....	372
12.5.2. Baza danych .....	373
12.5.3. Kod PHP aplikacji dyspozytora .....	374
12.5.4. Kod PHP do integracji z aplikacją mobilną .....	375
12.6. Podsumowanie .....	376

<b>13. Budowanie aplikacji Android w języku C</b>	<b>377</b>
13.1. Budowanie aplikacji Android bez SDK	378
13.1.1. Kompilator i linker języka C	379
13.1.2. Budowanie aplikacji Witaj, świecie	380
13.1.3. Instalowanie i uruchamianie aplikacji	381
13.1.4. Skrypt budujący aplikację w języku C	383
13.2. Rozwiązywanie problemu z łączeniem dynamicznym	384
13.2.1. Biblioteki systemu Android	384
13.2.2. Budowanie aplikacji łączonej dynamicznie	386
13.2.3. exit() oraz return()	389
13.2.4. Kod uruchamiający	390
13.3. Która godzina? Serwer DayTime	392
13.3.1. Aplikacja serwera DayTime	392
13.3.2. Plik daytime.c	393
13.3.3. Baza danych SQLite	395
13.3.4. Budowanie i uruchamianie serwera DayTime	397
13.4. Klient DayTime	399
13.4.1. Aktywność	399
13.4.2. Klient DayTime	401
13.4.3. Testowanie klienta DayTime	402
13.5. Podsumowanie	402
<b>Część IV. Dojrzewająca platforma</b>	<b>405</b>
<b>14. Bluetooth i sensory</b>	<b>407</b>
14.1. Przegląd możliwości Bluetooth w systemie Android	409
14.1.1. Zastępowanie kabli	409
14.1.2. Rola podstawowa i podrzędna oraz gniazda	410
14.1.3. Urządzenia zaufane	411
14.1.4. Podłączanie się do zdalnego urządzenia	413
14.1.5. Przechwytywanie zdarzeń Bluetooth	414
14.1.6. Uprawnienia Bluetooth	416
14.2. Interakcja z obiektem SensorManager	416
14.2.1. Typy czujników	417
14.2.2. Odczyt wartości czujnika	418
14.2.3. Włączanie i wyłączanie czujników	419
14.3. Budowanie aplikacji SenseBot	420
14.3.1. Interfejs użytkownika	420
14.3.2. Interpretowanie wartości czujnika	423
14.3.3. Jazda robotem	424
14.3.4. Komunikacja z robotem	425
14.4. Podsumowanie	426
<b>15. Integracja</b>	<b>429</b>
15.1. Poznajemy model kontaktów Androida	430
15.1.1. Wybór otwartych rekordów	431
15.1.2. Obsługa wielu kont	433
15.1.3. Unifikowanie widoku lokalnego z różnych zdalnych magazynów	434
15.1.4. Wspólny plac zabaw	436
15.2. Zaczynamy korzystać z LinkedIn	436



<b>15.3. Zarządzanie kontaktami</b>	<b>438</b>
15.3.1. Wykorzystanie wbudowanej aplikacji kontaktów	438
15.3.2. Żądanie wykonania operacji z naszej aplikacji	441
15.3.3. Bezpośredni odczyt i modyfikowanie bazy danych kontaktów	443
15.3.4. Dodawanie kontaktów	444
<b>15.4. Łączenie wszystkiego</b>	<b>446</b>
15.4.1. Marzenia o synchronizacji	446
15.4.2. Definiowanie kont	447
15.4.3. Usługa AccountManager	449
<b>15.5. Tworzenie konta LinkedIn</b>	<b>450</b>
15.5.1. Brak obsługi urządzeń mobilnych	450
15.5.2. Uwierzytelnianie w LinkedIn	450
<b>15.6. Synchronizacja w tle z użyciem SyncAdapter</b>	<b>458</b>
15.6.1. Cykl życia synchronizacji	458
15.6.2. Synchronizowanie danych LinkedIn	458
<b>15.7. Kończymy — LinkedIn w akcji</b>	<b>461</b>
15.7.1. Kończenie projektu LinkedIn	461
15.7.2. Rozwiązywanie problemów	462
15.7.3. Kontynuacja	463
<b>15.8. Podsumowanie</b>	<b>464</b>
<b>16. Tworzenie stron WWW dla systemu Android</b>	<b>465</b>
<b>16.1. Czym jest programowanie WWW dla Androida</b>	<b>466</b>
16.1.1. Wprowadzenie do WebKit	467
16.1.2. Analiza opcji architektury	467
<b>16.2. Optymalizacja aplikacji WWW dla Androida</b>	<b>468</b>
16.2.1. Projektowanie z zachowaniem mobilności stron	469
16.2.2. Dodawanie znacznika viewport	470
16.2.3. Selektywne ładowanie treści	472
16.2.4. Analiza parametru user agent	473
16.2.5. Zapytanie media	474
16.2.6. Aplikacja tylko dla urządzeń mobilnych	476
<b>16.3. Przechowywanie danych w przeglądarce</b>	<b>477</b>
16.3.1. Konfiguracja	477
16.3.2. Analiza kodu	479
16.3.3. Interfejs użytkownika	480
16.3.4. Otwieranie bazy danych	481
16.3.5. Analiza funkcji transaction()	482
16.3.6. Wstawianie i usuwanie wierszy	484
16.3.7. Testowanie aplikacji za pomocą narzędzi WebKit	485
<b>16.4. Budowanie aplikacji hybrydowej</b>	<b>486</b>
16.4.1. Poznajemy kontrolkę przeglądarki	487
16.4.2. Konfigurowanie kontrolki	487
16.4.3. Implementowanie obiektu obsługi JavaScript	489
16.4.4. Użycie kodu z JavaScript	491
16.4.5. Spotkanie z JavaScript	492
16.4.6. Przede wszystkim bezpieczeństwo	494
16.4.7. Implementacja WebViewClient	496
16.4.8. Rozszerzanie przeglądarki	496
16.4.9. Wykrywanie zdarzeń nawigacyjnych	497
16.4.10. Implementacja klasy WebChromeClient	500
<b>16.5. Podsumowanie</b>	<b>501</b>

<b>17. Widżety aplikacji</b>	<b>503</b>
17.1. Wprowadzenie do widżetów aplikacji	504
17.1.1. Co to jest widżet aplikacji?	504
17.1.2. Strategie instalowania widżetów	507
17.2. Wprowadzenie do aplikacji SiteMonitor	509
17.2.1. Korzyści z aplikacji SiteMonitor	509
17.2.2. Interfejs użytkownika	509
17.3. Architektura aplikacji SiteMonitor	513
17.3.1. Schemat projektu aplikacji	513
17.3.2. Plik po pliku	515
17.4. Obsługa danych w widżetach	515
17.5. Implementacja klasy AppWidgetProvider	519
17.5.1. Katalog metod klasy AppWidgetProvider	520
17.5.2. Implementowanie klasy SiteMonitorWidgetImpl	520
17.5.3. Obsługa widżetów zombie	523
17.6. Wyświetlanie widżetów za pomocą RemoteViews	524
17.6.1. Korzystanie z RemoteViews	524
17.6.2. Metoda UpdateOneWidget()	525
17.7. Konfigurowanie instancji widżetu	527
17.7.1. Metadane widżetu	527
17.7.2. Operacje na danych intencji	528
17.7.3. Potwierdzenie utworzenia widżetu	529
17.8. Aktualizacja widżetu	531
17.8.1. Porównanie usługi z alarmem	531
17.8.2. Uruchamianie aktualizacji	532
17.8.3. Aktualizowanie widżetów	534
17.9. Łączenie wszystkich elementów w pliku AndroidManifest.xml	538
17.10. Podsumowanie	539
<b>18. Lokalizowanie aplikacji</b>	<b>541</b>
18.1. Potrzeba lokalizowania	542
18.2. Ustawienia regionalne	543
18.3. Strategie lokalizowania aplikacji	545
18.3.1. Identyfikowanie docelowych języków i danych	545
18.3.2. Identyfikowanie tekstów i zarządzanie nimi	546
18.3.3. Rysunki i układy	548
18.3.4. Data, czas, liczby i waluty	549
18.3.5. Praca z zespołem tłumaczy	550
18.4. Wykorzystanie możliwości zasobów Androida	550
18.4.1. Więcej niż ustawienia regionalne	551
18.4.2. Przypisywanie tekstów z zasobów	551
18.5. Lokalizowanie kodu Java	553
18.6. Formatowanie lokalizowanych napisów	554
18.7. Problemy z lokalizowaniem	556
18.8. Podsumowanie	557
<b>19. Android Native Development Kit</b>	<b>559</b>
19.1. Wprowadzenie do NDK	560
19.1.1. Zastosowania NDK	561
19.1.2. Przegląd NDK	561

<b>19.2. Budowanie aplikacji za pomocą NDK .....</b>	<b>563</b>
19.2.1. Demonstracja gotowej aplikacji .....	563
19.2.2. Struktura projektu .....	565
<b>19.3. Budowanie biblioteki JNI .....</b>	<b>565</b>
19.3.1. Poznajemy JNI .....	566
19.3.2. Implementacja biblioteki .....	567
19.3.3. Kompilowanie biblioteki JNI .....	571
<b>19.4. Budowanie interfejsu użytkownika .....</b>	<b>572</b>
19.4.1. Układ interfejsu użytkownika .....	572
19.4.2. Wykonanie zdjęcia .....	574
19.4.3. Wyszukiwanie krawędzi .....	576
<b>19.5. Integracja NDK w Eclipse .....</b>	<b>577</b>
<b>19.6. Podsumowanie .....</b>	<b>578</b>

## **Część V. Dodatki** **581**

<b>A. Instalowanie Android SDK</b> .....	<b>583</b>
A.1. Wymagania środowiska programistycznego .....	584
A.2. Pobieranie i instalowanie Eclipse .....	584
A.3. Pobieranie i instalowanie Android SDK .....	587
A.4. Korzystanie z SDK oraz AVD Manager .....	587
A.5. Pobieranie i instalowanie wtyczki Eclipse .....	590
A.6. Konfigurowanie wtyczki Eclipse .....	592
<b>B. Publikowanie aplikacji</b> .....	<b>595</b>
<b>B.1. Przygotowanie aplikacji do dystrybucji</b> .....	<b>596</b>
B.1.1. Rejestrowanie.....	596
B.1.2. Powiadomienia debugowania.....	596
B.1.3. Przykładowe dane .....	596
B.1.4. Plik AndroidManifest.xml .....	597
B.1.5. Licencja użytkownika .....	597
B.1.6. Testowanie .....	598
B.1.7. Operacje końcowe .....	598
<b>B.2. Podpisywanie cyfrowe aplikacji</b> .....	<b>599</b>
B.2.1. Magazyny kluczy .....	599
B.2.2. keytool .....	599
B.2.3. jarsigner .....	600
<b>B.3. Publikowanie w Android Market</b> .....	<b>602</b>
B.3.1. Zasady Android Market .....	602
B.3.2. Umieszczanie aplikacji w Android Market.....	603
B.3.3. Android Market — właściwe rozwiązanie.....	603
<b>B.4. Inne sposoby dystrybucji</b> .....	<b>604</b>
<b>B.5. Podsumowanie Android Debug Bridge</b> .....	<b>606</b>
<b>Skorowidz</b> .....	<b>609</b>

# 14

## *Bluetooth i sensory*

### **W tym rozdziale:**

- ◆ Dołączanie Bluetooth do urządzenia
- ◆ Interakcja z obiektem SensorManager
- ◆ Budowanie i uruchamianie aplikacji SenseBot

Większość materiału przedstawionego w tej książce dotyczy wykorzystania różnych możliwości Android SDK. Teraz jednak pokażemy, jak skorzystać z możliwości sprzętowych urządzeń Android. Zapoznamy się tu z podłączaniem urządzenia Android do zdalnych urządzeń za pomocą połączenia bezprzewodowego Bluetooth oraz odczytem i interpretacją wartości ze sprzętowego czujnika orientacji urządzenia. W rozdziale tym te dwa tematy związane ze sprzętem połączyliśmy w jednym przykładowym programie, który pozwala sterować robotem zbudowanym przy użyciu popularnego zestawu LEGO Mindstorms NXT. Robot Mindstorm NXT obsługuje protokół komunikacyjny znany pod nazwą „Direct Commands<sup>1</sup>”, co pozwala na sterowanie nim z użyciem urządzenia zdalnego. Jest to rozdział, w którym konieczne jest użycie fizycznego urządzenia z zainstalowanym systemem Android w wersji 2 lub nowszej — sam symulator nie jest wystarczający do sprawdzania działania Bluetooth oraz sensorów.

Kod dołączony do tego rozdziału tworzy aplikację o nazwie SenseBot. Jest to średnio skomplikowany przykład użycia systemu Android do manipulowania zewnętrznymi obiektami. Czujniki orientacji urządzenia Android pozwalają użytkownikowi „jechać” robotem, przechylając telefon w określonym kierunku, w sposób podobny do używania Nintendo Wii lub innego zaawansowanego systemu gier. Przechylamy telefon do przodu, a robot jedzie do przodu. Przechylamy telefon do tyłu, a robot zmienia kierunek. Przechylanie w lewo lub w prawo powoduje skręt robota w odpowiednim kierunku. Po zinterpretowaniu wartości każdego czujnika ruchu aplikacja SenseBot wysyła polecenia do robota poprzez Bluetooth, powodując odpowiednie fizyczne działanie. LEGO NXT ma wbudowany zestaw poleceń pozwalających na operacje niskiego poziomu, takie jak bezpośrednie sterowanie silnikami. Ruchy urządzenia Android są interpretowane, konwertowane na polecenia i przesyłane za pomocą Bluetooth do robota.

Oprócz podstaw komunikacji Bluetooth oraz zarządzania sensorami kod demonstruje użycie dynamicznie tworzonych obiektów `BroadcastReceiver`, które obsługują zdarzenia związane z połączeniem Bluetooth.

Temat komunikacji Bluetooth jest znacznie szerszy i nie mamy możliwości opisać go w całości w jednym rozdziale. Na platformie Android dostępnych jest co najmniej sześć różnych sensorów, a w tym rozdziale demonstrujemy użycie tylko jednego. Jeżeli szukasz dokładnego opisu tych dwóch tematów, zachęcamy do odszukania dokumentacji w sieci lub być może innej książki na ten temat. Celem tego rozdziału jest przedstawienie funkcji Bluetooth oraz sensorów na platformie Android w kontekście działającej (i zabawnej) aplikacji. Jeżeli masz dostęp do robota LEGO Mindstorms NXT i zbudujesz tę aplikację, obiecujemy, że na długi czas nie będziesz się mógł oderwać od „jeżdżenia” robotem za pomocą telefonu. Jedną z wersji tej aplikacji jest dostępna do pobrania z Android Market.

---

<sup>1</sup> Więcej informacji na temat Direct Commands dla Lego Mindstorm można znaleźć pod adresem <http://mindstorms.lego.com/en-us/support/files/default.aspx>.

## 14.1. Przegląd możliwości Bluetooth w systemie Android

---

Pierwszym urządzeniem, jakie przychodzi na myśl, gdy wspomina się *Bluetooth*, jest bezprzewodowy zestaw słuchawkowy. W wielu krajach te cudeńka technologii bezprzewodowej są wymagane przez prawo w przypadku korzystania z telefonu w czasie prowadzenia samochodu. W rzeczywistości zestawy słuchawkowe to tylko jedno z wielu zastosowań technologii Bluetooth.

Bluetooth to technologia komunikacji bezprzewodowej podobna do Wi-Fi, ale ograniczona do komunikacji o niewielkim zasięgu, około 10 metrów. Oprócz zastosowania w bezprzewodowych słuchawkach i mikrofonie telefonu komórkowego Bluetooth pozwala również na realizację połączeń sieciowych między urządzeniami, wymianę obiektów, zastępowanie kabli oraz realizację zaawansowanych funkcji audiowizualnych.

Podobnie jak każdy inny standardowy protokół, tak i Bluetooth posiada własny „stos” protokołów, z których każdy implementuje osobne możliwości i funkcje protokołu. W rozdziale tym nie będziemy tracić czasu na prezentowanie tych warstw, ponieważ stos Bluetooth jest opisany w innych miejscach. Zamiast tego w rozdziale tym pokażemy możliwości nawiązania połączenia danych pomiędzy dwoma urządzeniami. Będziemy tu korzystać z „profilu” Bluetooth o nazwie RFCOMM<sup>2</sup>, będącego profilem zastępowania kabla.

W tym podrozdziale pokażemy, jak nawiązać połączenie pomiędzy Androidem a zdalnym urządzeniem, korzystając z pakietu `android.bluetooth`. Ponieważ platforma Android pozwala tylko na połączenia szyfrowane, dwa urządzenia komunikacyjne muszą być wcześniej sparowane, a później mogą łączyć się ze sobą bez konieczności podawania hasła. Aby wiedzieć, czy aplikacja jest połączona z urządzeniem Bluetooth, należy następnie zarejestrować dwa zdarzenia: `ACTION_ACL_CONNECTED` oraz `ACTION_ACL_DISCONNECTED`. Dodatkowo aplikacja Android musi posiadać uprawnienie `BLUETOOTH` zdefiniowane w pliku *AndroidManifest.xml*. Zaczynamy!

### 14.1.1. Zastępowanie kabli

Obecnie podłączanie się do internetu w celu sprawdzenia poczty lub przeglądania sieci WWW jest codzienną czynnością wielu użytkowników systemu Android. Przy użyciu naszego telefonu możemy połączyć się z komputerami znajdującymi się po drugiej stronie planety, ale jak komunikować się z czymś, co znajduje się w tym samym pokoju? W niezbyt odległej przeszłości programowaliśmy interfejsy pomiędzy komputerami połączonymi szeregowo kablem korzystającym z interfejsu RS232. Po kilku krótkich latach kable szeregowo RS232, zastąpione przez bardziej zaawansowane kable USB oraz przez profil Bluetooth Serial Port, stały się eksponatem muzealnym.

Tak jak USB może być używane przez wiele różnych aplikacji, tak i protokół bezprzewodowy Bluetooth może być wykorzystywany na wiele sposobów. W tym momencie interesuje nas możliwość użycia Bluetooth do zastąpienia kabla szeregowego za pomocą profilu Serial Port Profile (SPP), nazywanego czasami RFCOMM. RF pochodzi

---

<sup>2</sup> Więcej informacji na temat RFCOMM można znaleźć na stronie <http://www.bluetooth.com>.

od słów *radio frequency*, czyli częstotliwości radiowych. COMM pochodzi od portu komunikacyjnego, czyli antycznego już połączenia punkt-punkt, wykorzystującego protokół strumieniowy.

### 14.1.2. Rola podstawowa i podrzędna oraz gniazda

Protokół Bluetooth działa w sposób podobny do innych środowisk komunikacyjnych, w których urządzenie podstawowe inicjuje komunikację z jednym lub większą liczbą urządzeń podrzędnych. Android jest na tyle rozbudowany, że może być zarówno urządzeniem podstawowym, jak i podrzędnym w połączeniu Bluetooth.

Niezależnie od sposobu nawiązania połączenia — jako podstawowe lub drugorzędne urządzenie Bluetooth — aplikacja Android wymienia dane poprzez interfejs gniazd. To prawda — znany już mechanizm sieciowy korzystający z gniazd oraz skojarzonych z nimi strumieni wejściowych i wyjściowych jest również stosowany przy połączeniach Bluetooth. Jeżeli więc przejdziemy przez etap łączenia ze sobą dwóch urządzeń Bluetooth w celu utworzenia sesji komunikacyjnej, możemy mniej przejmować się szczegółami komunikacji i po prostu traktować urządzenie jako aplikację po drugiej stronie gniazda. Jest to podobne do relacji pomiędzy przeglądarką WWW i zdalnym serwerem, który wymienia dane poprzez gniazda TCP.

Aby użyć środowiska Bluetooth w urządzeniu Android, musimy skorzystać z pakietu `android.bluetooth`, który został dodany w wersji 2.0. Choć większość urządzeń Android w wersji niższej niż 2.0 mogła korzystać z zestawów słuchawkowych Bluetooth, to dopiero od wersji 2.0 aplikacje Android mogły wykorzystywać komunikację Bluetooth w sposób przedstawiony w tym rozdziale. W tabeli 14.1 pokazane są główne klasy Java używane w aplikacjach Android korzystających z komunikacji Bluetooth.

**Tabela 14.1.** Klasy Bluetooth

Klasa	Komentarz
<code>BluetoothAdapter</code>	Klasa ta reprezentuje sprzęt lokalnego urządzenia Bluetooth w systemie Android i pozwala na tworzenie interfejsu. Wszystko zaczyna się od <code>BluetoothAdapter</code> .
<code>BluetoothClass</code>	Klasa <code>BluetoothClass</code> zapewnia wygodne metody dostępu do statycznych wartości związanych z komunikacją i działaniem Bluetooth.
<code>BluetoothDevice</code>	Każde urządzenie zdalne jest reprezentowane jako <code>BluetoothDevice</code> .
<code>BluetoothSocket</code>	Klasa <code>BluetoothSocket</code> jest używana przy wymianie danych. Urządzenie podstawowe inicjuje połączenie z gniazdem w urządzeniu podrzędnym, tworząc wcześniej <code>BluetoothSocket</code> . W rozdziale tym przedstawione są przykłady kodu demonstrujące tę technikę.
<code>BluetoothServerSocket</code>	Urządzenie podrzędne Bluetooth oczekuje połączeń z urządzeniem nadrzędnym za pomocą <code>BluetoothServerSocket</code> w taki sam sposób, jak serwer WWW oczekuje na połączenie przez gniazdo TCP z przeglądarki. Po zestawieniu połączenia tworzony jest obiekt <code>BluetoothSocket</code> zapewniający dalszą komunikację.

W rozdziale tym przedstawimy użycie klas `BluetoothAdapter`, `BluetoothDevice` oraz `BluetoothSocket`. W następnym punkcie pokażemy, w jaki sposób urządzenie Android podłącza się do innego urządzenia Bluetooth.

**Uwaga**

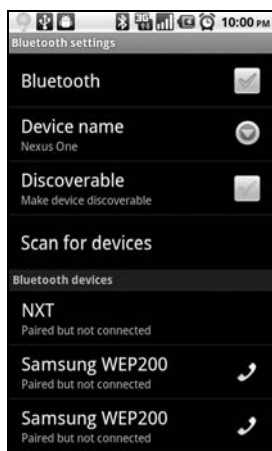
W przedstawionych tu przykładach urządzenie Android jest podstawowym, a kontroler LEGO Mindstorm NXT podrzędnym urządzeniem Bluetooth.

**14.1.3. Urządzenia zaufane**

Choć specyfikacja Bluetooth pozwala na korzystanie zarówno z połączeń szyfrowanych, jak i nieszyfrowanych, to jednak platforma Android pozwala wyłącznie na połączenia szyfrowane. W zasadzie oznacza to tylko, że dwa komunikujące się z sobą urządzenia muszą być wcześniej *sparowane* lub *związane*. Jest to nieco irytujący krok polegający na poinformowaniu każdego z urządzeń, że drugie jest urządzeniem zaufanym. Pomimo kłopotliwości tego procesu oraz faktu, że niemal wszystkie urządzenia Bluetooth na tej planecie korzystają z kodu zabezpieczeń 0000 lub 1234, jedynym zabezpieczeniem protokołu Bluetooth jest ten właśnie kod.

Urządzenia mogą być sparowane poprzez ich ekrany „ustawień” lub podczas nawiązywania pierwszego połączenia. W tym punkcie przedstawimy sposób parowania urządzenia Android<sup>3</sup> z modułem kontrolera robota LEGO.

Na rysunku 14.1 pokazana jest część ekranu ustawień Bluetooth z mojego telefonu Nexus One z systemem Android 2.2.



**Rysunek 14.1.**  
Ekran ustawień  
Bluetooth

Z tego ekranu możemy uzyskać następujące informacje:

- ◆ Bluetooth jest włączony.
- ◆ Urządzenie ma nazwę *Nexus One*.
- ◆ Urządzenie to nie jest obecnie wykrywalne. Oznacza to, że inne urządzenia Bluetooth nie widzą tego telefonu w czasie „skanowania”. Praktycznie rzecz biorąc, oznacza to, że telefon ignoruje pakiety wykrywania. Znajduje się tu również

<sup>3</sup> Aby dowiedzieć się więcej na temat typów używanych układów w sprzęcie Android, warto skorzystać z forum Talk Android, dostępnego pod adresem <http://www.talkandroid.com/android-forums/android-hardware/>.



przycisk pozwalający na zainicjowanie wykrywania znajdujących się w pobliżu urządzeń.

- ◆ Aby rozpocząć skanowanie znajdujących się w zasięgu urządzeń Bluetooth, należy kliknąć przycisk *Wyszukaj urządzenia*.
- ◆ W telefonie były również wcześniej sparowane inne urządzenia, które nie są obecnie połączone:
  - NXT — robot LEGO.
  - Dwa urządzenia bezprzewodowe Samsung. To nie pomyłka — mamy tu dwa osobne urządzenia sparowane z telefonem. (Autor w ten sposób „rozwiązał” problem częstego gubienia zestawów słuchawkowych przez kupienie kilku sztuk poprzez eBay, stąd kilka sparowanych urządzeń).

Długie kliknięcie jednego z tych urządzeń Bluetooth na liście pozwala otworzyć ekran opcji z dodatkowymi operacjami, zależnymi od urządzenia. Na przykład wybranie jednego z urządzeń Samsung powoduje otwarcie okna pokazanego na rysunku 14.2.



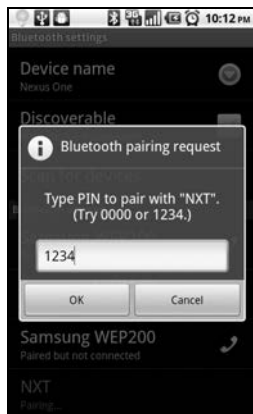
**Rysunek 14.2.**  
Opcje  
sparowanego  
urządzenia

Aby sparować urządzenie, należy najpierw przeprowadzić skanowanie. Po dodaniu urządzenia do listy możemy je wybrać i zainicjować parowanie. Na rysunku 14.3 pokazany jest kontroler robota LEGO, na którym pokazane jest żądanie PIN w czasie parowania.



**Rysunek 14.3.**  
Kontroler LEGO  
żądający podania  
kodu PIN

Podany PIN jest porównywany z kodem wprowadzonym w telefonie, co jest pokazane na rysunku 14.4.



**Rysunek 14.4.**  
Parowanie  
z robotem LEGO

W tym momencie nasz telefon i kontroler robota LEGO są sparowane. Od tego momentu będziemy w stanie podłączać się do tego urządzenia bez żądania podania kodu PIN.

#### 14.1.4. Podłączanie się do zdalnego urządzenia

Podłączanie się do sparowanego urządzenia wymaga wykonania dwóch kroków:

- ◆ Pobrania listy sparowanych urządzeń za pomocą stosu sprzętowo-programowego Bluetooth.
- ◆ Zainicjowania połączenia RFCOMM z urządzeniem docelowym. Na poniższym listingu pokazany jest sposób nawiązywania połączenia RFCOMM lub Serial Port Profile pomiędzy dwoma sparowanymi urządzeniami.

**Listing 14.1.** Inicjowanie połączenia z urządzeniem Bluetooth

```
public void findRobot(View v)
{
    try
    {
        btInterface = BluetoothAdapter.getDefaultAdapter(); ← 1
        pairedDevices = btInterface.getBondedDevices(); ← 2
        Iterator<BluetoothDevice> it = pairedDevices.iterator();
        while (it.hasNext()) ← 3
        {
            BluetoothDevice bd = it.next();
            if (bd.getName().equalsIgnoreCase(ROBOTNAME)) { ← 4
                connectToRobot(bd); ← 5
                return;
            }
        }
    }
    catch (Exception e)
    {
        Log.e(tag, "Błąd w findRobot() " + e.getMessage()); ← 6
    }
}
```

```

private void connectToRobot(BluetoothDevice bd) ← ❸
{
    try
    {
        socket = bd.createRfcommSocketToServiceRecord
(UUID.fromString("00001101-0000-1000-8000-00805F9B34FB")); ← ❹
        socket.connect(); ← ❺
    }
    catch (Exception e)
    {
        Log.e(tag, "Błąd komunikacji ze zdalnym urządzeniem [" + e.getMessage() + "]);
    }
}

```

Wszystkie operacje związane z Bluetooth<sup>4</sup> zaczynają się od `BluetoothAdapter` ❶. Przy użyciu referencji do adaptera można uzyskać listę sparowanych urządzeń ❷. Następnie przeglądamy listę ❸, szukając nazwy urządzenia ❹ odpowiadającej nazwie robota. Nazwa ta może być wpisana na stałe, jak jest to zrealizowane w przykładowej aplikacji, wprowadzana przez użytkownika w czasie uruchamiania lub nawet wybierana z bardziej zaawansowanej listy urządzeń. Niezależnie od metody naszym celem jest zidentyfikowanie obiektu `BluetoothDevice`, a następnie zainicjowanie połączenia, jak jest to pokazane w wywołaniu funkcji o nazwie `connectToRobot()` ❺. Dobrą praktyką jest przechwytywanie wyjątków ❻, szczególnie jeżeli korzystamy ze zdalnego urządzenia fizycznego, które może wyjść z zasięgu lub może mieć problemy z zasilaniem. Aby podłączyć się do zdalnego urządzenia za pomocą profilu `Serial Port Profile`, należy użyć metody `createRfComSocketToServiceRecord()` z klasy `BluetoothDevice`. Ciąg UUID pokazany w kodzie jest identyfikatorem profilu `Serial Port Profile` ❼. Gdy mamy dostępny obiekt `BluetoothSocket`, można wywołać metodę `connect()` ❸.

W tym momencie mamy odnalezione interesujące nas urządzenie i próbujemy wysłać żądanie połączenia. Czy to się udało? Jak możemy to sprawdzić? Można poczynić założenie na temat stanu urządzenia i ewentualnie czekać na błąd. Nie jest to jednak najlepsze podejście. Musi istnieć inna metoda — faktycznie jest taka metoda, ale wymaga wykorzystania obiektów `Intent`.

### 14.1.5. Przechwytywanie zdarzeń Bluetooth

Aby sprawdzić, czy aplikacja jest prawidłowo podłączona do `BluetoothDevice`, należy zarejestrować dwa zdarzenia związane z Bluetooth: `ACTION_ACL_CONNECTED` oraz `ACTION_ACL_DISCONNECTED`. Gdy wystąpią te dwa zdarzenia, możemy być pewni, że mamy prawidłowe połączenie lub je utraciliśmy. W jaki sposób można użyć tych zdarzeń w połączeniu z utworzonym wcześniej gniazdem? Na poniższym listingu pokazana jest technika polegająca na tworzeniu obiektów `BroadcastReceiver` bezpośrednio w aktywności i rejestrowaniu w nich interesujących nas zdarzeń.

<sup>4</sup> Więcej informacji można znaleźć w dokumentacji Google na temat Bluetooth oraz Android dostępnej pod adresem <http://developer.android.com/guide/topics/wireless/bluetooth.html>.

**Listing 14.2.** Monitorowanie połączenia Bluetooth

```

private BroadcastReceiver btMonitor = null; ← ❶
private void setupBTMonitor() { ← ❷
    btMonitor = new BroadcastReceiver() { ← ❸
        @Override
        public void onReceive(Context context, Intent intent) { ← ❹
            if (intent.getAction().equals(
                "android.bluetooth.device.action.ACL_CONNECTED")) {
                handleConnected(); ← ❺
            }
            if (intent.getAction().equals(
                "android.bluetooth.device.action.ACL_DISCONNECTED")) {
                handleDisconnected(); ← ❻
            }
        }
    };
}

```

Aby monitorować specyficzne rozgłaszane zdarzenia, należy użyć obiektu `BroadcastReceiver`. Zwykle realizuje się to przy użyciu osobnej klasy, ale aplikacja ta wymaga bliższej integracji z interfejsem użytkownika, więc zastosowaliśmy alternatywne podejście. Zwykle obiekty `BroadcastReceiver` są definiowane w pliku *AndroidManifest.xml*, ale w tym przypadku chcemy otrzymywać powiadomienia w określonych przypadkach. W kodzie tym zdefiniowany jest obiekt `BroadcastReceiver` o nazwie `btMonitor` ❶. W metodzie `onCreate()` wywoływana jest metoda `setupBTMonitor()` ❷, w której tworzony jest obiekt `BroadcastReceiver` ❸ zawierający implementację metody `onReceive()` ❹. Za każdym razem, gdy dla tego obiektu `BroadcastReceiver` dostępny jest rozgłaszany obiekt `Intent`, wywoływana jest metoda `onReceive()`. W implementacji tej obsługujemy akcje podłączenia i rozłączenia urządzenia Bluetooth. Gdy urządzenie jest podłączone, wywoływana jest metoda `handleConnected()` ❺. Podobnie gdy zdalne urządzenie zostanie rozłączone, wywoływana jest metoda `handleDisconnected()` ❻, której zadaniem jest wykonanie operacji porządkowych.

Gdy urządzenie jest podłączone, musimy wykonać kilka operacji przygotowawczych, takich jak skonfigurowanie strumienia wejściowego oraz wyjściowego dla gniazda. Na następnym listingu przedstawiona jest skrócona wersja metody `handleConnected()` zawierająca kod pozwalający obsługiwać Bluetooth.

**Listing 14.3.** Metoda `handleConnected`

```

private void handleConnected() {
    try {
        is =
        socket.getInputStream();
        os = socket.getOutputStream(); ← ❶
        bConnected = true; ← ❷
        btnConnect.setVisibility(View.GONE);
        btnDisconnect.setVisibility(View.VISIBLE); ← ❸
    } catch (Exception e) {
        is = null; ← ❹
        os = null;
        disconnectFromRobot(null); ← ❺
    }
}

```

Gdy wywołana zostaje metoda `handleConnection()`, zestawione jest prawidłowe połączenie gniazd Bluetooth, więc musimy tylko utworzyć strumienie wejścia i wyjścia ❶. Po utworzeniu tych strumieni może rozpocząć się komunikacja pomiędzy urządzeniem Android a robotem LEGO. Jak się okaże w dalszej części tego rozdziału, chcemy przetwarzać zdarzenia czujnika wyłącznie przy działającym połączeniu z robotem, więc ustawiamy znacznik ❷ informujący aplikację o stanie połączenia. W procedurze tej przełączamy również widoczność dwóch przycisków ❸ — jednego używanego do łączenia z robotem oraz drugiego pozwalającego zakończyć połączenie. W przypadku wystąpienia błędu w czasie wykonywania tych operacji musimy zamknąć strumienie ❹ i zainicjować żądanie rozłączenia ❺.

Kod pozwalający na rozłączenie gniazda jest bardzo prosty:

```
socket.close();
```

Aby można było wykonać większość operacji Bluetooth w systemie Android, nie należy zapominać o jeszcze jednym ważnym elemencie: uprawnieniach!

### 14.1.6. Uprawnienia Bluetooth

Korzystanie ze sparowanych urządzeń to nie jedyna czynność, do której wykonania konieczne są uprawnienia. Aby korzystać z Bluetooth API, aplikacja Android musi posiadać uprawnienie `BLUETOOTH` w pliku *AndroidManifest.xml*:

```
<uses-permission android:name="android.permission.BLUETOOTH">
</uses-permission>
```

Część kodu odpowiedzialnego za komunikację Bluetooth jest przedstawiona w trzecim podrozdziale, w którym przedstawiamy dokładniej kod aplikacji *SenseBot*. Zanim zajmiemy się kodowaniem i uruchamianiem aplikacji robota, przyjrzyjmy się klasie `SensorManager`, dzięki której będziemy mogli wykorzystać czujniki Androida do sterowania robotem.

## 14.2. Interakcja z obiektem `SensorManager`

---

Android udostępnia odczyty z czujników poprzez klasę o nazwie `SensorManager`. Klasa `SensorManager` jest podobna do `BluetoothAdapter`, ponieważ wszystkie związane z nią aktywności intensywnie korzystają z `SensorManager`. Klasa `SensorManager` wchodzi w skład pakietu `android.hardware`. W tym podrozdziale pokażemy, jak odczytywać informacje z sensora orientacji, co będzie nam potrzebne w aplikacji *SenseBot*.

W tabeli 14.2 wymienione są główne klasy skojarzone z `SensorManager`.

Korzystanie z klasy `SensorManager` jest bardzo proste. Na początek należy odczytać referencję do obiektu tej klasy:

```
SensorManager sManager = (SensorManager)
    getSystemService(Context.SENSOR_SERVICE);
```

**Tabela 14.2.** Klasy związane z sensorami

Klasa	Komentarz
<code>SensorManager</code>	Podstawowy interfejs do wszystkich zainstalowanych w urządzeniu sensorów.
<code>Sensor</code>	Reprezentuje pojedynczy sensor.
<code>SensorEvent</code>	Reprezentuje odczyt z sensora.
<code>SensorEventListener</code>	Interfejs ten jest używany do odbierania zdarzeń <code>SensorEvents</code> niemal w czasie rzeczywistym.

Po odczytaniu referencji można użyć tej zmiennej w całej aplikacji do realizowania interakcji z samymi sensorami. Na przykład aplikacja `SenseBot` korzysta z czujnika orientacji. Aby uzyskać referencję do obiektu tego czujnika, należy wywołać z klasy `SensorManager` metodę `getDefaultSensor()`:

```
Sensor orientationSensor =
    sManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
```

W aplikacji tej korzystamy tylko z czujnika orientacji, ale Android oferuje znacznie więcej. Przyjrzyjmy się liście czujników dostępnych w systemie Android 2.2.

### 14.2.1. Typy czujników

Android obsługuje typy czujników wymienione w tabeli 14.3.

**Tabela 14.3.** Czujniki systemu Android

<code>Sensor.TYPE_ACCELEROMETER</code>	Mierzy przyspieszenie w trzech wymiarach.
<code>Sensor.TYPE_GYROSCOPE</code>	Żyroskop.
<code>Sensor.TYPE_LIGHT</code>	Czujnik światła.
<code>Sensor.TYPE_MAGNETIC_FIELD</code>	Kompas mierzący pole magnetyczne.
<code>Sensor.TYPE_ORIENTATION</code>	Mierzy położenie w trzech wymiarach.
<code>Sensor.TYPE_PRESSURE</code>	Mierzy ciśnienie.
<code>Sensor.TYPE_PROXIMITY</code>	Mierzy odległość telefonu od innego obiektu, na przykład naszego ucha.
<code>Sensor.TYPE_TEMPERATURE</code>	Mierzy temperaturę otoczenia.

Każdy z obiektów czujnika może zawierać kilka przydatnych i interesujących atrybutów, takich jak:

- ◆ nazwa czujnika,
- ◆ zużycie prądu — w mA,
- ◆ rozdzielczość,
- ◆ maksymalny zakres,
- ◆ dostawca,
- ◆ wersja.

Czujnik orientacji w telefonie Nexus One ma następującą charakterystykę:

- ◆ Nazwa: *AK8973 Orientation Sensor*
- ◆ Pobór prądu: *7,0 mA*
- ◆ Rozdzielczość: *1,0 stopień*
- ◆ Maksymalny zakres: *360 stopni*

Teraz, gdy wiemy, jak uzyskać dostęp do czujnika poprzez `SensorManager`, zajmijmy się odczytem jego wartości.

### 14.2.2. Odczyt wartości czujnika

Wartości z czujników odczytujemy z użyciem obiektu implementującego interfejs `SensorEventInterface`. Do metody o nazwie `onSensorChanged()` wysyłane są obiekty `SensorEvent`. Klasa `SensorEvent` zawiera cztery pola wymienione w tabeli 14.4.

**Tabela 14.4.** Pola klasy `SensorEvent`

Pole	Komentarz
<code>accuracy</code>	Pole typu całkowitego reprezentujące dokładność odczytu szacowaną przez czujnik.
<code>sensor</code>	Referencja do czujnika, który utworzył ten obiekt <code>SensorEvent</code> .
<code>timestamp</code>	Znacznik czasu z dokładnością nanosekundową, informujący o momencie utworzenia zdarzenia. Pole to może być przydatne przy korelowaniu wielu zdarzeń.
<code>values[3]</code>	Wartości z czujnika zapisane jako tablica liczb zmiennoprzecinkowych z trzema wartościami. Jednostki i dokładność tych trzech wartości jest zależna od czujnika.

Obiekt `SensorEventListener` otrzymuje zdarzenie za każdym razem, gdy zmieni się wartość sensora. Na poniższym listingu przedstawiona jest uproszczona wersja metody `onSensorChanged()` z aplikacji `SenseBot`.

**Listing 14.4.** Uproszczona wersja metody `onSensorChanged()`

```
public void onSensorChanged(SensorEvent event) { ← 1
    try {
        if (bConnected == false) return; ← 2
        StringBuilder sb = new StringBuilder();
        sb.append "[" + event.values[0] + " "; ← 3
        sb.append "[" + event.values[1] + " ";
        sb.append "[" + event.values[2] + " ";
        readings.setText(sb.toString()); ← 4

        // przetworzenie tych danych z czujnika ← 5
        // updateMotors(); ← 6
    } catch (Exception e) {
        Log.e(tag, "Błąd w onSensorChanged :: " + e.getMessage());
    }
}
```

Za każdym razem, gdy dostępny jest obiekt `SensorEvent` ❶, jest on przekazywany do metody `onSensorChanged()`. Pierwszą operacją w tym kodzie jest sprawdzenie, czy mamy działające połączenie z robotem ❷. Jeżeli nie ma połączenia, ignorujemy dane. Każda z trzech wartości jest odczytywana i formatowana ❸ do wyświetlenia w polu `TextView` ❹. Wartości są interpretowane ❺ i odpowiednie instrukcje są wysyłane do kontrolera sterującego silnikami robota ❻. Kod odpowiedzialny za interpretację i interakcję z robotem przedstawiono w dalszej części tego rozdziału.

Aplikacja musi zarejestrować swój obiekt `SensorEventListener`, aby otrzymywać te powiadomienia. W następnym punkcie przedstawimy zalecany sposób wykonywania procesu rejestracji.

### 14.2.3. Włączanie i wyłączanie czujników

Obiekt implementujący interfejs `SensorEventListener` otrzymuje komunikaty wyłącznie wtedy, gdy jest zarejestrowany. Klasa `SensorManager` zawiera dwie funkcje pozwalające aplikacji na rejestrowanie zdarzeń czujnika. W kontekście aplikacji `SenseBot` interesuje nas otrzymywanie zdarzeń z czujnika położenia wyłącznie wtedy, gdy urządzenie jest połączone z robotem poprzez Bluetooth. Dlatego kod rejestracji umieściliśmy w przedstawionej wcześniej metodzie `handleConnected()`. Na poniższym listingu zamieszczony jest nowy kod, dodany do metody `handleConnected()`.

**Listing 14.5.** Kod rejestracji czujnika

```
sManager.registerListener(SenseBot.this, ← ❶  
    sManager.getDefaultSensor(  
        Sensor.TYPE_ORIENTATION), ← ❷  
    SensorManager.SENSOR_DELAY_UI); ← ❸
```

Metoda `registerListener()` z klasy `SensorManager` oczekuje trzech argumentów potrzebnych do przekazania danych z czujnika do aplikacji. Pierwszym argumentem jest obiekt implementujący interfejs `SensorEventListener`, którym w tym przypadku jest sama klasa `SenseBot.this` ❶. Drugim argumentem jest obiekt interesującego nas czujnika. W tym przypadku jesteśmy zainteresowani śledzeniem wartości z czujnika położenia ❷. Częstotliwość, z jaką są aktualizowane dane czujników, jest zmienna i definiowana przez programistę za pomocą trzeciego parametru. W tym przypadku użyliśmy stałej `SensorManager.SENSOR_DELAY_UI` ❸, która jest dobrą, uniwersalną wartością. Dla gier i innych aplikacji czasu rzeczywistego należy użyć większych wartości.

Jak pamiętamy, czujnik położenia potrzebuje 7 mA. Aby wydłużyć czas pracy baterii, należy pamiętać o wyłączeniu czujnika, jeżeli nie jest on potrzebny. W aplikacji `SenseBot` istnieją dwa miejsca, w których ma to miejsce. Pierwszym jest metoda `handleDisconnected()` — gdy utracimy połączenie z robotem, nie ma sensu próbować odczytywać dane z sensora. Innym miejscem, w którym należy dodać funkcję „wyrejestrowania”, jest metoda cyklu życia aktywności, `onStop()`.

Niezależnie od miejsca, z którego wywołany jest kod, obiekt `SensorEventListener` jest wyrejestrowywany za pomocą prostej metody `unregisterListener()` z klasy `SensorManager`:

```
sManager.unregisterListener(SenseBot.this);
```



Trzeba pamiętać, że jeżeli aplikacja zarejestrowała więcej niż jeden typ czujnika, konieczne jest wyrejestrowanie wszystkich tych czujników.

Wiemy już, jak podłączyć się do robota i odczytywać wartości z czujnika położenia. Czas połączyć ze sobą te informacje i zbudować aplikację SenseBot!

## 14.3. Budowanie aplikacji SenseBot

---

Założenia aplikacji SenseBot są proste — chcemy sterować robotem LEGO Mindstorms NXT<sup>5</sup> przez zmianę orientacji telefonu Android. Nie korzystamy z przewodów — cała komunikacja jest realizowana poprzez Bluetooth, a orientacja telefonu pozwala określić sposób poruszania się robota. Choć robot LEGO jest programowalny, korzystamy tylko z wbudowanych możliwości manipulowania poszczególnymi silnikami. Zaletą takiego podejścia jest możliwość wykorzystywania tego programu z niemal każdym robotem LEGO, niezależnie od umiejętności programisty robota. Jedynym wymaganieniem jest podłączenie silników do portów wyjściowych B oraz C, co jest częstą praktyką przy konstruowaniu robotów LEGO NXT. Na rysunku 14.5 pokazany jest robot z o prostej, dwusilnikowej konstrukcji.



**Rysunek 14.5.**  
Prosty robot LEGO  
NXT z silnikami  
podłączonymi  
do portów B  
oraz C

Robot może jechać w przód, w tył, skręcać w lewo oraz w prawo. Przechylenie telefonu do przodu lub do tyłu spowoduje jazdę robota, natomiast przechylenie go w lewo lub prawo powoduje, że robot skręca.

Choć robot jest kontrolowany wyłącznie przez ruchy telefonu, nadal musimy utworzyć przydatny i intuicyjny interfejs użytkownika. W rzeczywistości interfejs użytkownika w tej aplikacji jest bardzo ważny.

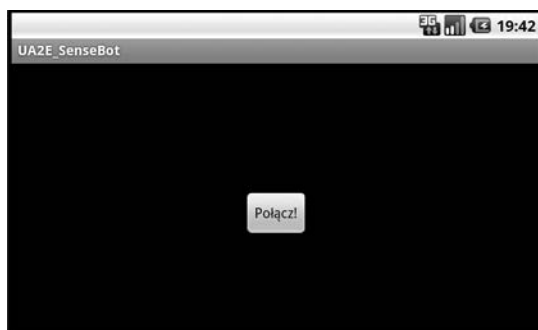
### 14.3.1. Interfejs użytkownika

Interfejs użytkownika tej aplikacji jest prosty, ale musi być również intuicyjny dla użytkownika. Chcemy pokazać użytkownikowi, co się dzieje, aby zapewnić mu informacje na temat sposobu używania aplikacji. Dodatkowo komunikujemy się z mechanicznym robotem, który niekoniecznie może prawidłowo działać. Robot może wykonać nie-

---

<sup>5</sup> Jeżeli masz w rodzinie przyszłego inżyniera robotyki, warto zapoznać się z ligą First Lego League: <http://www.firstlegoleague.org/>.

oczekiwaną akcję — dlatego pożądane jest, aby można było porównać ruchy robota z wizualnymi wskaźnikami wyświetlanymi w interfejsie użytkownika. Musimy więc informować użytkownika o stanie silników przez cały czas, gdy urządzenie Android jest podłączone do robota. Na rysunku 14.6 pokazany jest domyślny interfejs użytkownika przed podłączeniem robota.



**Rysunek 14.6.**  
Oczekiwanie  
na podłączenie  
robota

Kliknięcie przycisku *Połącz* inicjuje sekwencję łączenia, zawierającą metodę `findRobot()`, pokazaną wcześniej w punkcie „Podłączanie się do zdalnego urządzenia”. Po podłączeniu do robota musimy ukryć przycisk *Połącz* i zapewnić możliwość odłączenia od robota za pomocą przycisku *Rozłącz*. Dodatkowo chcemy wyświetlić stan silników i wyświetlać odczyty z czujnika. Na rysunku 14.7 pokazana jest aplikacja po połączeniu, gdzie robot ma wyłączone silniki.



**Rysunek 14.7.**  
Podłączenie  
do robota  
z zatrzymanymi  
silnikami

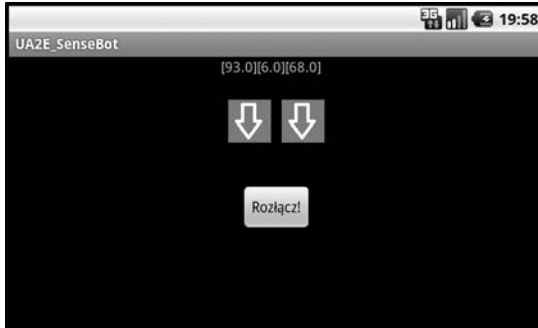
### Uwaga

Wskaźniki silników na ekranie są wartościami zdefiniowanymi w aplikacji i skorelowanymi z instrukcjami sterowania silnikami, wysyłanymi do robota. Nie są to wartości odczytywane z robota.

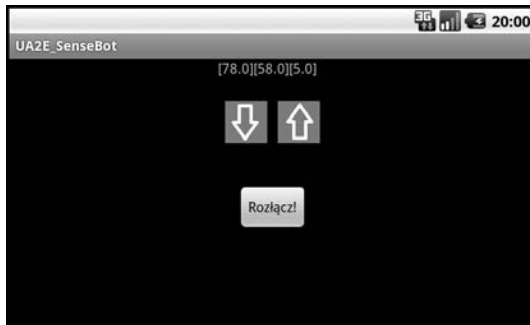
Jeżeli silniki robota pracują, a na ekranie pokazane jest, że oba są zatrzymane, występuje problem z wysłanym poleceniem lub z samym robotem.

Na rysunku 14.8 pokazany jest ekran aplikacji w przypadku wydania polecenia cofania.

Na rysunku 14.9 pokazana jest aplikacja wysyłająca komunikat skrętu w lewo. Aby wykonać taką operację, lewy silnik pracuje w tył, a prawy w przód.

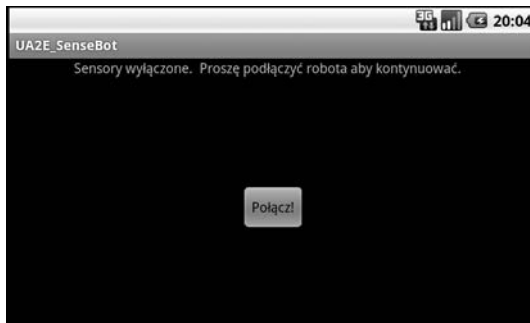


**Rysunek 14.8.**  
 Oba silniki  
 działają w tył



**Rysunek 14.9.**  
 Skręt w lewo

Na koniec, gdy aplikacja odłączy się od robota (albo przez kliknięcie przycisku *Rozłącz*, albo przez wyłączenie robota), aplikacja wykrywa zdarzenie rozłączenia i wywołuje metodę `handleDisconnect()`, a interfejs użytkownika jest aktualizowany, jak jest to pokazane na rysunku 14.10.



**Rysunek 14.10.**  
 Stan odłączenia,  
 oczekiwanie na  
 nowe połączenie

Interfejs użytkownika składa się z dwóch elementów View oraz trzech rysunków<sup>6</sup>: stop, w górę (w przód) oraz w dół (w tył). Na podstawie wartości z czujnika odpowiedni element View ma ustawiane odpowiednie tło.

<sup>6</sup> Warto pobrać aplikację wyświetlającą wszystkie zasoby znajdujące się w *android.R.drawable* urządzenia Android: <http://www.appbrain.com/app/android-drawables/auw.apps.androidDrawables>.

Aplikacja ta jest tak zależna od orientacji telefonu przy sterowaniu robotem, że nie możemy pozwolić na zmiany z układu pionowego na poziomy i odwrotnie, ponieważ zarówno powoduje to ponowne uruchomienie aktywności, co może wprowadzić sporo zamieszania, jak również zmienia orientację czujników. Aby spełnić to wymaganie, w pliku `AndroidManifest.xml` należy do znacznika `activity` dodać poniższy atrybut:

```
android:screenOrientation=landscape
```

Po skonfigurowaniu orientacji nie ma obawy o zmianę układu z poziomego na pionowy przy sterowaniu robotem. Uznaliśmy, że trzymanie telefonu poziomo pozwala na wygodne „kierowanie”.

Aby zapewnić dokładne skoordynowanie interfejsu użytkownika z fizycznymi silnikami, konieczne jest przygotowanie mechanizmu sprzężenia, dzięki czemu można lepiej sterować robotem, jak i pomóc przy rozwiązywaniu problemów z anomaliami w czasie testowania tego projektu inżynierskiego.

Komunikacja jest gotowa do działania, a czujniki orientacji dostarczają danych; czas zająć się interpretowaniem wartości z czujników.

### 14.3.2. Interpretowanie wartości czujnika

Aby sterować robotem przy użyciu orientacji telefonu, należy zdefiniować „martwą strefę”, której „środek” jest reprezentowany przez pozycję telefonu w poziomie, z niewielkim odchyleniem w tył. Po zdefiniowaniu tej pozycji centralnej dodajemy w wymiarach  $x$  i  $y$  wygodne w użyciu zakresy czułości. Dopóki orientacja telefonu nie przekroczy wartości czułości, silniki pozostają wyłączone. Zmienne o nazwach `xCenter`, `yCenter`, `xSensitivity` oraz `ySensitivity` pozwalają utworzyć „neutralny prostokąt”.

Spójrzmy na metodę `onSensorChanged()`: w niej właśnie odbieramy obiekty `SensorEvent` zawierające wartości dla każdego z wymiarów,  $x$ ,  $y$  oraz  $z$ . Na poniższym listingu pokazana jest kompletna implementacja tej metody wraz z analizą czujników oraz sugestiami ruchu.

**Listing 14.6.** Metoda `onSensorChanged()` interpretująca orientację

```
public void onSensorChanged(SensorEvent event) {
    try {
        if (bConnected == false) return;
        StringBuilder sb = new StringBuilder();
        sb.append "[" + event.values[0] + " ";
        sb.append "[" + event.values[1] + " ";
        sb.append "[" + event.values[2] + " ";
        readings.setText(sb.toString());
        // przetwarzanie danych z czujników
        movementMask = MOTOR_B_STOP + MOTOR_C_STOP; ← ❶

        if (event.values[2] < (yCenter - ySensitivity)) {
            movementMask = MOTOR_B_FORWARD + MOTOR_C_FORWARD; ← ❷
            motorPower = 75; ← ❸
        } else if (event.values[2] > (yCenter + ySensitivity)) {
            movementMask = MOTOR_B_BACKWARD + MOTOR_C_BACKWARD; ← ❹
            motorPower = 75; ← ❺
        }
    } catch (Exception e) {
        // ...
    }
}
```

```

    } else if (event.values[1] > (xCenter + xSensitivity)) {
        movementMask = MOTOR_B_BACKWARD + MOTOR_C_FORWARD;

        motorPower = 50; ← 5
    } else if (event.values[1] < (xCenter - xSensitivity)) {
        movementMask = MOTOR_B_FORWARD + MOTOR_C_BACKWARD;
        motorPower = 50; ← 5
    }
    updateMotors(); ← 6
} catch (Exception e) {
    Log.e(tag, "Błąd onSensorChanged ::" + e.getMessage());
}
}

```

Przy interpretacji wartości dla silników domyślnie mamy oba silniki zatrzymane **1**. Zwróć uwagę, że silniki B i C są zarządzane osobno. Na początek sprawdzamy, czy wartość czujnika *y* jest poza martwą strefą osi *y* **2**. Jeżeli wykryta wartość jest poza granicą „przechylony w przód”, ruszamy robotem do przodu. Podobnie jeżeli odczytana wartość jest mniejsza od granicy wartości spoczynkowej, ruszamy robotem do tyłu, przez włączenie wstecznych obrotów obu silników. Jeżeli nie określiliśmy ruchu robota w przód lub w tył, sprawdzamy kolejne opcje, dla ruchu w lewo lub w prawo **3**. Jeżeli robot przesuwa się w przód lub w tył, szybkość jest ustawiona na 75% **4**. Jeżeli robot skręca, jego szybkość jest ustawiona na 50% **5**. Ostatnim krokiem jest przekształcenie tych masek ruchu na prawdziwe akcje przez zmodyfikowanie stanu silników **6** i zaktualizowanie interfejsu, aby odzwierciedlał te polecenia.

Ponieważ metoda `onSensorChanged()` w pełni przetwarza dane `SensorEvent`, czas na uruchomienie silników robota i zaktualizowanie interfejsu użytkownika.

### 14.3.3. Jazda robotem

Jazda robotem jest bardzo prosta — i jednocześnie złożona — ponieważ polega tylko na włączeniu silników za pomocą serii poleceń. Sam protokół poleceń jest przedstawiony w następnym punkcie, a teraz skupimy się na metodzie `updateMotors()`, w której modyfikowany jest interfejs użytkownika oraz stan silników. Na poniższym listingu zamieszczona jest metoda `updateMotors()`.

**Listing 14.7.** Metoda `updateMotors()`

```

private void updateMotors() {
    try {
        if ((movementMask & MOTOR_B_FORWARD) == MOTOR_B_FORWARD) { ← 1
            motorB.setBackgroundResource(R.drawable.uparrow); ← 2
            MoveMotor(MOTOR_B, motorPower); ← 3
        }
        else if ((movementMask & MOTOR_B_BACKWARD) == MOTOR_B_BACKWARD) { ← 1
            motorB.setBackgroundResource(R.drawable.downarrow); ← 2
            MoveMotor(MOTOR_B, -motorPower); ← 3
        }
        else {
            motorB.setBackgroundResource(R.drawable.stop); ← 2
            MoveMotor(MOTOR_B, 0);
        }
    }
}

```

```

if ((movementMask & MOTOR_C_FORWARD) == MOTOR_C_FORWARD) {
    motorC.setBackgroundResource(R.drawable.uparrow); ← ❷
    MoveMotor(MOTOR_C,motorPower); ← ❸
} else if ((movementMask & MOTOR_C_BACKWARD) == MOTOR_C_BACKWARD) {
    motorC.setBackgroundResource(R.drawable.downarrow); ← ❷
    MoveMotor(MOTOR_C,-motorPower); ← ❸
} else {
    motorC.setBackgroundResource(R.drawable.stop); ← ❷
    MoveMotor(MOTOR_C,0);
}
} catch (Exception e) {
    Log.e(tag,"Błąd w updateMotors ::" + e.getMessage());
}
}

```

W metodzie `updateMotors()` porównywane są żądania ruchu zdefiniowane w postaci zmiennej `movementMask`, osobnej dla każdego z silników ❶. Gdy zostanie znaleziona pasująca wartość — na przykład gdy ustawiony jest bit `MOTOR_B_FORWARD` — włączany jest dany silnik w zdefiniowanym kierunku i szybkości ❸. Kierunek ujemny oznacza jazdę w tył, a wartość szybkości jest skalowana do zakresu od 0 do 100. Dodatkowo aktualizowany jest interfejs użytkownika ❷ w połączeniu z samymi silnikami, co daje użytkownikowi możliwie dokładny obraz działania.

### 14.3.4. Komunikacja z robotem

Protokół komunikacji pozwalający na interakcję z robotem LEGO NXT składa się z poleceń strukturalnych z opcjonalnym protokołem odpowiedzi. Każdy pakiet danych jest umieszczony w kopercie opisującej jego rozmiar. Wewnątrz koperty każde polecenie protokołu Direct Command posiada standardowy nagłówek, po którym następują specyficzne parametry. W aplikacji tej potrzebujemy tylko jednego polecenia — powodującego ustawienie stanu działania silnika. Kod pozwalający na zbudowanie i wysłanie tych pakietów jest pokazany na poniższym listingu.

**Listing 14.8.** Metoda `MoveMotor`

```

private void MoveMotor(int motor,int speed)
{
    try
    {
        byte[] buffer = new byte[14]; ← ❶
        buffer[0] = (byte) (14-2); // długość lsb
        buffer[1] = 0; // długość msb
        buffer[2] = 0; // polecenie Direct Command (z odpowiedzią)
        buffer[3] = 0x04; // ustawienie stanu wyjścia
        buffer[4] = (byte) motor; // wyjście 0, 1, 2 (silniki A, B, C)
        buffer[5] = (byte) speed; // moc
        buffer[6] = 1 + 2; // włączenie silnika + hamulec pomiędzy PWM
        buffer[7] = 0; // regulacja
        buffer[8] = 0; // rotacja skrętu
        buffer[9] = 0x20; // stan działania
        buffer[10] = 0; // cztery bajty danych pozycji
    }
}

```

```

buffer[11] = 0; // ustawione na zero
buffer[12] = 0;
buffer[13] = 0;

os.write(buffer);
os.flush();
byte response [] = ReadResponse(4);
}
catch (Exception e)
{
    Log.e(tag, "Błąd w MoveForward(" + e.getMessage() + ")");
}
}

```

Kod ten realizuje prostą, choć precyzyjną operację formatowania polecenia wysyłanego do robota LEGO, które zapewnia bezpośrednią kontrolę nad silnikami. Na początek deklarujemy bufor o odpowiedniej wielkości **1**. Rozmiar tego bufora jest definiowany przez polecenie `SetOutputState`, które jest jednym z wielu poleceń obsługiwanych przez robota. Każda z informacji jest umieszczana w odpowiednim miejscu bufora **2**. Po sformatowaniu bufora polecenia jest on zapisywany do gniazda, a gniazdo opróżniane **3**. Kod odpowiedzi odczytywany przez metodę `ReadResponse()` jest znacznikiem prawidłowego odbioru polecenia. Jak się okazuje, oprócz specyficznego formatowania danych sterujących robotem, wysyłanie i odbieranie danych poprzez Bluetooth jest równie proste, co odczyt i zapis do bufora bajtów.

Na tym etapie czujniki działają, a urządzenie Android i robot LEGO komunikują się ze sobą. Z czasem, po nabraniu praktyki, możesz się stać doskonałym pilotem robota LEGO. Pełny kod źródłowy tej aplikacji jest dostępny do pobrania.

## 14.4. Podsumowanie

W tym rozdziale wprowadziliśmy dwie funkcje platformy Android zorientowane sprzętowo: Bluetooth i czujniki. Z tych dwóch pozornie niezwiązanych obszarów działania wyrosła aplikacja pozwalająca na operowanie robotem LEGO Mindstorms NXT. Pokazaliśmy tu najważniejsze kroki wymagane do połączenia urządzenia Android ze zdalnym partnerem obsługującym Bluetooth, z użyciem protokołu zastępującego połączenie kablowe, RFCOMM. Ten kanał komunikacyjny jest używany do wymiany zestawu poleceń znanych jako protokół `Direct Command`, udostępniany przez kontroler LEGO NXT. Dzięki tym poleceniom można manipulować silnikami robota, wprawiając go w ruch. Aby interfejs użytkownika był możliwie intuicyjny, skorzystaliśmy z czujnika orientacji, wbudowanego w większość telefonów z systemem Android, który pozwala na odczytywanie ruchów użytkownika. Położenie urządzenia jest interpretowane i odpowiadająca im seria poleceń jest wysyłana do robota. Sensory nie tylko są dobrą metodą sterowania robotem, ale również dają dużo zabawy!

Oprócz komunikacji poprzez Bluetooth i użycia czujników w rozdziale tym pokazaliśmy również techniki zapewniające intuicyjne informowanie użytkownika o operacjach wykorzystywanych przez aplikację. Na przykład gdy silniki są włączone, użytkownik widzi na ekranie kierunek obrotu każdego z nich. Podobnie ruchy użytkownika są przetwarzane wyłącznie w przypadku aktywnego połączenia Bluetooth. Ponieważ jest to

scenariusz sterowany zdarzeniami, aplikacja demonstruje nasłuch tych zdarzeń poprzez dynamicznie rejestrowany obiekt `BroadcastReceiver`, z odpowiednimi filtrami intencji.

Mamy nadzieję, że lekcja użycia komunikacji Bluetooth oraz czujników, przedstawiona w tym rozdziale, była ciekawa, a jeżeli masz dostęp do robota LEGO Mindstorm, zachęcamy do jazdy próbnej.

W następnym rozdziale pokażemy inne sposoby podłączania urządzenia Android do zewnętrznego świata — tym razem będziemy wykorzystywać możliwości platformy do synchronizowania danych z popularną biznesową witryną społecznościową LinkedIn.



# Skorowidz

## A

- Ableson Frank, 18
- AccountManager, 451
- adapter, 96, 112
  - ArrayAdapter, 96
  - CursorAdapter, 96
  - GalleryAdapter, 96
  - ListAdapter, 96
- adapter synchronizacji, 460
- Adaptive Multi-Rate (AMR), 282
- adb shell, 381
- adres URI, 40, 41, 49
- adresy IP, 193
- ADT, 80
- Advanced Audio Coding (AAC), 282
- Advanced Video Coding (AVC H.264), 282
- akcja, 131
  - EDIT, 41
  - Intent.ACTION\_CALL, 134, 226
  - Intent.ACTION\_DELETE, 134
  - Intent.ACTION\_EDIT, 134, 442
  - Intent.ACTION\_INSERT, 134, 441
  - Intent.ACTION\_VIEW, 133
  - MAIN, 51, 128
  - PICK, 41
  - VIEW, 41
- aktualizacja dynamiczna, 228
- aktualizowanie widżetów, 529, 534
- aktywności, 92, 103
  - działające, 52
  - niedziałające, 52
  - uśpione, 53
- aktywność
  - CloseJob, 366
  - FieldService, 340, 343, 344
  - ManageJobs, 340
  - ManageJob, 372
  - ManageJobs, 359
  - MapViewActivity, 316, 318
  - RefreshJobs, 355
  - ShowJob, 340, 362
  - ShowSettings, 340
  - SiteMonitorConfigure, 539
  - SMSNotifyActivity, 239
  - Splash, 339, 340, 341
- alarm, 247, 531
  - ELAPSED\_REALTIME, 250
  - ELAPSED\_REALTIME\_WAKEUP, 250
  - RTC, 250
  - RTC\_WAKEUP, 250
- AMR-NB, 282
- AMR-WB, 282
- analiza pliku układu XML, 120

- analizowanie danych intencji, 136
  - Android, 30
  - Android Cloud to Device Messaging (C2DM), 232
  - Android Debug Bridge, 69, 606
  - Android Interface Definition Language (AIDL), 130
    - pakiety i interfejsy, 149
    - typy danych, 149
  - Android Market, 36, 603
  - Android Open Source Platform (AOSP), 35
  - Android SDK, 57, 60, 68, 89, 378, 585
  - animacja
    - <alpha>, 125
    - <rotate>, 126
    - <scale>, 125
    - <translate>, 126
  - animacja poklatkowa, 261
  - Ant in Action
    - Second Edition of Java Development with Ant, 79
  - API, 81
    - AhhtClient, 201
    - Apache HttpClient, 199
    - GData, 212
    - Google Base Atom, 105
    - graficzne, 256
    - java.net, 199
    - OpenGL ES, 268
    - Yahoo! Weather, 130
  - aplikacja
    - Android
      - kod źródłowy, 341
      - określanie procesów, 337
      - pliki zasobów, 339
      - podstawowe wymagania, 333
      - sposoby przesyłania danych, 350
    - AppWidgetHost, 505
    - Dev Tools, 464
    - hybrydowa, 487, 489
    - klient-serwer, 195
    - konsolidowana statycznie, 381
    - mobilnego serwisanta, 332
    - OCR, 560
    - SenseBot, 417, 420
    - serwera DayTime, 393, 399
    - SiteMonitor, 509–515
      - interfejs użytkownika, 509
      - podstawy architektury, 515
      - schemat projektu, 513
    - UA2EFindingEdges, 563
  - aplikacje integracyjne, 465
  - aplikacje systemu, 52
  - aplikacje użytkownika, 67
  - architektura aplikacji i integracji, 335
  - argument
    - argc, 380, 390
    - argv[], 380, 390
  - arkusze stylów, 474
  - arkusze stylów dedykowane, 475
  - arm-nonelinux-gnueabi-gcc, 379
  - arm-none-linux-gnueabi-ld, 379
  - arm-none-linux-gnueabi-objdump, 379
  - atrybut
    - android
      - id, 75
      - text, 552
    - autoLink, 511
    - media, 475
    - updatePeriodMillis, 528
  - atrybuty, 113
  - atrybuty animacji
    - duration, 126
    - interpolar, 126
    - startOffset, 126
  - Authentication key (Ki), 220
  - AVD Manager, 588
- ## B
- baza danych, 174
    - otwieranie bazy danych, 481
  - baza danych kontaktów, 443
  - baza danych MySQL, 373
  - baza danych SQLite, 396
  - bezpieczeństwo transmisji danych, 334
  - bezpieczeństwo urządzenia, 334
  - biblioteka
    - JNI, Java Native Interface, 563, 565
    - PhoneGap, 494
    - Stagefright, 37, 282
    - NDK, 562
    - przetwarzająca obraz, 565
    - testowa PV, 282
    - uruchomieniowa, 37
  - biblioteki, 37
    - OpenCORE, 37
    - OpenGL ES, 37
    - Scalable Games Language (SGL), 37
    - Secure Socket Layer (SSL), 37
    - SQLite, 37, 398
    - WebKit, 37
      - systemowe Androida, 385, 390
  - BlackBerry, 34
  - Bluetooth, 190, 409

BOOT\_COMPLETED, 143  
 budowanie dynamicznej wersji aplikacji, 387  
 bufor głębi, 275  
 Burnette Ed, 65

## C

CDMA, Code Division Multiple Access, 218–220  
 cross-kompilacja, 382  
 cross-kompilator, 379  
 CSS, 466  
 cykl życia aktywności, 100  
   faza całego cyklu życia, 101  
   faza działania, 101  
   faza widoczności, 101  
 cykl życia aplikacji, 92  
 cykl życia synchronizacji, 458  
 czujnik orientacji, 417, 418  
 czujniki systemu Android, 417

## D

dane, 131  
   binarne, 49  
   instancyjne i metody klasy SiteMonitorModel, 516  
   lokalizacji, 311  
   POST, 372  
   SensorEvent, 424  
   tekstowe, 358  
   widżetu, 526  
   XML, 352  
 datagram, 192  
 DDL, 398  
 DDMS, 381  
 debugowanie aplikacji, 86, 464  
 definiowanie  
   animacji, 125  
   CONTENT\_URI oraz MIME\_TYPE, 182  
   intencji, 131  
   kolorów, 123  
   kont, 448  
   plików XML, 125  
   stylów, 123  
   surowych plików, 125  
   tablic, 124  
   układu i widoków, 95  
   widoków i układów w zasobach XML, 120  
   wymiarów, 123  
 Delicious, 209  
 deskryptor aplikacji, 51  
 deskryptor intencji, 46

Developer Network, 437  
 dezasemblacja pliku wykonywalnego, 391  
 diagram klas API widoków, 104  
 dodawanie  
   kontaktów, 444  
   łączy zewnętrznych, 132  
   widżetu do ekranu głównego, 506  
 dokumentacja Android SDK, 60  
 dostawca  
   LocationManager.GPS\_PROVIDER, 314  
   LocationManager.NETWORK\_PROVIDER, 314  
   LocationManager.PASSIVE\_PROVIDER, 314  
   treści, 49  
 dostęp do sieci, 190  
 drzewo hierarchiczne, 113  
 drzewo widoków, 120  
 dynamiczny interfejs użytkownika, 96  
 dystrybucja aplikacji Android, 596, 606  
 dziennik systemu, 45

## E

Eclipse IDE, 585  
 Eclipse in Action  
   A Guide for Java Developers, 65  
 ekran, 93, 113  
   dyspozytora, 373  
   główny, 343  
   MapViewActivity, 318  
   powitalny, 341  
   startowy, 341  
   ustawień, 346  
   ustawień Bluetooth, 411  
 eksportowanie usługi, 152  
 Electronic Serial Number (ESN), 220  
 element  
   MIME, 182  
   <activity>, 127  
   <application>, 127  
   <array>, 125  
   <dimen>, 123  
   <intent-filter>, 128, 135  
   <manifest>, 127  
   <userpermission>, 127  
   <uses-permission>, 128  
   div, 480  
   ImageView, 576  
   manifest, 51  
   ProgressDialog, 366  
   TextView, 400, 487  
   uses-library, 319  
   VideoView, 285, 286

elementy  
 intencji, 131  
 menu, 98  
 podrzędne <item>, 125  
 emulator, 295, 305  
 emulator Android, 62, 80, 288, 292, 381, 588  
 emulator-tcp5554, 65  
 encje, 335  
 etykiety, 122

## F

figura  
 GL\_LINE\_LOOP, 273  
 GL\_LINE\_STRIP, 273  
 GL\_LINES, 273  
 GL\_POINTS, 273  
 GL\_TRIANGLE\_FAN, 273  
 GL\_TRIANGLE\_STRIP, 273  
 GL\_TRIANGLES, 273  
 figury proste OpenGL ES, 273  
 filtr intencji, 41, 130, 135, 539  
 filtrowanie informacji, 78  
 fokus, 115  
 typ DEFAULT\_FOCUS, 115  
 typ WEAK\_FOCUS, 115  
 folder src, 72  
 format  
 .dex, 79  
 AARRGGBB, 576  
 GPS Exchange, 307  
 ISO, 543  
 JSON, 336  
 Keyhole Markup Language (KML), 309  
 @string/<identyfikator>, 552  
 formatowanie numerów telefonów, 228  
 forum WWW, 23  
 funkcja  
 alert(), 492  
 converttgray(), 567, 569  
 deleteAllRecords(), 484  
 detectedges(), 569  
 exit(), 390  
 file\_get\_contents(), 375  
 getJobsXML(), 375  
 glDrawArrays(), 274  
 GLU.gluLookAt(), 278  
 gluPerspective(), 276  
 parametr aspect, 276  
 parametr fovy, 276  
 parametr gl, 276

parametr zFar, 276  
 parametr zNear, 276  
 httons(), 396  
 LogCat(), 567  
 main(), 380, 392  
 RecordHit(), 396  
 saveRecord(), 484  
 SetSearchTerm(), 491, 493  
 toString(), 359  
 transaction(), 482  
 wait(), 495  
 funkcje debugowania, 86  
 funkcje eksportu i importu danych, 598  
 funkcje gniazd, 396  
 funkcje lokalizacyjne, 304  
 funkcje środowiska Eclipse, 63  
 funkcje telefoniczne, 221

## G

Gallardo David, 65  
 geokodowanie, 325  
 globalne dane stanu, 98  
 gniazdo, 199  
 gniazdo serwera, 196  
 Google Maps, 82, 304  
 GPS, 33  
 GSM, Global System for Mobile  
 Communications, 218

## H

hasło, 450  
 Hatcher Erik, 79  
 hierarchiczna struktura widoków, 92  
 hierarchiczne drzewo elementów View, 103  
 HTML, 466  
 HTTP, HyperText Transfer Protocol, 190, 199  
 HttpClient, 201

## I

ICANN, Internet Corporation  
 for Assigned Names and Numbers, 194  
 IDE Eclipse, 63  
 identyfikator  
 kontaktu, 439, 441  
 numeryczny widżetu, 515  
 php input, 375  
 procesu, 66  
 użytkownika, 79, 373, 450

IDL, język definicji interfejsu, 148  
implementacja biblioteki, 567  
implementacja interfejsu użytkownika, 346  
informacja o lokalizacji, 309  
instalacja  
  ADT, 590  
  Android SDK, 584  
  CodeSourcery, 379  
  Eclipse, 584  
  wtyczki ADT dla Eclipse, 584, 590  
instancja  
  AVD, 81  
  klasy android.media.MediaRecorder, 293, 297  
  MediaPlayer, 284  
instrukcja  
  import, 77  
  return, przepełnienie stosu, 390  
Integrated Circuit Card Identifier (ICCID), 220  
intencja, 39, 41, 130  
  inicjowanie akcji, 133  
  Intent.ACTION\_CALL, 134, 226  
  Intent.ACTION\_DIAL, 134, 226  
  jawna, 42  
  mechanizmy rozpoznawania, 132  
  metody do rozgłaszania, 142  
  niejawna, 42  
  wbudowana, 131  
  wypełnianie danych, 134  
  wywołanie jawne, 132  
  wywołanie niejawne, 131, 134  
  zdefiniowana, 43  
interfejs, 148  
  IBinder, 150  
  LocationListener, 317  
  programowy aplikacji (API), 81  
  SensorEventInterface, 418  
  SurfaceHolder, 271  
  użytkownika, 44, 77  
  użytkownika dyspozytora, 372  
  zdalny, 148, 151  
IMEI, International Mobile Equipment Identity, 220  
IMSI, International Mobile Subscriber Identity, 220  
International Telecommunication Union H.263  
  (H.263), 282  
IP, Internet Protocol, 190  
IPC, 148  
iPhone, 34

**J**

jarsigner, 600  
Java Development Tools (JDT), 64  
Java ME, 39  
Java Micro Edition, 34  
JavaBean, 98  
Javadoc, 60, 588  
JavaScript, 466  
jądro systemu Linux, 36  
jednowątkowość interfejsu, 117  
język C, 378  
język Java, 57  
JPEG, Joint Photographic Experts Group, 282  
jQuery, 468  
JRE, 584

**K**

kanal alfa, 568  
karta SD, 171  
karta SIM, 220  
katalog  
  android, 599  
  app, 67  
  drawable, 262  
  build, 562  
  gen, 149  
  jni, 565  
  libs, 565  
  platforms, 587  
  res, 73, 117  
  res/anim, 125, 262  
  res/drawables, 73  
  res/layout, 54, 73, 121  
  res/raw, 168, 283  
  res/values, 74, 123, 547  
  samples, 589  
  sdcard, 295  
  shared\_prefs, 164  
  system/lib, 385  
  tools, 68, 305, 589  
katalog docelowy, 383  
kategoria, 41  
  CATEGORY\_LAUNCHER, 41  
  LAUNCHER, 128  
keytool, 599  
King Chris, 18  
klasa  
  Activity, 44, 181  
  AlarmManager, 247, 253  
  AlertDialog, 99

## klasa

- android.content.ContentValues, 293
- android.app.Activity, 94
- android.util.Log, 46
- Application, 574
- AppWidgetManager, 514
- AppWidgetProvider, 505, 506, 514, 519
  - metoda checkForZombies, 520
  - metoda onDeleted, 520
  - metoda onDisabled, 520
  - metoda onEnabled, 520
  - metoda onReceive, 520
  - metoda onUpdate, 520
  - metoda UpdateOneWidget, 520
- Binder, 128
- Bluetooth, 410
- BroadcastReceiver, 46, 143, 240
- Bundle, 350
- Camera, 287
- ConnectivityManager, 190, 195
- ContentProvider, 49, 180–188, 516
- ContentResolver, 49
- Context, 46, 167
- Criteria, 314
- Cursor, 49
- DoCloseJob, 370
- Drawable, 257
- DrawSurfaceView, 276
- FieldService, 344
- Geocoder, 305, 325, 326
- GeoPoint, 319
- Handler, 109, 201, 265
- Intent, 40, 57, 133
- IntentFilter, 40
- ItemizedOverlay, 322
- java.text.DecimalFormat, 549
- java.text.DecimalFormatSymbols, 549
- java.text.SimpleDateFormat, 549
- java.util.Formatter, 554
- java.util.GregorianCalendar, 549
- java.util.Locale, 549
- JobEntry, 349
- JobList, 350
- JobListHandler, 352
- LayoutParams, 113
- Locale, 546
- LocationListener, 315
- LocationManager, 304, 311, 313
- LocationProvider, 311, 314
- Log, 78
- Looper, 109, 358
- MapActivity, 317, 318, 319
- MapController, 319
- MapView, 317, 319
- MapViewActivity, 312
- MediaRecorder, 292
- Message, 265
- Notification, 242, 243, 253
- NotificationManager, 47, 253
- Overlay, 319
- OverlayItem, 322
- PhoneNumberUtils, 227
- PhoneStateListener, 218
- Prefs, 343
- R, 56, 77, 120
- R.java, 92
- RefreshJobs, 356
- RemoteViews, 524
- Requester, 401
- ResponseHandler, 201
- Sensor, 417
- SensorEvent, 417
- SensorEventListener, 417
- SensorManager, 416, 417
- Service, 45
- SharedPreferences, 343, 519
- SiteMonitorConfigure, 514
- SiteMonitorBootstrap, 514, 533
- SiteMonitorModel, 514, 516, 529
- SiteMonitorWidget, 513
- SiteMonitorWidgetImpl, 514, 520
- SmsManager, 231
- SmsMessage, 231, 238
- Stub, 149
- SurfaceView, 271
- TelephonyManager, 218, 222
- Toast, 238, 241
- UA2EFindEdgesApp, 574
- UAChrome, 500
- View, 103, 368
- ViewGroup, 113
- WeatherAlertService, 141, 147
- WebChromeClient, 500
- WTApplication, 491, 496
- klasa bazowa
  - BroadcastReceiver, 143
  - ItemizedOverlay, 322
  - LayoutParams, 113
  - OverlayItem, 322
- klasa dziedzicząca, 45
- klasa globalna WTApplication, 491, 496
- klasa menedżera informacji, 221

klasa odbiornika, 144  
klasa pochodna, 95  
klasa pomocnicza, 203  
klasa wewnętrzna, 113  
klasa zasobów aplikacji, 118  
klauzula  
    Order By, 49  
    Where, 49  
klient, 194  
    J2ME, 215  
    DayTime, 400, 402  
kliknięcie, 40  
klucz API Google Maps, 319  
kod  
    bajtowy, 38  
    nasłuchu, 78  
    natywny, 576  
    obsługi ekranu, 94  
    odpowiedzi, 426  
    PHP aplikacji dyspozytora, 374  
    PHP do integracji z aplikacją mobilną, 375  
    serwera, 372  
    startowy, 391  
    symulatora, 81  
    z JavaScript, 491  
kodeki dźwięku, 282  
kodeki mowy, 282  
kodery oraz dekodery wideo, 282  
kolejkowanie podpisów, 368  
kompilator gcc, 379  
kompilowanie biblioteki JNI, 571  
kompilowanie pliku, 386  
komponent ListView, 105  
komponent Observable, 116  
komponenty widoku, 119  
komunikat skrętu w lewo, 421  
komunikat Toast, 238  
konfigurowanie  
    instancji widżetu, 527  
    karty SD, 286  
    obiektu Handler, 342  
    opcji budowania, 580  
    strumienia wejściowego, 415  
    środowiska emulowanego, 81  
    wtyczki Eclipse, 592  
konsolidowanie programu, 386  
konstruktor klasy ReviewAdapter, 112  
kontakt wywołany z aplikacji, 442  
kontener, 95  
kontener ViewGroup, 113  
konto Google, 433

konto Microsoft Exchange, 433  
kontrolka  
    EditText, 510  
    przeglądarki, 468, 487, 489  
    WebView, 488  
korzeń drzewa widoków, 121  
krawędzie obrazu, 576  
kreator projektu aplikacji, 72  
kwalifikatory do organizowania  
    i definiowania zasobów, 551

## L

licencja  
    Apache Software License (ASL), 35  
    General Public License (GPL), 35  
    użytkownika (EULA), 597  
    License Verification Library (LVL), 602  
LinkedIn, 430, 436  
    interfejs użytkownika, 451  
    logowanie, 453  
    synchronizowanie danych, 459  
    tworzenie konta, 450  
    uwierzytelnianie, 451  
linker, 379, 386–387, 399  
lista recenzji, 105  
lista zleceń, 336, 352  
localhost, 71  
Location Area Identity (LAI), 220  
LogCat, 596  
lokalizacja dokładna (FINE), 315  
lokalizacja zgrubna (COARSE), 315  
lokalizowanie  
    aplikacji, 550  
    aplikacji Android, 542  
    kodu Java, 553  
lokalna baza danych SQL, 477  
Loughran Steve, 79

## Ł

łącze symboliczne do katalogu, 561  
łączenie i rozdzielanie kontaktów, 435  
łączenie danych z widokiem, 112  
łączenie surowych danych kontaktów, 435

## M

macierz tożsamości, 274  
magazyn kluczy, 599  
magazyn SharedPreferences, 514

- manifest dostawcy treści, 187
- MapView, 324
- maszyna Java, 57
- maszyna wirtualna Dalvik, 37, 39
- McGovern Robert, 65
- mechanizm późnego łączenia, 134
- mechanizm rejestrowania, 46
- mechanizm SharedPreferences, 188
- menedżer SMS, 232
- menu
  - Develop, 485
  - Favorites, 86
  - Favorites Debug, 86
  - Favorites Run, 86
  - Window, 86
- metadane widżetu, 527
- metoda
  - addAccount(), 456
  - addToDB(), 294
  - AndroidBitmap\_getInfo(), 569
  - asInterface(), 150
  - beginRecording, 300
  - Binder.onTransact(), 151
  - cancel(), 247
  - Canvas.drawColor(), 370
  - checkForZombies(), 522
  - confirm(), 501
  - Context.bindService(), 153
  - Context.startService(), 156
  - Context.stopService(), 156
  - detectEdges(), 577
  - draw(), 324
  - execute(), 206
  - findViewById(), 122
  - findAll(), 499
  - findViewById(), 54, 78
  - formatNumber(), 228
  - GET, 207
  - getAppWidgetIds(), 524
  - getAvtivity(), 525
  - getDataFromSite(), 537
  - getDefault(), 546
  - getDefaultSensor(), 417
  - getFormattedDate(), 519
  - getInstance(), 524
  - getProvider(), 313
  - getSharedPreferences(), 160
  - getSystemService(), 195
  - getter(), 574
  - getText(), 55, 78
  - getView(), 112
  - glVertexPointer(), 274
  - handleDisconnected(), 419
  - handleConnected(), 415
  - handleDisconnect(), 422
  - handleDisconnected(), 415
  - handleMessage(), 110
  - loadReviews(), 108, 109
  - managedQuery(), 181
  - MediaRecorder.setAudioEncoder(), 293, 297
  - MediaRecorder.setOutputFormat(), 293, 297
  - MediaRecorder.setAudioSource(), 293, 297
  - MediaRecorder.setPreviewDisplay(), 293, 297
  - MediaRecorder.setVideoEncoder(), 297
  - MediaRecorder.setVideoSource(), 297
  - nextFocusDown(), 115
  - nextFocusLeft(), 115
  - nextFocusRight(), 115
  - nextFocusUp(), 115
  - onActivityResult(), 365, 438, 576
  - onBind(), 46
  - OnClickListener(), 78
  - onCreate(), 95, 100–101, 401
  - onCreateOptionsMenu(), 368
  - onDeleted(), 522
  - onDestroy(), 101, 530
  - onDisabled(), 523
  - onDraw(), 112, 370
  - onJsAlert(), 500
  - onKeyDown(), 291
  - onLayout(), 112
  - onListItemClick(), 108
  - onMeasure(), 112
  - onPageFinished(), 496
  - onPageStarted(), 496, 499
  - onPause(), 100, 319
  - onPrrformSync(), 459
  - onReceive(), 47, 522–523, 531
  - onReceivedSslError(), 496
  - onRestart(), 101
  - onResume(), 101, 319
  - onSensorChanged(), 418, 423
  - onServiceConnected(), 153
  - onStart(), 101
  - onStop(), 101, 419
  - onUpdate(), 522, 525, 531
  - onVisibilityChanged(), 112
  - openInputStream(), 49
  - PlaceCall(), 491
  - prepare(), 284, 293
  - ProgressDialog.show(), 109
  - ReadResponse(), 426



release(), 293  
 requestLocationUpdates(), 317  
 Resources.getXml(), 125  
 Resources.openRawResource(), 125  
 run(), 110  
 saveWidgetData(), 519  
 sendBroadcast(), 142  
 sendDataMessage(), 231  
 sendEmptyMessage(), 110  
 sendEmptyMessageAtTime(), 110  
 sendEmptyMessageDelayed(), 110  
 sendMessage(), 110  
 sendMultipartTextMessage(), 231  
 sendOrderedBroadcast(), 142  
 sendStickyBroadcast(), 142  
 sendTextMessage(), 231  
 set(), 247  
 setAlarm(), 534  
 setContentView(), 54  
 setContentView(), 95, 524  
 setInt(), 525  
 setListAdapter(), 109  
 setOnClickListener(), 525  
 setOnClickPendingIntent(), 525  
 setOnFocusChangeListener(), 115  
 setRepeating(), 247  
 setResult(), 47  
 setter(), 574  
 setText(), 78  
 setTimeZone(), 247  
 setVideoSize(), 297  
 start(), 284, 293  
 startActivity(), 44  
 startActivityForResult(), 44  
 startService(), 46  
 stop(), 293  
 stopSelf(), 536  
 System.loadLibrary(), 566  
 tap(), 324  
 toString(), 546  
 touch(), 324  
 transaction(), 482
 

- funkcja wywoływana po wykonaniu instrukcji SQL, 483
- parametryzowana instrukcja SQL, 482
- procedura obsługi błędów, 484
- tablica obiektów JavaScript, 482

 updateMotors(), 424  
 updateOneSite(), 537  
 UpdateOneWidget(), 525  
 validate(), 99

View.requestFocus(), 115  
 metody
 

- cyklu życia aktywności, 101
- cyklu życia aplikacji, 95
- cyklu życia widoku, 112
- lokalizacji, 304
- statyczne klasy SiteMonitorModel, 517
- uzyskania referencji obiektów
  - LocationProvider, 314
  - w API klasy bazowej View, 105
  - w klasie TextView, 105
  - wywołania zwrotnego, 523

 Microsoft Exchange, 34  
 Mobile Equipment Identity (MEID), 220  
 mobilne aplikacje WWW, 485  
 moduł rozszerzający Android Development Tools (ADT), 60  
 modyfikowanie danych kontaktowych, 444  
 MP3, Moving Picture Experts Group Audio Layer 3, 282  
 MPEG-4, Moving Picture Experts Group 4, 282  
 MXL, 209

## N

nadmiarowe instrukcje, 596  
 nadpisywanie funkcji przeglądarki, 487  
 nagłówek content-type, 206  
 NANP, North American Numbering Plan, 229  
 narzędzia wiersza poleceń Codesourcery, 381  
 narzędzie
 

- adb, 382, 589, 607
- Android Asset Packaging Tool, 69
- arm-none-linux-gnueabi-objdump, 391
- Error Console, 485
- GUI Layout, 76
- mkshcard, 171
- objdump, 379
- telnet, 48, 71, 307
- Web Inspector, 485

 nasłuch, 105  
 nasłuch zdarzeń, 116  
 nasłuch żądań intencji, 135  
 National Data Buoy Center (NDBC), 304  
 NDK, Android Native Development Kit, 378, 560–577  
 ndk builder, 577  
 NDK w Eclipse, 577  
 NTLM, Windows NT Lan Manager, 206

## O

## obiekt

- AccountManager, 455
- Activity, 41, 136, 232
- Adapter, 107
- Application, 499
- Binder, 449
- BluetoothDevice, 414
- Broadcast, 232
- BroadcastReceiver, 41, 136, 142, 532
- Bundle, 102, 143, 361
- Button, 55
- Canvas, 257
- DefaultHttpClient, 206
- Drawable, 266
- FileInputStream, 352, 372
- GeoPoint, 305, 326
- grabReviews, 95
- Handler, 107, 109, 119
- HandlerThread, 110
- Intent, 41
- IntentFilter, 48, 135, 136
- java.net, 200
- JNIEnv, 566
- JobList, 354
- JobListHandler, 352
- JsResult, 501
- LayoutParams, 113
- ListView, 106
  - właściwości, 108
- LocationListener, 317
- LocationManager, 141, 317
- LocationProvider, 317
- Looper, 110
- NetworkInfo, 195
- Notification, 245
- null, 494
- Observer, 116
- OnFocusChangeListener, 115
- Overlay, 304, 321, 326
- PendingIntent, 232, 314, 525
- PhoneStateListener, 218
- Prefs, 348, 368
- ProgressDialog, 357
- RemoteViews, 524
- ReviewListView, 111
- Runnable, 110
- SensorManager, 416
- Service, 41, 136, 232
- SharedPreferences, 160
- Spinner, 96
- Toast, 441
- Uri, 131
- View, 44
  - właściwości, 122
- ViewGroup, 113
- WebView, 501
  - WebViewClient, 496
- obiekt nasłuchu, 285, 317
- obiekt nasłuchu zdarzeń telefonicznych, 221
- obiekt obsługi JavaScript, 489
- obiekt uwierzytelniania, 206
- obsługa
  - błędu, 361
  - intencji, 43
  - interfejsu użytkownika, 43
  - kamery, 287
  - OpenGL, 267
  - transakcji wymiany danych, 335
  - widżetów zombie, 523
  - wielu kont, 433
  - zlecenia, 362
  - zadań intencji, 135
- odbieranie wiadomości SMS, 230, 233, 239
- odbiornik aktualizacji lokalizacji, 315
- odbiornik intencji, 240
- odbiornik SiteMonitorBootstrap, 539
- odecisk MD5 certyfikatu, 320
- odczyt wartości czujnika, 418
- oddzielanie literałów znakowych od kodu, 122
- odtworzenie przyrostowe, 282
- odtworzenie strumieniowe, 282
- odtworzenie wideo, 285
- odwołanie, 121
- odwołanie typu int, 122
- odwrotne geokodowanie, 325
- okno powłoki komputera Linux, 382
- okno wiersza poleceń, 382
- określanie pasujących akcji i kategorii, 136
- określanie pasujących intencji, 136
- opcja
  - c, 386
  - static, 381
  - Konta, 448
  - push, 383
  - Reset Perspective, 65
  - shell, 383
  - Show View, 65
  - viewport, 470
  - linkera, 389
  - wiersza poleceń, 84
  - znacznika meta viewport, 472

Open Headset Alliance, 32  
 OpenCORE, 282  
 OpenGL, 256  
 OpenGL ES, 256, 267, 268, 279  
 operacje CRUD, 185

## P

pakiet, 192  
 Android Native Development Kit (NDK), 378, 560–577  
 android.app, 44, 61  
 android.app.Service, 46  
 android.bluetooth, 61, 409  
 android.content, 61  
 android.gesture, 61  
 android.graphics, 61, 256  
 android.hardware, 416  
 android.location, 61  
 android.net, 61  
 android.opengl, 61  
 android.os, 61  
 android.provider, 62  
 android.telephony, 62, 222, 231  
 android.text, 62  
 android.util, 62  
 android.view, 62, 103  
 android.webkit, 62, 487, 488, 496  
 android.widget, 62, 112  
 com.google.android.maps, 319  
 com.msi.manning.nlockingandroid, 48  
 java.io, 61  
 java.lang, 61  
 java.math, 61  
 java.net, 61  
 java.text, 61  
 javax.net, 61  
 javax.security, 61  
 javax.xml, 61  
 kSOAP, 215  
 org.apache, 61  
 org.xml, 61  
 Sourcery G++, 379  
 pakiet graficzny, 256  
 panel Emulator Control, 305  
 parametr  
 Class, 132  
 ComponentName, 132  
 data, 232  
 deliveryIntent, 232  
 destinationAddress, 232  
 destinationPort, 232  
 scAddress, 232  
 sentIntent, 232  
 user agent, 473  
 wipe-data, 84  
 parowanie urządzenia, 412  
 parser SAX, 106, 352  
 parsowanie numerów telefonów, 228  
 PDU, 234, 241  
 PendingIntent, 232  
 perspektywa, 63  
 DDMS, 65, 225, 295, 305, 382  
 Debug, 86  
 Eclipse, 88  
 Java, 64  
 OpenGL, 276  
 pętla zwrotna, 71, 193, 197  
 Phone 7, 34  
 PID aplikacji, 66  
 platforma, 31  
 platforma LinkedIn, 430  
 platforma multimedialna, 282  
 plik  
 .aidl, 149  
 addjob.php, 374  
 android.jar, 44, 73, 589  
 Android.mk, 565  
 AndroidManifest.xml, 41, 46–47, 51, 72, 93, 126, 238, 293, 296, 339–340, 423, 516, 538, 597  
 arrays.xml, 124  
 BounceActivity.java, 264  
 BounceView.java, 266  
 ChapterTwo.java, 72  
 CloseJob.java, 339  
 closejob.php, 374, 375  
 colors.xml, 123  
 corestuff.css, 474  
 db.html, 480  
 db.js, 480  
 db.php, 374  
 debug.keystore, 599  
 dims.xml, 123  
 eclipse.exe, 585  
 export.php, 374  
 FieldService.java, 339, 343, 344  
 footer.php, 374  
 getjoblist.php, 374, 375  
 handheld-small.css, 475  
 header.php, 374  
 index.html, 489  
 index.php, 374

## plik

- jni.h, 566
- JobEntry.java, 339
- JobList.java, 339
- JobListHandler.java, 339
- libua2efindedges.so, 572
- main.xml, 54, 74, 516, 572
- make, 571
- manage.php, 374
- ManageJobs.java, 339
- monitor.xml, 516
- output.txt, 565
- Prefs.java, 339
- R.class, 75
- R.java, 55, 73, 338, 339, 554
- RefreshJobs.java, 339
- releasekey.keystore, 599
- review\_criteria.xml, 96
- savejob.php, 374
- screen.css, 475
- screenfonts.css, 475
- SDK Setup.exe, 587
- ShowJob.java, 339
- showjob.php, 374
- ShowSettings.java, 339
- Simple\_animation.xml, 262
- SiteMonitorBootstrap.java, 516, 532
- SiteMonitorConfigure.java, 516
- SiteMonitorModel.java, 516
- SiteMonitorService.java, 516
- sitemonitorwidget.xml, 516, 527
- SiteMonitorWidgetImpl.java, 516
- Splash.java, 339
- splash.xml, 341
- strings.xml, 56, 437, 516, 547, 551
- styles.xml, 124
- ua2efindedges.c, 567, 572
- updatejob.php, 374
- utils.php, 374
- xmldrawable.xml, 259
- plik deskryptora aplikacji, 128
- plik instalacji, 71
- plik nagłówkowy, 380
- plik wsadowy, 78, 387
- plik wynikowy, 79
- plik zasobu XML, 95
- pliki
  - AIDL, 148
  - APK, 600
  - class, 79
  - CSS, 473

- dex, 39, 79
- ELF, 391
- GPX, 307
- JAR, 437
- KML, 310
- PNG, 265
- XML, 79, 95
  - zasobów ze stylami, 124
- pobieranie napisu, 554
- podmenu Open Perspective, 86
- podpisywanie cyfrowe aplikacji, 599
- pola klasy Notification, 243
- pole
  - accuracy, 418
  - Latitude, 306
  - Longitude, 306
  - Sensor, 418
  - timestamp, 418
  - values, 418
- pole kategorii, 41
- polecenia powloki, 71
- połączenia Bluetooth, 410
- połączenie sieciowe
  - EDGE, 81
  - EVDO, 81
  - GPRS, 81
- połączenie szyfrowane SSL, 334
- położenie użytkownika na mapie, 322
- port 5037, 71
- port TCP 1024, 394
- porty, 194
  - dynamicznie przydzielane, 194
  - prywatne, 194
  - rejestrowane, 194
  - zarezerwowane, 194
- POSIX, Portable Operating System Interface for UNIX, 196
- powiadomienia, 243
  - debugowania, 596
  - na pasku stanu, 244
    - o alarmie na pasku stanu, 253
- powiadomienie Toast, 244, 494, 498
- POX, 209
- POX XML over HTTP, 191
- późne dołączanie, 43
- pracownik mobilny, 333, 334, 336
- procedura startowa, 390
- proces AIDL, 150
- program
  - aapt, 69, 118
  - adb, Android Debug Bridge, 69, 387, 399
  - AVD Manager, 81

- netstat, 71
- pocztowy POP3, 447
- SDK and AVD Manager, 81
- sqlite3, 179
- programowanie WWW, 466
- projekt
  - SiteMonitor, 508
  - linkedin-j, 437
  - open source PhoneGap, 494
- protokół
  - Atom Publishing (AtomPub), 212
  - bezprzewodowy Bluetooth, 409
  - Direct Command, 426
  - IP, 192
  - komunikacji z robotem, 425
- Prototype, 468
- przechwytywanie
  - alarmu, 249
  - mediów, 287
  - podpisu, 366
  - połączeń, 229
  - rozmów, 226
  - zdarzeń, 116, 370
  - zdarzeń Bluetooth, 414
- przeciążony konstruktor klasy, 42
- przepływ danych pomiędzy centralą a pracownikiem mobilnym, 335
- przepływy sterowania w aplikacji serwisowej, 337
- przesyłanie danych do serwera, 366
- przetwarzanie obrazu
  - interfejs użytkownika, 572
- przyciski ekranowe, 98
- przykładowe dane aplikacji, 596
- Pseudo Electronic Serial Number (pESN), 221
- publikowanie w Android Market, 602
- puste procesy, 53

## R

- referencja
  - do adaptera, 414
  - do AppWidgetManager, 526
  - do danych osoby, 442
  - do obiektu AlarmManager, 534
  - do obiektu Application, 577
  - do OutputStream, 370
  - do progresshandler, 359
  - do samej siebie (this), 99
- rejestr filtrów intencji, 135
- rejestracja nasłuchów, 317
- rejestracja zdarzeń czujnika, 419

- rekord kontaktu, 41
- rekord o stałej długości, 432
- rekord otwarty, 431
- relacje między aktywnościami, widokami, zasobami, 92
- repozytorium, 447
- REST, 209, 211
  - POST, 207, 212
  - PUT, 212
  - GET, 212
  - DELETE, 212
- REST, Representational State Transfer, 191
- RFC, Requests for Comments, 199
- RFCOMM, 409
- robot LEGO Mindstorms NXT, 420, 426
- routing pakietów, 193
- rozgłaszanie akcji, 143
- rozgłaszanie intencji, 142
- rozmiary widoku, 114
- rozmiary aktualne widoku, 114
- rozpoznawanie intencji, 137
- rozszerzanie klasy WebViewClient, 500
- rozszerzanie przeglądarki, 496
- rysowanie przy użyciu XML, 257

## S

- schemat, nazwa i ścieżka w adresach URI, 136
- selektywne ładowanie treści, 472
- Selman Daniel, Java 3D Programming, 279
- Sen Robi, 18
- SenseBot
  - interfejs użytkownika, 420
- Serial Port Profile (SPP), 409
- serwer, 194, 197
- serwer czasu, 378
- serwer DayTime, 393, 403
- serwer korzystający z gniazd TCP, 394
- sieci telefoniczne, 218
- silnik JavaScript, 467
- silnik przeglądarki WebKit, 33, 466
- skalowanie, 472
- skalowanie widoku, 126
- skrypt budowania, 399
- skrypt budowania aplikacji, 384
- skrypt ndk-build, 572
- skrzynka SMS, 247
- smartfon, 32, 430
- SmartPhone, 34
- SMS, Short Message Service, 230
- SMTP, Simple Mail Transfer Protocol, 193

SOAP, Simple Object Access Protocol, 191, 209, 215–216, 335  
 specyfikacja 3GPP, 3rd Generation Partnership Project, 234, 282  
 sprawdzanie połączenia sieciowego, 195  
 SQLite, 174, 188  
 sqlite3, 179  
 Stagefright, 283  
 stała  
   Context.MODE\_PRIVATE (), 163  
   Context.MODE\_WORLD\_READABLE (), 163  
   Context.MODE\_WORLD\_WRITEABLE (), 163  
   FILL\_PARENT, 114  
   MIME\_TYPE, 437  
   UPDATE\_FREQUENCY, 533  
   WRAP\_CONTENT, 114  
 stan odłączenia, 422  
 stan procesów, 100  
 stan telefonu, 224  
 standard pkzip, 79  
 sterowanie robotem, 423  
 stos, 36, 37  
 stos TCP/IP, 194  
 struktura drzewiasta, 96  
 styl lokalny, 124  
 Symbian, 34  
 symbol @, 51, 56  
 symulator, 81  
 synchronizacja, 447, 458, 465  
 synchronizacja i współdzielenie danych, 404  
 synchronizowanie kont, 458  
 system centralny, 333  
 system komunikacji międzyprocesowej (IPC), 128  
 system OpenCORE, 282

## Ś

ścieżka logiczna android\_asset, 489  
 środowisko  
   emulowane, 67  
   programistyczne, 584

## T

tabela raw\_contacts, 434  
 tabela tbl\_jobs, 373  
 TCP, 193  
 TCP/IP, Transmission Control Protocol/Internet Protocol, 192  
 TDMA, Time Division Multiple Access, 219  
 technologia zarządzania bazami danych, 477

telefonía, 218  
 telefony  
   Android, 218  
   CDMA, 232  
   GSM, 232  
 telnet, 48, 71, 307  
 tematy, 124  
 test głębi, 276  
 testowanie  
   aktywności, 598  
   aplikacji w emulatorze, 85  
   klienta DayTime, 403  
   regresyjne aplikacje, 598  
   wiadomości SMS, 48  
 tryb macierzy GL\_PROJECTION, 274  
 tryb mapy, 321  
 tryb projektowania, 79  
 tworzenie  
   aktywności, 93  
   animacji, 125, 261  
   animacji za pomocą kodu, 263  
   aplikacji multimedialnych, 301  
   aplikacji WWW, 468  
   dostawcy treści, 182  
   klasy Activity, 94  
   klucza, 599  
   konfiguracji uruchamiania emulatora, 82  
   konta LinkedIn, 450  
   kontekstu OpenGL, 268  
   nowego kontaktu, 441  
   nowego urządzenia AVD, 83  
   odbiornika, 143  
   prostokąta z wykorzystaniem figur prostych  
     OpenGL, 272  
   strumieni wejścia i wyjścia, 416  
   trójwymiarowych kształtów, 275  
   widżetu, 508  
   własnych widoków, 111  
 typ danych  
   GL\_BYTE, 274  
   GL\_FLOAT, 274  
   GL\_SHORT, 274  
 typ MIME, 136  
 typ tablicy GL\_VERTEX\_ARRAY, 274  
 typy alarmów, 250

## U

U.S. National Oceanic and Atmospheric Administration (NOAA), 304  
 uchwyt transakcji bazy danych, 483

UDP, User Datagram Protocol, 193  
 układ FrameLayout, 113  
 układ LinearLayout, 113  
 układ RelativeLayout, 113  
 układy, 113  
 umieszczanie aplikacji w Android Market, 603  
 URI, Uniform Resource Identifier, 40, 212  
 URL, Uniform Resource Locator, 40  
 uprawnienie android.permission  
   .CALL\_PHONE, 128, 224  
   .CALL\_PRIVILEGED, 224  
   .MODIFY\_PHONE\_STATE, 224  
   .PROCESS\_OUTGOING\_CALLS, 224  
   .READ\_PHONE\_STATE, 224  
   .READ\_SMS, 233  
   .RECEIVE\_SMS, 233  
   .SEND\_SMS, 233  
   .WRITE\_CONTACTS, 443  
   .WRITE\_SMS, 233  
 uprawnienie  
   AUTHENTICATE\_ACCOUNTS, 450  
   BLUETOOTH, 409, 416  
   CAMERA, 296  
   GET\_ACCOUNTS, 450  
   MANAGE\_ACCOUNTS, 450  
   READ\_PHONE\_STATE, 224  
   RECORD\_AUDIO, 296  
   USE\_CREDENTIALS, 450  
   WRITE\_EXTERNAL\_STORAGE, 296  
   WRITE\_SETTINGS, 450  
 urządzenie wirtualne Android, 306  
 urządzenia AVD, 83  
 usługa, 531  
   AccountManager, 449  
   prognozy pogody, 144  
   SiteMonitorService, 514, 534, 539  
   WeatherAlertService, 144  
 usługi, 52  
   cykl życia, 156  
   uruchamianie i dołączanie, 155  
   sieciowe, 190, 209  
   wiązania, 148  
 ustawienia regionalne, 543, 546  
 uwierzytelnianie  
   bazujące na formularzu, 206  
   konta, 449  
   podstawowe, 206  
   w LinkedIn, 451  
 użytkownik mobilny, 474

## V

VideoView, 285, 297

## W

warstwa, 192  
   abstrakcji, 36  
   aplikacji, 192  
   danych, 49  
   komunikacyjna, 128  
   połączenia, 192  
   sieci, 192  
   transportowa, 192  
 wartość identyfikatora zlecenia, 375  
 wątek, 109, 358  
 wątek główny interfejsu użytkownika, 109, 204  
 węzeł, 192  
 wideotelefony, 282  
 widok, 92, 103  
   cuisine, 95  
   Devices, 65  
   EditText, 54  
   EditView, 54  
   Emulator Control, 67  
   File Explorer, 67  
   grabReviews, 95  
   Javadoc, 64  
   LinearLayout, 55  
   location, 95  
   LogCat, 66, 77  
   MapView, 304, 319  
   Package Explorer, 64, 72  
   Problems, 64  
 widoki  
   podrzędne, 113  
   statyczne, 96  
   złożone, 112  
 widżet, 504  
 wiersz poleceń, 71  
 Wi-Fi, 190  
 WiMAX, Worldwide Interoperability for  
   Microwave Access, 190  
 Windows Mobile, 34  
 właściwości ListView, 108  
 właściwości View, 122  
 współrzędne geograficzne, 306  
 wstawianie danych do bazy danych, 394  
 wstawianie i usuwanie wierszy, 484  
 wyczerpywanie zasobów systemu, 52  
 wykonanie zdjęcia, 574  
 wykrywanie krawędzi, 560, 576

wykrywanie krawędzi metodą Sobela, 560

wymagania funkcjonalne, 336

wynik aktywności

    RESULT\_CANCELED, 530

    RESULT\_OK, 530

wysyłanie wiadomości SMS, 231

wywołania zwrotne cyklu życia, 100

wywołanie

    characters(), 354

    endElement(), 354

    Looper.loop(), 359

    Looper.prepare(), 359

    POST, 375

    setResult(), 366

    startElement(), 354

wywołanie aplikacji zewnętrznych, 134

wzorce instalowania widżetów, 508

## Z

zabezpieczenie CAPTCHA, 214

zamknięcie zlecenia, 370

zapis @id, 122

zapisywanie dźwięku, 292

zapisywanie mediów, 282

zapisywanie wideo, 295

zapytanie, 49

zapytanie media, 474

zarządzanie kontaktami, 438

zasady Android Market, 602

zasoby, 92, 117

    graficzne, 120

    grupa animacje, 123

    grupa kolory, 123

    grupa napisy, 123

    grupa style, 123

    grupa tablice, 123

    grupa wymiary, 123

    lokalne, 120

    res/anim, 117

    res/drawable, 117

    res/layout, 117

    res/raw, 118

    res/values, 118

    res/xml, 118

    XML, 169

zdarzenia systemowe, 45

zdarzenie, 116

    ACTION\_(ACL\_DISCONNECTED, 414

    ACTION\_ACL\_(DISCONNECTED, 409

    ACTION\_ACL\_CONNECTED, 409, 414

    ACTION\_DOWN, 370

    ACTION\_MOVE, 370

    ACTION\_UP, 370

    Aplikacja nie odpowiada (ANR), 109

    onCreate(), 116

    onFocusChanged(), 116

    onFreeze(), 116

    onLayout(), 116

    onSizeChanged(), 116

    onTouchEvent(), 116

    setMessage(), 117

zdarzenie interfejsu, 117

zdarzenie rozłączenia, 422

zdefiniowane urządzenia AVD, 83

ZigBee, 190

złączenie widoków, 96

zmienna

    \$data, 375

    AuthToken, 453

    browser, 488

    java.util.Date, 519

    LOCAL\_LIBS, 572

zmienne typu wyliczeniowego (enum), 266

zmniejszanie liczby procesów, 99

znacznik

    <intent-filter>, 41, 48, 51

    <joblist>, 375

    <receiver>, 46, 51

    <service>, 51

    <uses-permission>, 48, 539

    <uses-permissions>, 51

    corners, 260

    Drawable, 323

    ImageView, 262

    meta, 470

    meta viewport, 492

    padding, 260

    stroke, 260

    viewport, 470, 471, 474, 477

związanie kamery emulatora z widokiem, 288

## Ż

żądanie

    ACTION\_PICK, 438

    HTTP GET, 199

    HTTP POST, 201

    klucza dla API LinkedIn, 437

żeton uwierzytelniania, 450, 455



## Wkrocz wreszcie do akcji i zacznij tworzyć własne aplikacje dla Androida!

Skoro zwróciłeś uwagę właśnie na tę książkę, zapewne dobrze wiesz, czym jest Android i co potrafi — teraz przyszła pora, abyś sprawdził go także w akcji! Oto doskonała propozycja dla wszystkich programistów, którym marzy się tworzenie własnych aplikacji dla robiącego oszałamiającą karierę systemu. Choć ta książka nie jest przeznaczona dla początkujących, zawiera wszystkie informacje potrzebne osobom, dla których Android to całkowicie nowe środowisko. Można w niej znaleźć instrukcje niezbędne do szybkiego zorientowania się w architekturze tej platformy oraz sposobie jej działania, co pozwoli sprawnie rozpocząć pracę w tym środowisku. Pozostałe rozdziały to już czysta frajda programowania!

Od czego zaczniesz zabawę z Androidem? Dowiesz się, jak budować aplikacje dla tego systemu od najmniejszych cegiełek aż po ekran, dodawać funkcje telefoniczne i wykorzystywać bibliotekę OpenGL ES do tworzenia złożonej grafiki 2D oraz 3D. Następnie poznasz zasady tworzenia większych aplikacji oraz techniki pisania aplikacji w języku C, także z użyciem Android Native Development Kit. Opanujesz potężne narzędzie Android SDK oraz budowanie aplikacji dla WebKit z wykorzystaniem HTML 5, a nawet nauczysz się rozszerzać lub zastępować wbudowane funkcje Androida na podstawie użytecznych i intrygujących przykładów.

- Wprowadzenie do systemu Android
- Środowisko programowania
- Komponenty interfejsu użytkownika, w tym View i Layout
- Metody przechowywania i odczytywania lokalnych danych
- Sieci oraz usługi sieciowe
- Przegląd podstawowych zagadnień związanych z telefonią
- Powiadomienia i alarmy
- Grafika i animacja w Androidzie
- Korzystanie z funkcji multimedialnych Androida
- Usługi oparte na lokalizacji
- Integracja kontaktów z platformy społecznościowej
- Aplikacja wspomagająca serwisantów
- Budowanie aplikacji na Androida w języku C
- Bluetooth, sensory i widżety aplikacji
- Tworzenie stron WWW dla systemu Android
- Strategie lokalizowania aplikacji
- Android Native Development Kit
- Korzystanie z SDK oraz AVD Manager

nr katalogowy 7122

 **Katalogia Internetowa:**  
<http://helson.pl>

 **Zamówienia telefonicznie:**  
**0 801 339900**  
 **0 601 339900**

**helson.pl**  
katalogia  
internetowa

Sprawdź najnowsze promocje:  
• <http://helson.pl/promocje>  
Książki najchętniej czytane:  
• <http://helson.pl/bestsellery>  
Zamów informacje o nowościach:  
• <http://helson.pl/newsy>



**Helion**

ul. Rydyżowska 14, 44-100 Gliwice  
tel.: 32 230 95 43  
e-mail: [helson@helson.pl](mailto:helson@helson.pl)  
<http://helson.pl>

sklepij po WIECEJ!



NEO KODZYŚCI

Cena 99,00 zł

ISBN 978-83-246-3380-7



9 788324 633807

Informatyka w najlepszym wydaniu