

Charlie Collins, Michael Galpin, Matthias Kaeppler

Android w praktyce



Najlepsze techniki programowania
na Androida w zasięgu ręki!



Tytuł oryginału: Android in Practice

Tłumaczenie: Tomasz Walczak

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

ISBN: 978-83-246-4810-8

Original edition copyright © 2012 by Manning Publications Co.
All rights reserved.

Polish edition copyright © 2012 by HELION SA.
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie?gimpbi>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

<i>Wstęp</i>	11
<i>Podziękowania</i>	13
<i>O książce</i>	17
<i>O ilustracji z okładki</i>	23

CZĘŚĆ I TŁO HISTORYCZNE I PODSTAWY25

1	<i>Wprowadzenie do Androida</i>	27
1.1.	Android w pigułce	30
1.2.	HelloAndroid	34
1.3.	Java, ale nie do końca	45
1.4.	Linux, ale nie do końca	51
1.5.	Więcej możliwości dzięki bibliotekom natywnym	56
1.6.	Potrzebne narzędzia	59
1.7.	Podsumowanie	67
2	<i>Podstawy tworzenia aplikacji na Android</i>	69
2.1.	Aplikacja DealDroid	70
2.2.	Podstawowe cegielki	72
2.3.	Manifest aplikacji	74
2.4.	Zasoby	76
2.5.	Układ, widoki i kontrolki	80
2.6.	Aktywności	82
2.7.	Adaptery	91
2.8.	Intencje i filtry intencji	96
2.9.	Obiekty klasy Application	103
2.10.	Podsumowanie	105
3	<i>Zarządzanie cyklem życia i stanem</i>	107
3.1.	Czym są aplikacje w Androidzie?	108
3.2.	Cykl życia aktywności	113

- 3.3. Kontrolowanie stanu egzemplarza aktywności 125
- 3.4. Wykonywanie operacji za pomocą zadań 131
- 3.5. Podsumowanie 133

CZĘŚĆ II PRAKTYCZNE ROZWIĄZANIA 135

4 *Precyzja co do piksela* 137

- 4.1. Aplikacja MyMovies 138
- 4.2. Hierarchie widoków i ich wyświetlanie 139
- 4.3. Porządkowanie widoków w układy 143
 - TECHNIKA 1. Dyrektywy scalania i dołączania 152
- 4.4. Rozwinięcie informacji o klasach ListView i Adapter 156
 - TECHNIKA 2. Zarządzanie listą z pamięcią stanu 156
 - TECHNIKA 3. Widoki nagłówka i stopki 161
- 4.5. Stosowanie motywów i stylów 165
 - TECHNIKA 4. Stosowanie i pisanie stylów 165
 - TECHNIKA 5. Stosowanie i pisanie motywów 167
 - TECHNIKA 6. Określanie stylu tła widoku ListView 170
- 4.6. Korzystanie z obiektów graficznych 174
 - TECHNIKA 7. Używanie obiektów graficznych w postaci kształtów 175
 - TECHNIKA 8. Stosowanie selektorów obiektów graficznych 179
 - TECHNIKA 9. Skalowanie widoków za pomocą dziewięciopolowych obiektów graficznych 182
- 4.7. Tworzenie przenośnych interfejsów użytkownika 186
 - TECHNIKA 10. Automatyczne dostosowywanie aplikacji do różnych ekranów 186
 - TECHNIKA 11. Wczytywanie zasobów zależnych od konfiguracji 191
 - TECHNIKA 12. Uniezależnienie się od pikseli 194
- 4.8. Podsumowanie 196

5 *Używanie usług do zarządzania zadaniami wykonywanymi w tle* 199

- 5.1. Wielozadaniowość jest najważniejsza 200
- 5.2. Do czego służą usługi i jak z nich korzystać? 201
 - TECHNIKA 13. Tworzenie usługi 202
 - TECHNIKA 14. Automatyczne uruchamianie usługi 206

- TECHNIKA 15. Komunikowanie się z usługą 208
- TECHNIKA 16. Wykorzystanie usługi do zapisywania danych w pamięci podręcznej 214
- TECHNIKA 17. Tworzenie powiadomień 217
- 5.3. Planowanie i usługi 222
 - TECHNIKA 18. Używanie klasy AlarmManager 222
 - TECHNIKA 19. Podtrzymywanie działania usługi 226
 - TECHNIKA 20. Używanie usługi Cloud to Device Messaging 229
- 5.4. Podsumowanie 234

6 Wątki i współbieżność 237

- 6.1. Współbieżność w Androidzie 238
 - TECHNIKA 21. Proste wątki 240
 - TECHNIKA 22. Przekazywanie informacji o zmianach między wątkami 243
 - TECHNIKA 23. Zarządzanie wątkami w puli wątków 249
- 6.2. Korzystanie z klasy AsyncTask 255
 - TECHNIKA 24. Implementowanie prac za pomocą klasy AsyncTask 256
 - TECHNIKA 25. Przygotowanie do zmian w konfiguracji 261
- 6.3. Różne techniki 268
 - TECHNIKA 26. Wyświetlanie ekranów powitalnych za pomocą zegarów 268
 - TECHNIKA 27. Implementowanie niestandardowych pętli komunikatów 272
- 6.4. Podsumowanie 276

7 Lokalne zapisywanie danych 279

- 7.1. Odczyt i zapis plików 280
 - TECHNIKA 28. Korzystanie z pamięci wewnętrznej 282
 - TECHNIKA 29. Korzystanie z pamięci zewnętrznej 286
 - TECHNIKA 30. Używanie katalogów na pamięć podręczną 292
 - TECHNIKA 31. Stosowanie synchronizacji przy zapisie plików 293
- 7.2. Przechowywanie ustawień 294
 - TECHNIKA 32. Odczyt i zapis ustawień 295

- TECHNIKA 33. Korzystanie z klasy PreferenceActivity 296
- 7.3. Korzystanie z bazy danych 299
 - TECHNIKA 34. Tworzenie bazy danych i obiektów modelu 303
 - TECHNIKA 35. Tworzenie obiektów DAO i menedżera danych 312
- 7.4. Badanie baz SQLite 323
- 7.5. Podsumowanie 325

8 *Współużytkowanie danych między aplikacjami* 327

- 8.1. Współużytkowanie danych między procesami 328
 - TECHNIKA 36. Stosowanie intencji 329
 - TECHNIKA 37. Zdalne wywołania procedur 335
 - TECHNIKA 38. Współużytkowanie danych (i innych elementów) przez współdzielenie kontekstu 341
- 8.2. Dostęp do niestandardowych danych 347
 - TECHNIKA 39. Korzystanie ze standardowych dostawców treści 347
 - TECHNIKA 40. Korzystanie z niestandardowego dostawcy treści 352
- 8.3. Podsumowanie 356

9 *Protokół HTTP i usługi sieciowe* 357

- 9.1. Podstawy pracy z siecią z wykorzystaniem protokołu HTTP 358
 - TECHNIKA 41. Protokół HTTP i klasa HttpURLConnection 360
 - TECHNIKA 42. Praca z protokołem HTTP za pomocą klasy HttpClient Apache'a 366
 - TECHNIKA 43. Konfigurowanie obiektu klasy HttpClient bezpiecznego ze względu na wątki 370
- 9.2. Korzystanie z usług sieciowych generujących dane w formatach XML i JSON 375
 - TECHNIKA 44. Przetwarzanie danych w XML-u za pomocą interfejsu SAX 379
 - TECHNIKA 45. Przetwarzanie dokumentów XML na podstawie specyfikacji XmlPull 385
 - TECHNIKA 46. Przetwarzanie danych w formacie JSON 389

- 9.3. Elegancka obsługa awarii sieci 393
 - TECHNIKA 47. Ponawianie żądań za pomocą komponentów obsługi 393
 - TECHNIKA 48. Obsługa zmian konfiguracji sieci 397
- 9.4. Podsumowanie 400

10 *Najważniejsza jest lokalizacja* 403

- 10.1. Krótkie wprowadzenie do współrzędnych geograficznych 404
- 10.2. Menedżery, dostawcy i odbiorniki położenia 407
 - TECHNIKA 49. Sprawdzanie stanu dostawcy położenia 414
 - TECHNIKA 50. Określanie aktualnego położenia za pomocą odbiornika LocationListener 416
- 10.3. Tworzenie aplikacji z wykorzystaniem map 422
 - TECHNIKA 51. Przekształcanie adresu na współrzędne geograficzne 425
 - TECHNIKA 52. Tworzenie aktywności MapActivity z powiązaniem widokiem MapView 427
 - TECHNIKA 53. Wyświetlanie elementów OverlayItems w widoku MapView 430
- 10.4. Podsumowanie 433

11 *Uatrakcyjnianie aplikacji za pomocą multimedialnych* 435

- 11.1. Funkcje zbyt zaawansowane dla telefonu wielofunkcyjnego 436
 - TECHNIKA 54. Wykrywanie możliwości 437
- 11.2. Zarządzanie multimediami 440
 - TECHNIKA 55. Korzystanie z zasobów i plików 440
 - TECHNIKA 56. Korzystanie z dostawców treści multimedialnych 447
 - TECHNIKA 57. Używanie intencji i aktywności 450
- 11.3. Odtwarzanie multimedialnych 453
 - TECHNIKA 58. Zdjęcia i proste animacje 454
 - TECHNIKA 59. Kontrolowanie dźwięku 458
 - TECHNIKA 60. Wyświetlanie filmów 462
- 11.4. Rejestrowanie multimedialnych 465
 - TECHNIKA 61. Robienie zdjęć 465
 - TECHNIKA 62. Rejestrowanie dźwięku i filmów 470
- 11.5. Podsumowanie 475

12 Grafika dwu- i trójwymiarowa 477

- 12.1. Rysowanie z wykorzystaniem bibliotek do obsługi grafiki dwuwymiarowej 478
 - TECHNIKA 63. Przechodzenie do trybu pełnoekranowego 480
 - TECHNIKA 64. Rysowanie prostych kształtów 481
 - TECHNIKA 65. Ciągłe wyświetlanie widoku w wątku interfejsu użytkownika 484
 - TECHNIKA 66. Wyświetlanie tekstu na ekranie 485
 - TECHNIKA 67. Określanie czcionki przy wyświetlaniu tekstu 487
 - TECHNIKA 68. Wyświetlanie bitmap 489
 - TECHNIKA 69. Stosowanie efektów dwuwymiarowych 490
- 12.2. Grafika trójwymiarowa i biblioteka OpenGL ES 493
 - TECHNIKA 70. Rysowanie pierwszego trójkąta 500
 - TECHNIKA 71. Tworzenie piramidy 504
 - TECHNIKA 72. Kolorowanie piramidy 510
 - TECHNIKA 73. Dodawanie tekstury do piramid 513
- 12.3. Podsumowanie 519

CZĘŚĆ III POZA STANDARDOWE ROZWIĄZANIA 521

13 Testowanie i instrumentacja 523

- 13.1 Testowanie aplikacji na Android 525
 - TECHNIKA 74. Prosty test jednostkowy aplikacji na Android 533
- 13.2. Pociąganie za sznurki — instrumentacja w Androidzie 538
 - TECHNIKA 75. Testy jednostkowe aktywności 539
 - TECHNIKA 76. Scenariusz użytkownika jako testy funkcjonalne 544
 - TECHNIKA 77. Eleganckie testy z wykorzystaniem frameworku Robotium 549
- 13.3. Poza instrumentację — atrapy i testy losowe 554
 - TECHNIKA 78. Atrapy i sposoby ich stosowania 554
 - TECHNIKA 79. Przyspieszanie testów jednostkowych z zastosowaniem Robolectrica 561
 - TECHNIKA 80. Przeprowadzanie testów obciążeniowych za pomocą narzędzia Monkey 567
- 13.4. Podsumowanie 573

14 Zarządzanie budowaniem 575

- 14.1. Budowanie aplikacji na Android 577
 - TECHNIKA 81. Budowanie aplikacji za pomocą Anta 583
- 14.2. Zarządzanie procesem budowania za pomocą Mavena 592
 - TECHNIKA 82. Budowanie za pomocą Mavena 595
 - TECHNIKA 83. Wtyczka Mavena dla środowiska Eclipse 607
 - TECHNIKA 84. Narzędzie maven-android-sdk-deployer 610
- 14.3. Serwery budowania i ciągłe budowanie 615
 - TECHNIKA 85. Ciągłe budowanie z wykorzystaniem Hudsona 617
 - TECHNIKA 86. Budowanie macierzowe 625
- 14.4. Podsumowanie 630

15 Pisanie aplikacji na tablety z Androidem 633

- 15.1. Przygotowania do tworzenia aplikacji na tablety 635
 - TECHNIKA 87. Wykorzystywanie istniejącego kodu za pomocą projektów bibliotek 635
 - TECHNIKA 88. Tworzenie aplikacji przeznaczonej na tablety 638
- 15.2. Podstawowe informacje o tabletach 641
 - TECHNIKA 89. Fragmenty 642
 - TECHNIKA 90. Pasek akcji 650
 - TECHNIKA 91. Przeciąganie 655
- 15.3. Podsumowanie 662

Dodatek A Narzędzia do debugowania 665

Dodatek B Niestandardowe techniki tworzenia aplikacji na Android 677

Dodatek C ProGuard 687

Dodatek D Monkeyrunner 701

Skorowidz 713

Pisanie aplikacji na tablety z Androidem

W tym rozdziale

- Stosowanie fragmentów
- Pasek akcji
- Implementowanie przeciągania

Wszystko staje się coraz większe. Dlatego obecnie należy pisać programy w bardziej wyrafinowany sposób.

Bill Budge

Był rok 2001. Microsoft, największa firma technologiczna na świecie, zaprezentował przełomową wersję niezwykle popularnego systemu operacyjnego — Windows XP Tablet PC Edition. Zdaniem Microsoftu miał to być początek ery urządzeń dotykowych. Wiemy, jak to się skończyło. System XP Tablet PC Edition okazał się niewypałem.

Tak naprawdę komputery z systemem XP Tablet nie były pierwszymi urządzeniami z wyświetlaczem dotykowym przeznaczonymi na rynek masowy. Dziesięć lat wcześniej inżynierowie z Apple'a opracowali prototyp, który w przekształconej postaci wprowadzono na rynek jako komputer Newton. Opracowane 10 lat później przez Microsoft komputery Tablet PC były podejrzanie podobne do Newtona. Jednak prototypowa wersja Newtona nigdy nie trafiła do sprzedaży; komputery z tej rodziny stały się za to poprzednikami palmtopów.

Stwierdzenie, że urządzenia dotykowe przez wiele lat były ogłaszane jako następny wielki hit, to poważne niedomówienie. Można uznać, że wcześniejsze

próby zakończyły się niepowodzeniem, ponieważ producenci próbowali przekształcić komputery PC w tablety. Dwadzieścia lat po opracowaniu prototypowego tabletu Newton stało się oczywiste, że naturalnym urządzeniem dotykowym są smartfony i to na nich należy wzorować tablety. W ten sposób powstały iPad Apple'a i Android dla tabletów. W wersji Android 3.0 (Honeycomb) wprowadzono istotne zmiany, opracowane z myślą o tabletach. Co to oznacza dla programistów aplikacji na Android?

Jeśli już udostępniasz w sklepie Android Market kilka aplikacji, być może nasuwa Ci się pytanie: „Co powinienem zrobić, aby programy działały na tabletach?”. Liczba użytkowników tabletów jest znacznie mniejsza niż liczba osób korzystających z telefonów. Zyskanie popularności przez nowe urządzenia, jakimi są tablety, wymaga czasu, a użytkowników smartfonów są miliony. Prawdopodobnie dużo większe zyski przyniesie inwestowanie w rozwijanie aplikacji na Android przeznaczonych na smartfony, a nie na tablety. Jest to prawdą zwłaszcza w przypadku nowych aplikacji. Tworząc programy na smartfony, dotrzesz do znacznie większej grupy odbiorców.

Istnieją jednak wyjątki od tej reguły. Niektóre rodzaje aplikacji lepiej nadają się dla tabletów niż dla smartfonów. Na przykład czytniki wiadomości, aplikacje z sieci społecznościowych, sklepy internetowe czy inne programy wyświetlające bogate materiały wyglądają znacznie lepiej na większym ekranie. Na smartfonie powiązane informacje trzeba wyświetlać na dwóch lub trzech ekranach, a na tablecie można je zaprezentować na jednej angażującej czytelnika stronie. Nawet w innych obszarach wczesne wprowadzanie aplikacji na tablety do sklepu Android Market przynosi poważne korzyści. Program może zyskać popularność z uwagi na niewielką konkurencję, co zapewnia dobrą pozycję wyjściową, kiedy więcej osób zacznie kupować tablety i szukać ciekawych aplikacji w sklepie.

Niezależnie od tego, czy lubisz nowinki, czy nie, w pewnym momencie zechcesz utworzyć aplikację na tablety (w przeciwnym razie prawdopodobnie nie czytałbyś tego rozdziału). W tym rozdziale wzbogacamy aplikację DealDroid z rozdziału 2. i tworzymy jej wersję na tablety. Gotowy produkt przedstawiono na rysunku 15.1.

Decyzji o utworzeniu aplikacji na tablety nie podejmuje się stopniowo. Trzeba od początku stwierdzić, że program ma działać w tabletach. Rozwijanie takich aplikacji jest ekscytujące, ponieważ tablety w porównaniu ze smartfonami otwierają przed programistami wiele nowych możliwości. Liczne typowe problemy, na przykład mały ekran, niewielka ilość pamięci, konieczność obsługi dawnych wersji Androida i wolna sieć, w tabletach są mniej dotkliwe. Jednak zanim zaczniemy tworzyć fragmenty pełne widoków `StackView`, warto zastanowić się, jakie elementy aplikacji są już gotowe i jak je wykorzystać.



Rysunek 15.1.
Aplikacja DealDroid
w wersji na tablety

15.1. Przygotowania do tworzenia aplikacji na tablety

Pisanie programów na tablety nie polega tylko na stosowaniu nowych interfejsów API i większych grafik. Musisz zdecydować, czy chcesz utworzyć odrębną aplikację, czy rozwinąć istniejący program na smartfony, tak aby działał poprawnie także na tabletach. W tym rozdziale koncentrujemy się na pisaniu nowych aplikacji. Pozwala to wykorzystać wszystkie możliwości Androida 3.0. Wszystkie (lub prawie wszystkie) techniki z tego rozdziału można zastosować w czasie tworzenia standardowych aplikacji.

TECHNIKA 87. Wykorzystywanie istniejącego kodu za pomocą projektów bibliotek

Choć tworzymy odrębną aplikację przeznaczoną na tablety z Androidem, nie oznacza to, że program jest zupełnie odmienny od rozwiązań tworzonych na smartfony. Obie wersje aplikacji mają wiele wspólnych funkcji i mogą korzystać nawet z tych samych danych. Dane te można przechowywać lokalnie w urządzeniu lub w chmurze na serwerze. Sposób dostępu do takich danych, a nawet porządkowania ich po pobraniu do pamięci lokalnej jest taki sam jak w smartfonach. Na szczęście istnieje dobry sposób na współużytkowanie kodu przez różne aplikacje na Android.

PROBLEM

W nowej aplikacji na tablety zamierzamy wykorzystać kod istniejącego programu na smartfony. Chcemy, aby istniała tylko jedna kopia kodu. Pozwala to dodawać nowe funkcje, naprawiać błędy i wykonywać podobne zadania w jednym miejscu.

ROZWIĄZANIE

Sposobem na współużytkowanie kodu między aplikacjami na Android jest zastosowanie projektów bibliotek dla Androida. Takie projekty dobrze nadają się do porządkowania kodu i wprowadzono je w tym samym czasie co Android 2.2. Projekty bibliotek nie są przeznaczone tylko dla tabletów i Androida 3.0 —

pozwalają współużytkować kod także między różnymi aplikacjami na smartfony. Taki projekt jest bardzo przydatny przy tworzeniu aplikacji, która ma działać na smartfonach i tabletach.

Wspomnieliśmy już, że projekty bibliotek wprowadzono w tym samym okresie co Android 2.2. Dostosowano je jednak do jeszcze wcześniejszych wersji Androida. Możliwe, że przechowujesz kod w takich bibliotekach i możesz natychmiast wykorzystać je do tworzenia aplikacji na tablety na podstawie istniejącego kodu programów na smartfony. Jeśli tak jest, możesz przejść bezpośrednio do następnej techniki. Jeżeli jednak jeszcze nie stosujesz projektów bibliotek, czeka Cię refaktoryzacja i zmiana uporządkowania kodu.

W tej technice przekształcamy aplikację DealDroid przedstawioną po raz pierwszy w rozdziale 2. na program na tablety. Cały kod aplikacji DealDroid znajduje się w jednym projekcie aplikacji na Android. Musimy to zmienić. Na rysunku 15.2 pokazano nowy sposób uporządkowania kodu, pozwalający wykorzystać ten kod w nowym projekcie aplikacji na tablety.

Na rysunku 15.2 widoczne są projekty DealsLib i TabletDeals. DealsLib to projekt biblioteki dla Androida, obejmujący kod współużytkowany przez aplikacje na smartfony i tablety. TabletDeals to projekt aplikacji na tablety. W tym rozdziale opisujemy go bardzo szczegółowo. Na rysunku widać, jaki kod znajduje się w projekcie biblioteki. Umieszczono w niej (w pakiecie `com.manning.aip.dealdroid.xml`) cały kod do pobierania danych z internetu i ich przetwarzania. Kod ten może być taki sam dla tabletów i smartfonów. Dane w formacie XML są przetwarzane na obiekty modelu (określone w pakiecie `com.manning.aip.dealdroid.model`) używane przez aplikację. Także kod tych obiektów jest częścią biblioteki.

W pakiecie najwyższego poziomu (`com.manning.aip.dealdroid`) znajduje się kilka innych klas. Najciekawszą z nich jest `DealsApp`. Jest to klasa typu `Application` używana w obu aplikacjach. Obejmuje pamięć podręczną z danymi, a także stan aplikacji. Stan w aplikacjach na tablety mógłby być inny, jednak tu jest taki sam, dlatego wspomnianą klasę można współużytkować w wersjach programu na smartfony i tablety.

Ponadto między aplikacjami współużytkowane są niektóre zasoby, przede wszystkim pliki *strings.xml* i *plurals.xml*. Współużytkować można także inne zasoby, na przykład obiekty graficzne.

OMÓWIENIE

Projekty bibliotek dają duże możliwości. Programiści — zwłaszcza piszący pakiety aplikacji współużytkujących duże fragmenty kodu — dobitnie domagali się tego mechanizmu. Często wspólny kod służy do obsługi dostępu do sieci i modelu danych, podobnie jak w przykładzie. Nieraz wspólny jest też kod do uwierzytelniania użytkowników i późniejszego zarządzania informacjami o tożsamości (na przykład znacznikami uwierzytelniającymi i (lub) określającymi uprawnienia).



Rysunek 15.2. Kod przygotowany do współużytkowania z aplikacją na tablety

Tego rodzaju kod często obejmuje elementy interfejsu użytkownika, ponieważ programiści zwykle starają się ujednolicić sposób logowania się do aplikacji. Uzyskanie tego efektu nie jest trudne, gdyż w bibliotece można umieścić aktywności, XML-owy kod układu itd. Dla projektu biblioteki trzeba też utworzyć plik

AndroidManifest.xml. Można w nim deklarować aktywności, usługi i inne elementy — tak jak w innych manifestach. Jeśli jednak chcesz wykorzystać w aplikacji aktywność z projektu biblioteki (lub inny komponent deklarowany w manifestcie), musisz zadeklarować tę aktywność w pliku *AndroidManifest.xml* aplikacji. Manifest projektu biblioteki pełni funkcję menu dostępnych komponentów, które trzeba zadeklarować w pliku manifestu aplikacji.

Kod można współużytkować między projektami na różne sposoby. Jeśli korzystasz ze środowiska Eclipse (albo Anta, Mavena lub innego narzędzia zarządzającego zależnościami), możesz utworzyć własną bibliotekę z kolekcją kodu w Javie i wykorzystać ją w aplikacji. Zależność między biblioteką a aplikacją może występować na poziomie kodu źródłowego lub na poziomie binarnym. W tym drugim przypadku najpierw należy skompilować projekt biblioteki bądź nawet spakować go do archiwum JAR. Utrudnieniem jest wtedy tylko konieczność upewnienia się, że w kodzie biblioteki nie występują żadne standardowe klasy Javy niedozwolone w Androidzie, a także użycie odpowiedniego archiwum *android.jar*. Przy stosowaniu projektów bibliotek dla Androida zadania te są wykonywane automatycznie.

Inną ważną zaletą projektów bibliotek dla Androida w porównaniu ze standardowymi bibliotekami Javy jest obsługa zarządzania zasobami. W przykładzie w bibliotece umieszczamy standardowy plik *strings.xml*. Pozwala to współużytkować ten plik w wersjach na smartfony i tablety. Można też zastąpić konkretny łańcuch znaków lub dodać nowe fragmenty tekstu przez umieszczenie odrębnego pliku *strings.xml* w projekcie aplikacji. Kompilator scala wtedy zasoby. Dotyczy to także innych zasobów, na przykład stylów i obiektów graficznych, a nawet plików układu.

Teraz, kiedy znasz już dobry sposób na porządkowanie kodu i współużytkowanie go w aplikacjach na smartfony i tablety, można dokładniej zastanowić się nad wersją na tablety. Wspomnieliśmy już, że tworzymy aplikację przeznaczoną tylko na tablety. Nie zamierzamy rozwijać wersji działającej równie dobrze na smartfonach i tabletach. Na szczęście stosowanie naszego podejścia jest proste i przynosi duże korzyści.

TECHNIKA 88. Tworzenie aplikacji przeznaczonej na tablety

Programiści aplikacji na Android niechętnie mówili o zróżnicowaniu urządzeń (ang. *fragmentation*). Określenie to często stosowali przeciwnicy Androida, twierdzący, że zbyt trudno jest tworzyć aplikacje na tę platformę, ponieważ trzeba zapewnić obsługę urządzeń z ekranami o różnej wielkości i z innymi niejednołitymi cechami. Jednocześnie jest to jednak ukryta wartość Androida. Właściwy sposób programowania wymagał, aby nie robić założeń co do wielkości i proporcji ekranu. Programiści mieli wiele narzędzi do projektowania aplikacji z układem dostosowującym się do wyświetlacza. Kiedy więc pojawiały się nowe urządzenia z ekranami o przekątnej wynoszącej 4 lub 4,3 cala albo z mniejszymi,

2,5-calowymi wyświetlaczami, większość programów działała w nich prawidłowo. Nawet na pierwszych tabletach z 7-calowymi ekranami i Androidem 2.2 większość aplikacji funkcjonowała bez problemów (choć były też wyjątki od tej reguły — niektórzy programiści nie stosowali najlepszych praktyk i w czasie projektowania układów robili założenia dotyczące wielkości wyświetlacza). Było to wielką zaletą małych tabletów. Kiedy pojawiły się na rynku, od razu istniało wiele aplikacji, które prawidłowo na nich działały.

Jednak w czasie prac nad większymi tabletami okazało się, że opracowanie systemu operacyjnego pod kątem takich urządzeń ma duże zalety. Dlatego powstał Android 3.0. Platforma ta obejmuje elementy sprawiające, że aplikacje mogą działać równie dobrze zarówno na smartfonach, jak i na tabletach. Ponadto pozwala tworzyć atrakcyjne programy przeznaczone tylko na tablety. Wymaga to jednak zablokowania dostępu do aplikacji użytkownikom mniejszych urządzeń.

PROBLEM

Piszemy aplikację przeznaczoną tylko na tablety. Chcemy wykorzystać duży ekran i wszystkie możliwości platformy dostępne w tabletach. Nie zamierzamy dostosowywać aplikacji do urządzeń z mniejszymi ekranami, niezależnie od wersji Androida działającej na tych urządzeniach.

ROZWIĄZANIE

Możliwe, że już znasz rozwiązanie. W pliku *AndroidManifest.xml* należy określić wszystkie wymagania aplikacji. Następnie filtry w sklepie Android Market sprawia, że aplikacja nie będzie pojawiać się w urządzeniach innych niż tablety. Na listingu 15.1 znajduje się fragment manifestu pozwalający uzyskać ten efekt.

Listing 15.1. W manifeście można określić, że aplikacja jest przeznaczona tylko na tablety

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.manning.aip.tabdroid"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="11" /> ← ❶
    <supports-screens android:smallScreens="false"
        android:normalScreens="false"
        android:largeScreens="false"
        android:xlargeScreens="true" /> ← ❷
</manifest>
```

Aby aplikacja była przeznaczona tylko na tablety, w manifeście trzeba podać dwa podstawowe wymagania. Otóż w urządzeniu musi działać Android 3.0 (Honeycomb) lub nowsza wersja tej platformy ❶. Może się wydawać, że to wystarczy. W końcu jeśli dostępny jest interfejs API w wersji 11 (Android 3.0) lub nowszej, można korzystać ze wszystkich interfejsów API potrzebnych w rozwijanej aplikacji. Gdy powstawała ta książka, wersja Android 3.0 była najnowsza i działała tylko na tabletach, ale do czasu trafiaenia tej pozycji na półki pojawią się prawdopodobnie

nowsze wersje, które będą obejmować wszystkie funkcje Androida 3.0 i pracować zarówno na smartfonach, jak i na tabletach. Dlatego trzeba też określić, że aplikacja działa tylko na urządzeniach z ekranami xlarge ❷. Ten rozmiar wyświetlaczy wprowadzono w Androidzie 2.3. Odpowiada on ekranom mającym przynajmniej siedem cali. Po określeniu w manifestie tych dwóch wymagań można mieć pewność, że każde urządzenie, na którym uruchamiana jest aplikacja, to tablet ze zoptymalizowanymi pod jego kątem interfejsami API wprowadzonymi w wersji Honeycomb.

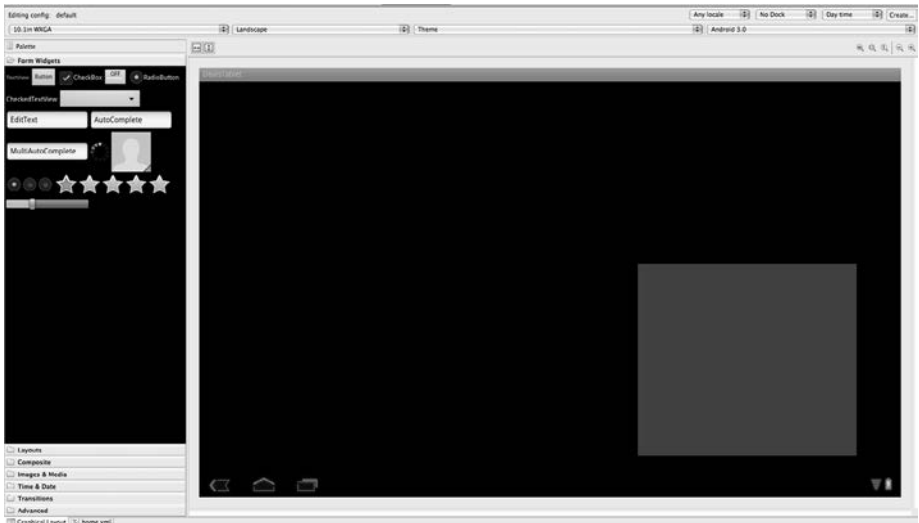
Należy wspomnieć o jeszcze jednym aspekcie programowania aplikacji na tablety. W czasie pisania aplikacji na smartfony programiści często zakładają, że urządzenie zwykle znajduje się w orientacji pionowej. Na szczęście system operacyjny dobrze obsługuje zmiany orientacji, dlatego nawet jeśli programista całkowicie zapomni o przygotowaniu wersji dla poziomego układu ekranu, aplikacja prawdopodobnie będzie działać poprawnie po obróceniu urządzenia. Warto jednak zastanowić się nad trybem poziomym, a czasem dobrze jest nawet przygotować dla niego odrębne układy. W Androidzie standardowo należy utworzyć katalog ze zoptymalizowanymi plikami XML z kodem układów. Inna możliwość to pominięcie orientacji poziomej i obsługiwanie tylko trybu pionowego. Ma to pewne zalety, choć użytkownicy urządzeń z wysuwanymi klawiaturami nie będą zadowoleni z aplikacji napisanej w ten sposób.

Tablety różnią się od smartfonów. Orientacja to jeden z obszarów, gdzie różnice między tymi typami urządzeń są duże. Z tabletów zwykle korzysta się w orientacji poziomej. Dlatego standardowo pliki układu dla tej orientacji umieszcza się w katalogu `/res/layout`, a pliki dla trybu pionowego — w katalogu `/res/layout-port`. Jeśli korzystasz z wtyczki ADT dla środowiska Eclipse, mechanizm tworzenia interfejsu użytkownika z tej wtyczki pomoże Ci w rozwijaniu aplikacji na tablety, co pokazano na rysunku 15.3.

OMÓWIENIE

Opisane tu podejście pod wieloma względami różni się od tworzenia typowych aplikacji na Android. Zwykle warto obsługiwać jak najwięcej różnych wyświetlaczy. Tu wykluczamy wszystkie wymiary oprócz jednego. Kiedy pojawiły się pierwsze tablety z wersją Honeycomb Androida, nie tylko miały podobne wymiary, ale też tę samą rozdzielczość ekranu. Było to coś nowego dla programistów aplikacji na Android, przyzwyczajonych do tworzenia rozwiązań z uwzględnieniem ekranów o różnej wielkości i rozdzielczości. Od czasów urządzeń G1 nie można było tworzyć programów dostosowanych do ekranu o konkretnych cechach (przyczyn fizyczne wymiary poszczególnych modeli tabletów były zróżnicowane). Unikaj jednak stosowania przestarzałych układów `AbsoluteLayout` lub podawania wymiarów w układzie za pomocą fizycznych pikseli.

Przedstawiliśmy projekty bibliotek, najnowsze interfejsy API i układy dla dużych ekranów. Pora rozpocząć tworzenie programów na tablety z Androidem.



Rysunek 15.3. Tworzenie interfejsu na tablety z wykorzystaniem wtyczki ADT

Zaczynamy od podstawowych technik, które powinien znać każdy programista aplikacji na tablety. Ponadto pokazujemy, że techniki te nie są ograniczone do tabletów i że można je łatwo wykorzystać także przy tworzeniu aplikacji na smartfony.

15.2. Podstawowe informacje o tabletach

Tablety z Androidem istniały już na długo przed pojawieniem się Androida 3.0. Miały ekrany o przekątnej od pięciu do siedmiu cali, były więc mniejsze niż pierwsze urządzenia z wersją Honeycomb. Takie miniaturkowe tablety były ciekawe same w sobie. Jak już wspomnieliśmy, większość aplikacji na Android działała w nich prawidłowo. Z uwagi na dodatkową przestrzeń niektóre takie programy wyglądały całkiem dobrze. Mimo to można było je opisać najwyżej jako poprawne. Jest to dowód na to, że Android potrafi dostosować aplikację do wyświetlacza bez nadmiernego utrudniania pracy użytkownikom.

Android 3.0 zaprojektowano tak, aby był więcej niż poprawny. Podejście nie polegało na dostosowaniu Androida do poprawnej pracy na większym ekranie lub na dodaniu nowych komponentów interfejsu użytkownika. Twórcy platformy wprowadzili poważne zmiany, aby pomóc programistom w skutecznym pisaniu aplikacji na urządzenia z większymi wyświetlaczami. Omawianie podstawowych technik tworzenia aplikacji na tablety zaczynamy od przyjrzenia się jednemu z najważniejszych mechanizmów wprowadzonych w wersji Honeycomb. Są nim fragmenty.

TECHNIKA 89. Fragmenty

Wspomnieliśmy już, że Android 3.0 zaprojektowano z myślą o tabletach. Opracowanie nowej wersji nie polegało na dodaniu nowych elementów do wcześniejszych odmian Androida. Jednym z najlepszych dowodów na to jest interfejs API fragmentów. Fragmenty umożliwiają porządkowanie kodu aplikacji w nowy sposób, znacznie ułatwiający radzenie sobie z tworzeniem układów dostosowanych do dużych ekranów tabletów z Androidem. Jednak mechanizm ten jest przydatny nie tylko w programach na tablety.

PROBLEM

Chcemy podzielić kod aplikacji na moduły, aby można było stosować zupełnie odmienne układy dla orientacji poziomej i pionowej bez konieczności powielania kodu oraz funkcji.

ROZWIĄZANIE

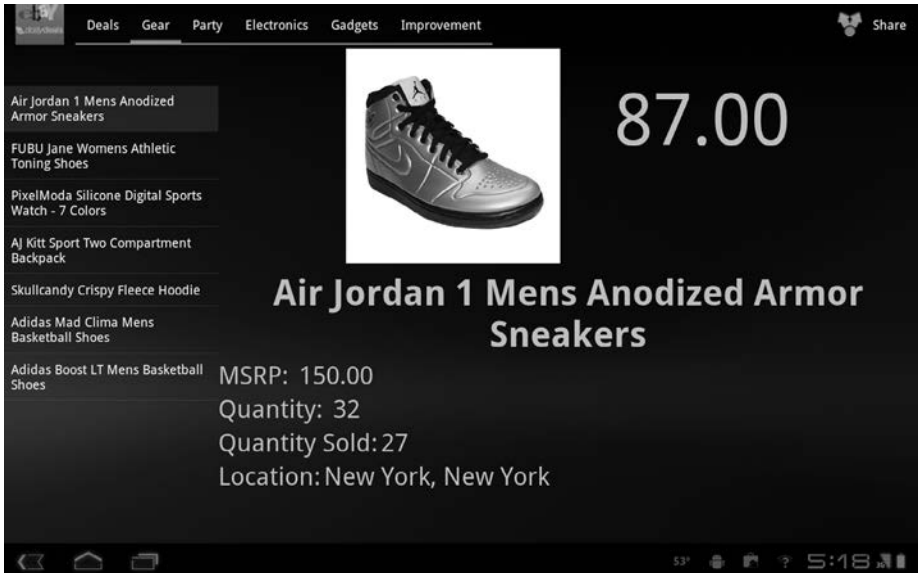
Rozwiązanie polega na użyciu fragmentów do uporządkowania kodu. Stosowanie różnych układów dla orientacji poziomej i pionowej nie jest niczym nowym. W przypadku tabletów istotna jest natomiast ilość miejsca na ekranie. W smartfonach, gdzie wyświetlacze są mniejsze, w układach poziomych i pionowych zwykle dostępne są te same informacje oraz funkcje. Zmiana orientacji prowadzi do sensownego nowego uporządkowania elementów. W tabletach nie jest niczym niezwykłym wyświetlanie na ekranie odmiennych komponentów w różnych układach. Przyjrzyjmy się konkretnemu przykładowi.

Aplikacja DealDroid (rozdział 2.) umożliwia użytkownikom wyświetlanie ofert dnia z eBaya. Jedna z aktywności aplikacji wyświetla listę ofert, a druga — szczegółowe informacje o wybranych ofertach. Na tablecie oba zadania można wykonywać w jednej aktywności, ale tylko w układzie poziomym. Na rysunku 15.4 pokazano wygląd takiej aktywności.

W aplikacjach na tablety często stosuje się pewien wzorzec. Po lewej stronie ekranu wyświetla się przewijaną listę, a po prawej — szczegółowe informacje o wybranym elemencie. Na rysunku 15.5 pokazano, że wybranie elementu z listy prowadzi do zmiany danych wyświetlanych w dużym obszarze ze szczegółami.

Wróćmy do problemu, czyli wyświetlania różnych komponentów w zależności od orientacji tabletu. Na rysunku 15.6 widać, co się dzieje po obróceniu urządzenia.

Porównaj rysunki 15.5 i 15.6. Widać, że obszar ze szczegółowymi informacjami wygląda tak samo w obu orientacjach. Różnice są podobne do tych, które znamy z aplikacji na smartfony. Jednak listy ofert wyglądają inaczej. Tego właśnie dotyczy określenie „zupełnie odmienne układy” z opisu problemu. W smartfonach takie podejście stosuje się bardzo rzadko, jednak w aplikacjach na tablety nie jest ono niczym niezwykłym.

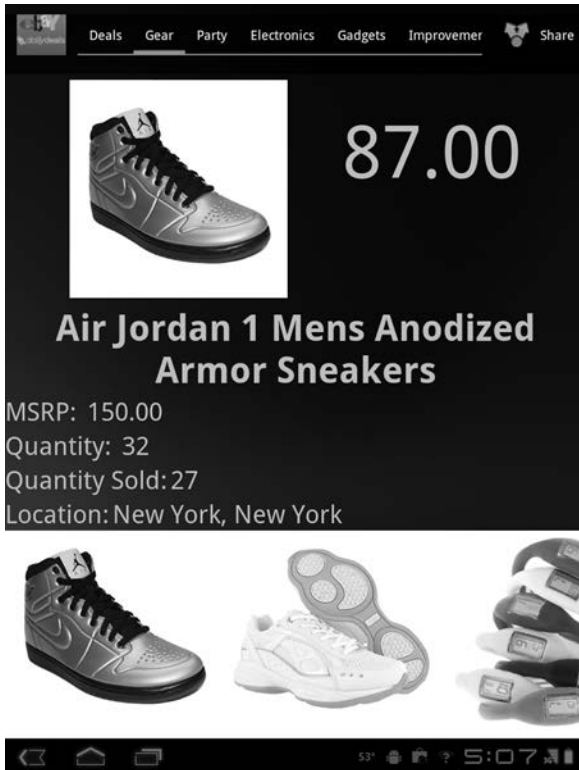


Rysunek 15.4. Lista ofert i szczegółowe informacje w układzie poziomym



Rysunek 15.5. Przeglądanie elementów z listy ofert

Najważniejszym mechanizmem przy tworzeniu aplikacji podobnych do pokazanej są fragmenty. Umożliwiają one podział interfejsu użytkownika na moduły. Na listingu 15.2 przedstawiono kod układu z rysunku 15.4.



Rysunek 15.6. Lista ofert i szczegółowe informacje w układzie pionowym

Listing 15.2. Kod XML układu ze szczegółowymi informacjami o ofercie (/res/layout/details.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/details_container">

    <fragment
        class="com.manning.aip.tabdroid.SectionDetailsFragment"  ← ❶
        android:id="@+id/section_list_fragment"
        android:visibility="gone"
        android:layout_marginTop="?android:attr/actionBarSize"
        android:layout_width="300dp"
        android:layout_height="match_parent" />

    <fragment class="com.manning.aip.tabdroid.DealFragment"  ← ❷
        android:id="@+id/deal_fragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Mamy nadzieję, że miłym zaskoczeniem jest dla Ciebie to, jak prosty jest plik układu dla widoku ze szczegółowymi informacjami. Kod obejmuje dwa fragmenty. Pierwszy ❶ wyświetla listę ofert po prawej stronie ekranu. Drugi ❷ pokazuje szczegółowe informacje na temat wybranego elementu. Kod pierwszego fragmentu przedstawiono na listingu 15.3.

Listing 15.3. Fragment wyświetlający listę ofert (plik `SectionDetailsFragment.java`)

```
public class SectionDetailsFragment extends ListFragment {    ←❶
    Section section;
    int currentPosition = 0;
    DealsApp app;
    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        app = (DealsApp) this.getActivity().getApplication();
        section = app.currentSection;

        if (savedInstanceState != null){    ←❷
            currentPosition = savedInstanceState.getInt("currentPosition");
            int savedSectionPos =
                savedInstanceState.getInt("currentSection", -1);
            if (savedSectionPos >= 0){
                section = app.sectionList.get(savedSectionPos);
                app.currentSection = section;
            }
        } else if (app.currentItem != null){
            for (int i=0; i<section.items.size(); i++){
                if (app.currentItem.equals(section.items.get(i))){
                    currentPosition = i;
                    break;
                }
            }
        }
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {    ←❸
        super.onActivityCreated(savedInstanceState);
        buildUi();
    }
    private void buildUi(){
        ListView listView = this.getListView();    ←❹
        listView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
        String[] dealTitles = new String[section.items.size()];
        int i = 0;
        for (Item item : section.items){
            dealTitles[i++] = item.title;
        }
        setListAdapter(new ArrayAdapter<String>(getActivity(),
            R.layout.deal_title_list_entry, dealTitles));    ←❺
        listView.setSelection(currentPosition);
        showDeal(currentPosition);
    }
}
```

Aby utworzyć fragment, wystarczy utworzyć klasę pochodną od `android.app.Fragment`. Tu stworzymy klasę pochodną od `ListFragment` ❶, która sama jest klasą pochodną od klasy `Fragment`. Klasa `ListFragment` obejmuje jeden widok `ListView` i często służy do wyświetlania list elementów w układzie z podzielonym ekranem, takim jak na rysunku 15.4. Fragment ma odrębny cykl życia powiązany z cyklem życia nadrzędnej aktywności. Aktywność ta żąda widoku od fragmentu przez wywołanie metody `onCreateView` danego fragmentu. Metoda `onCreate` ❷ fragmentu jest wywoływana bezpośrednio po wywołaniu metody `onCreate` aktywności, jednak przed metodą `onCreateView` fragmentu. W metodzie `onCreate` przywracamy lub ustawiamy stan fragmentu (podobnie jak robimy z aktywnością). Pewien czas po wywołaniu metody `onCreateView` fragmentu następuje wywołanie metody `onActivityCreated` ❸. Jak wskazuje nazwa, wywołanie to ma miejsce po utworzeniu aktywności. We wspomnianej metodzie konfigurujemy widok `ListView` ❹ będący częścią fragmentu `ListFragment`. Widok ten działa jak inne widoki `ListView`, dlatego trzeba określić dla niego adapter `ListAdapter` ❺, który zapewni dane i układ elementów z widoku.

Warto zauważyć, że ostatnią operacją w ramach konfigurowania interfejsu użytkownika fragmentu `ListFragment` jest wywołanie metody `showDeal`. Metoda ta wyświetla konkretną ofertę w głównym fragmencie ze szczegółowymi informacjami. Dlatego metodę tę należy wywoływać po wybraniu elementu listy. Na listingu 15.4 przedstawiono kod do wyświetlania ofert i obsługi dotknięcia elementu.

Listing 15.4. Wyświetlanie konkretnej oferty (plik `SectionDetailsFragment.java`)

```
@Override
public void onItemClick(ListView l, View v, int position, long id) {
    this.currentPosition = position;    ← ❶
    showDeal(position);
}

private void showDeal(int position){
    app.currentItem = app.currentSection.items.get(position);
    DealFragment fragment =
        (DealFragment) getFragmentManager().findFragmentById(
            R.id.deal_fragment);    ← ❷
    fragment.showCurrentItem();    ← ❸
}
```

Jednym z wygodnych aspektów korzystania z klasy `ListFragment` jest to, że trzeba przesłonić metodę `onItemClick`, aby obsługiwać dotknięcie elementów listy. Tu sprawdzamy wybrany element listy ❶. Następnie wywołujemy metodę `showDeal` z listingu 15.3. Drugi fragment ma wtedy wyświetlić inną ofertę, dlatego potrzebny jest uchwyt do tego fragmentu ❷. Do pobrania uchwytu używamy egzemplarza klasy `FragmentManager`, dostępnego w każdym fragmencie. Wróć do listingu 15.2. Zwróć uwagę, że przypisaliśmy do fragmentu identyfikator, który

można teraz wykorzystać do uzyskania uchwytu. Po jego pobraniu należy wywołać metodę `showCurrentItem` ❸, aby nakazać ponowne wyświetlenie fragmentu. Na listingu 15.5 przedstawiono tę metodę i pozostały kod klasy `DealFragment`.

Listing 15.5. Fragment do wyświetlania ofert (plik `DealFragment.java`)

```
public class DealFragment extends Fragment { ← ❶
    DealsApp app;
    private ProgressBar progressBar;
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container,
        Bundle savedInstanceState) { ← ❷
        app = (DealsApp) getActivity().getApplication();
        View dealView = inflater.inflate(R.layout.deal_details,
            container,
            false); ← ❸
        progressBar = (ProgressBar) dealView.findViewById(R.id.progress);
        progressBar.setIndeterminate(true);
        Item item = app.currentItem;
        if (item != null) {
            populateDealView(dealView, item);
        }
        return dealView;
    }
    private void populateDealView(View dealView, Item item) { ← ❹
        ImageView icon = (ImageView) dealView.findViewById(
            R.id.details_icon);
        icon.setImageResource(R.drawable.placeholder);
        new RetrieveImageTask(icon).execute(item.picUrl);
        TextView title =
            (TextView) dealView.findViewById(R.id.details_title);
        title.setText(item.title);
        CharSequence pricePrefix =
            getText(R.string.deal_details_price_prefix);
        TextView price =
            (TextView) dealView.findViewById(R.id.details_price);
        price.setText(pricePrefix + item.convertedCurrentPrice);
        TextView msrp = (TextView) dealView.findViewById(
            R.id.details_msrp);
        msrp.setText(item.msrp);
        TextView quantity =
            (TextView) dealView.findViewById(R.id.details_quantity);
        quantity.setText(Integer.toString(item.quantity));
        TextView quantitySold = (TextView) dealView.findViewById(
            R.id.details_quantity_sold);
        quantitySold.setText(Integer.toString(item.quantitySold));
        TextView location =
            (TextView) dealView.findViewById(R.id.details_location);
        location.setText(item.location);
    }
    public void showCurrentItem(){ ← ❺
        Item item = app.currentItem;
        View dealView = getView();
        populateDealView(dealView, item);
    }
}
```

To kolejny fragment. Tym razem bezpośrednio tworzymy klasę pochodną od klasy `Fragment` ❶. Nie trzeba przejmować się zarządzaniem stanem tego fragmentu, ponieważ jest on powiązany ze stanem aktywności (i fragmentu `ListFragment` z listingu 15.3). Dlatego wystarczy przesłonić metodę `onCreateView` ❷. Zauważ, że do tej wywoływanej zwrótnie metody przekazujemy obiekt klasy `LayoutInflater`. Wykorzystujemy go do przekształcenia pliku XML układu na widok ❸. Następnie wiążemy ❹ dane wybranej oferty z kontrolkami z pliku XML układu. W końcowej części listingu znajduje się metoda `showCurrentItem`, którą mogą wywoływać inne fragmenty. Metoda ta sprawdza, który element jest wybrany, i przekazuje go do używanej już wcześniej metody `populateDealView` ❺.

Po zmianie orientacji tabletu na pionową należy wyświetlić inny układ, widoczny na rysunku 15.6. Najprościej uzyskać ten efekt przez zastosowanie odrębnego pliku XML układu. Kod układu dla orientacji pionowej przedstawiono na listingu 15.6.

Listing 15.6. Układ dla orientacji pionowej (/res/layout-port/details.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/details_container"
    android:gravity="bottom">

    <fragment class="com.manning.aip.tabdroid.DealFragment" ← ❶
        android:id="@+id/deal_fragment"
        android:layout_marginTop="?android:attr/actionBarSize"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
    />

    <fragment class="com.manning.aip.tabdroid.FilmstripFragment" ← ❷
        android:id="@+id/section_filmstrip_fragment"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:layout_gravity="bottom"
    />
</LinearLayout>
```

Kod z listingu 15.6 jest podobny do kodu z listingu 15.2. Ponownie wykorzystujemy tu ❶ opisany już fragment `DealFragment`. Przeznaczeniem fragmentów jest właśnie umożliwianie powtórnego wykorzystania kodu. Zauważ, że nie pokazujemy kodu aktywności obejmującej fragmenty. Nie ma takiej potrzeby. Fragmenty są niezależne. W orientacji pionowej zastępujemy klasę `SectionDetailsFragment` klasą `FilmstripFragment` ❷. Kod tej ostatniej znajduje się na listingu 15.7.

Klasa `FilmstripFragment` jest nieco podobna do klasy `SectionDetailsFragment` z listingu 15.3. Obie klasy wyświetlają wszystkie oferty z danej kategorii i umożliwiają dotknięcie oferty w celu wyświetlenia szczegółowych informacji na jej

Listing 15.7. Pozioma lista rysunków używana do wybierania ofert (plik FilmstripFragment.java)

```

public class FilmstripFragment extends Fragment {
    @Override
    public void onCreate(Bundle savedInstanceState){ ← ❶
        super.onCreate(savedInstanceState);
        // Kod do zarządzania stanem pominięto.
    }
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container,
        Bundle savedInstanceState){ ← ❷
        HorizontalScrollView strip =
            (HorizontalScrollView) inflater.inflate(R.layout.filmstrip,
                container,
                false);
        fillWithPics(strip);
        return strip;
    }
    private void fillWithPics(HorizontalScrollView strip) {
        ViewGroup pics = (ViewGroup) strip.findViewById(R.id.pics);
        if (pics.getChildCount() > 0){
            pics.removeAllViews();
        }
        int i =0;
        for (Item item : section.items){ ← ❸
            ImageView imgView = new ImageView(getActivity());
            // Kod do pobierania bitmapy.
            imgView.setOnClickListener(new OnClickListener(){
                @Override
                public void onClick(View img) {
                    currentPosition = pos; ← ❹
                    showDeal(pos);
                }
            });
        }
        showDeal(currentPosition);
    }
    private void showDeal(int position){
        app.currentItem = app.currentSection.items.get(position);
        DealFragment fragment = (DealFragment) getFragmentManager()
            .findFragmentById(R.id.deal_fragment);
        fragment.showCurrentItem(); ← ❺
    }
}

```

temat w komponencie DealFragment. Podobny mechanizm stosujemy w innych fragmentach. Najpierw w metodzie onCreate ❶ przywracamy stan. Dalej znajduje się implementacja metody onCreateView ❷, która przekształca plik XML układu w zwracany widok. Tym razem nie stosujemy widoku ListView, ale „rolkę filmu” — przewijany w poziomie zbiór rysunków (jest to widok HorizontalScrollView; zobacz plik */res/layout-port/filmstrip.xml*). Wystarczy wypełnić ten widok ❸ obiektami klasy ImageView, obejmującymi bitmapy z rysunkami ofert. Dla każdego widoku ImageView trzeba ustawić metodę obsługi zdarzeń ❹ wywołowaną w reakcji

na dotknięcie grafiki przez użytkownika. Zdarzenie to prowadzi do pobrania za pomocą klasy `FragmentManager` uchwytu do fragmentu `DealFragment` i wywołania metody `showCurrentItem` tego fragmentu ❸.

OMÓWIENIE

Fragmenty umożliwiają porządkowanie kodu aplikacji w nowy sposób. Fragment pod wieloma względami może być „samowystarczalny”. Samodzielnie zarządza wyświetlanymi danymi i obsługuje swój stan. Fragmenty z przykładowej aplikacji zależą od globalnego stanu aplikacji (obiektu typu `Application`). Można z nich korzystać w dowolnym miejscu omawianego programu, ale już nie poza nim. Jest to celowe. Inna możliwość to utworzenie fragmentów zależnych od usług lub całkowicie niezależnych.

Programiści stosowali ten wzorzec na długo przed wprowadzeniem fragmentów, choć framework Androida tego nie ułatwiał. Jedno z często używanych podejść polegało na tworzeniu komponentów interfejsu użytkownika, które potrafią zarządzać stanem, pobierać dane i wykonywać podobne operacje. Odmianą tego wzorca jest rozwijanie komponentów interfejsu użytkownika bezpośrednio komunikujących się z usługą wykonującą wszystkie skomplikowane operacje. Choć zdaniem niektórych programistów należy unikać takich rozwiązań, ponieważ naruszają paradygmat model-widok-kontroler, opisane wzorce często okazują się przydatne. Załóżmy, że w aplikacji znajduje się nagłówek z informacjami o stanie, na przykład o liczbie nieprzeczytanych wiadomości lub nowych ofert dnia. Przed wprowadzeniem fragmentów często (oprócz łączenia kodu modelu z kodem komponentu interfejsu użytkownika) tworzono klasę aktywności do zarządzania stanem. Następnie aktywność tę stosowano jako klasę bazową dla wszystkich pozostałych aktywności aplikacji. Rozwiązania te mają wady i zalety, przy czym fragmenty pozwalają pisać dużo bardziej przejrzysty kod, niż jest to w przypadku innych podejść.

Technika oparta na bazowej aktywności często służy też do obsługi menu. Taka aktywność może tworzyć menu wyświetlane we wszystkich aktywnościach. Jednym z powodów tworzenia menu na poziomie aplikacji jest to, że zazwyczaj ma ono dla programisty niewielkie znaczenie. Jeśli w menu znajduje się jakaś ważna opcja, i tak trzeba umieścić ją także w innym miejscu ekranu, ponieważ bywa, że użytkownicy nie korzystają z menu. W menu nierzadko znajdują się też typowe opcje, takie jak *O programie*, *Pomoc techniczna*, *Wyrejestruj się* itd. W wersji Honeycomb wprowadzono rozwiązanie znacznie wygodniejsze od menu — pasek akcji (ang. *Action Bar*). W następnej technice opisujemy ten mechanizm i wyjaśniamy, kiedy warto go stosować.

TECHNIKA 90. Pasek akcji

Warto stosować menu w Androidzie. Można umieścić w nim wiele skrótów i przydatnych opcji. Można też udostępniać w nim operacje kontekstowe. Poważnym problemem jest jednak to, że użytkownicy rzadko zaglądają do menu.

Wskutek tego zaczęto tworzyć paski akcji. Często stosuje się je w tym samym celu co samo menu, są jednak skuteczniejsze z uwagi na większą widoczność dla użytkownika.

PROBLEM

Chcemy wyświetlać dodatkowe, ale użyteczne funkcje dostępne w kontekście używanej akurat aktywności. Nie zamierzamy jednak stosować standardowego menu Androida, ponieważ użytkownicy często nie korzystają z niego.

ROZWIĄZANIE

Rozwiązanie polega na zastosowaniu paska akcji. Znajduje się on w górnej części ekranu i jest dobrze widoczny dla użytkowników. Eliminuje to największy kłopot związany z menu. Na rysunku 15.7 pokazano przykładowy pasek akcji w aplikacji na tablety.



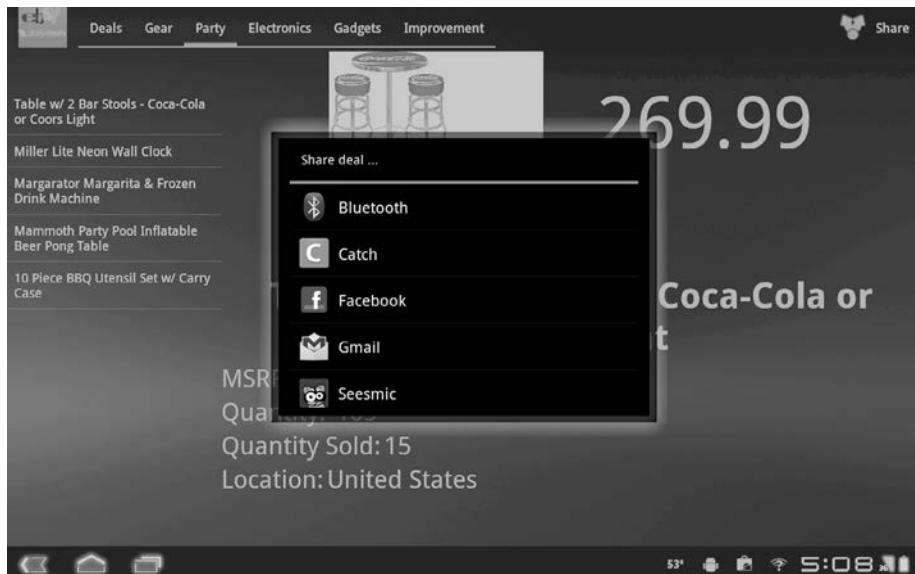
Rysunek 15.7. Pasek akcji w akcji

Jak widać, pasek akcji znajduje się w górnej części ekranu. W przykładowym programie na pasku są ikona aplikacji, kilka zakładek i przycisk *Podziel się*. Ikona aplikacji pozwala użytkownikom przejść do głównego ekranu, a zakładki służą do przechodzenia do różnych kategorii ofert dnia z eBaya. Na rysunku 15.8 pokazano, że przycisk *Podziel się* pozwala „podzielić się” ofertą z innymi osobami za pomocą aplikacji zainstalowanych w urządzeniu.

Jak może pamiętać, w pierwszej wersji aplikacji DealDroid funkcja „dzielenia się” była ukryta w menu. W wersji dla tabletów nawigacja jest wygodniejsza. Pasek akcji nie tylko pozwala rozwiązać problem z menu, ale ma też inne funkcje. Zakładki nawigacyjne omawiamy dalej. Teraz skupimy się na ikonach aplikacji i funkcji „dzielenia się”. Na listingu 15.8 przedstawiono kod tych elementów.

Listing 15.8. Ikona aplikacji i funkcji „dzielenia się” z paska akcji (plik `DetailsActivity.java`)

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.details_menu, menu); ← ❶
    return true;
}
```



Rysunek 15.8. „Dzielenie się” ofertą na tablecie

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            Intent intent = new Intent(this, DealsMain.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(intent);
            return true;
        case R.id.share_action:
            shareDealUsingChooser("text/*");
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

private void shareDealUsingChooser(final String type) {
    // Z uwagi na zwięzłość pominięto. Kod jest taki sam jak w rozdziale 2.
}

private String createDealMessage() {
    // Z uwagi na zwięzłość pominięto.
}

```

Na listingu 15.8 widać, że przodkiem paska akcji jest menu. Aby utworzyć pasek akcji, należy zaimplementować wywołaną zwrótnie metodę `onOptionsItemSelected` aktywności ❶. Elementy paska akcji można tworzyć programowo. Jest to przydatne zwłaszcza wtedy, gdy wyświetlanie elementów zależy od stanu aktywności. Inna możliwość to określenie zawartości paska akcji w XML-u. Oto kod w XML-u tworzący pasek akcji z rozdziału 15.7:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/share_action"
        android:title="@string/deal_details_share_menu"
        android:icon="@drawable/ic_menu_share"
        android:showAsAction="ifRoom|withText" />
</menu>
```

Jak widać, określony jest tu jeden element z tytułem i ikoną. W przykładzie tytuł i ikona to zewnętrzne zasoby, dlatego można utworzyć ich wersje dla innych języków. Zwróć też uwagę na atrybut `showAsAction`. Pojawił się on w Androidzie 3.0 i służy do określenia, kiedy dana opcja menu ma być dostępna jako akcja i jak ma wyglądać. Można ustawić ten atrybut na `always`, jednak jeśli na ekranie brakuje miejsca, pasek wygląda nieelegancko.

Wróćmy do listingu 15.8. Aby zdefiniować działanie paska akcji (reakcję na dotknięcie opcji przez użytkownika), należy zaimplementować metodę `onOptionsItemSelected` aktywności. W ten sam sposób określana jest reakcja na dotknięcie ikony aplikacji, widocznej po lewej stronie paska ❷. Ikona ta jest identyfikowana na podstawie predefiniowanego identyfikatora zasobu (`home`). Aplikacja w reakcji na wybranie tej ikony opróżnia stos aktywności i kieruje użytkownika do głównego ekranu ❸. Dotknięcie przycisku *Podziel się* można wykryć przez dopasowanie identyfikatora zdefiniowanego w XML-owym kodzie menu do identyfikatora wybranego elementu `MenuItem` ❹. Wybranie wspomnianego przycisku prowadzi do wywołania metody `shareDealUsingChooser` z aplikacji *Deal-Droid* z rozdziału 2. Aplikacja wyświetla wtedy interfejs użytkownika widoczny na rysunku 15.8.

Wiesz już, jak tworzyć ikony i określać ich działanie. Przyjrzyjmy się teraz, jak tworzyć zakładki widoczne na rysunku 15.7. Potrzebny kod pokazano na listingu 15.9.

Listing 15.9. Tworzenie zakładek paska akcji i zarządzanie nimi (plik `DetailsActivity.java`)

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.details);
    app = (DealsApp) getApplication();
    ActionBar bar = this.getActionBar(); ← ❶
    TabListener listener = new TabListener(){
        @Override
        public void onTabReselected(Tab t, FragmentTransaction txn) {}
        @Override
        public void onTabSelected(Tab t, FragmentTransaction txn) {
            if (active){
                changeTab(t.getPosition()); ← ❷
            }
        }
        @Override
        public void onTabUnselected(Tab t, FragmentTransaction txn) {}
    };
    for (int i=0;i<Math.min(6, app.sectionList.size());i++){
```

```

final Section section = app.sectionList.get(i);
Tab tab = bar.newTab();
tab.setText(champ(section.title));
tab.setTabListener(listener);
if (app.currentSection != null &&
    app.currentSection.equals(section)){
    bar.addTab(tab, true);
} else {
    bar.addTab(tab);
}
}
bar.setDisplayShowTitleEnabled(false);
bar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
active = true;
}
private void changeTab(int position){
    FragmentManager fm = getFragmentManager();
    int orientation = getResources().getConfiguration().orientation;
    if (orientation == ORIENTATION_LANDSCAPE){
        SectionDetailsFragment fragment =
            (SectionDetailsFragment) fm.findFragmentById(
                R.id.section_list_fragment);
        fragment.setSection(position);
    } else {
        FilmstripFragment fragment =
            (FilmstripFragment) fm.findFragmentById(
                R.id.section_filmstrip_fragment);
        fragment.setSection(position);
    }
}
}

```

Od wersji Android 3.0 każda aktywność może mieć pasek akcji. Jest on dostępny poprzez metodę `getActionBar` aktywności ❶. Pomysł polega na programowym tworzeniu zakładek i dodawaniu ich do paska akcji. Każda zakładka wymaga odbiornika `TabListener`, który będzie reagował na dotknięcie, dlatego tworzymy jeden taki odbiornik do obsługi wszystkich zakładek. Dalej znajduje się implementacja metody `onTabSelected` ❷ i wywołanie metody `changeTab` na podstawie pozycji wybranej zakładki. Działanie metody `changeTab` omawiamy dalej.

Kiedy egzemplarz odbiornika `TabListener` jest już gotowy, można utworzyć zakładki i dodać je do paska akcji. Programowo tworzymy zakładkę ❸, ustawiamy jej tytuł i odbiornik `TabListener` ❹. Zauważ, że na podstawie kategorii wybranej przez użytkownika aplikacja określa obecnie zaznaczoną zakładkę. Nakazujemy też paskowi akcji, aby nie wyświetlał nazwy aktywności, a w zamian pokazywał zakładki nawigacyjne.

Przyjrzyjmy się teraz metodzie `changeTab` wywoływanej przez zwrótną metodę `onTabSelected` odbiornika `TabListener`. Metoda `changeTab` najpierw sprawdza orientację urządzenia ❺. Jest to potrzebne, ponieważ układ tabletu wpływa na zawartość aktywności. Metoda wykorzystuje informację o orientacji i obiekt klasy `FragmentManager` z aktywności do uzyskania uchwytu do wyświetlanego fragmentu. Następnie ustawiamy kategorię dla fragmentu ❻, co pozwala określić, jakiego rodzaju oferty dnia mają być widoczne.

OMÓWIENIE

Nawigacja z wykorzystaniem zakładek nie jest niczym nowym ani specjalnym dla tabletów. Od lat jest powszechnie używana w aplikacjach sieciowych i występuje w Androidzie od wersji 1.0. Do tworzenia zakładek zawsze służyły klasy `TabHost` i `TabWidget`. Pierwsza z nich umożliwia tworzenie zestawu zakładek, z których każda powiązana jest z wyświetlaną aktywnością. Zakładki paska akcji to rozwinięcie tej techniki, podobnie jak inne aspekty tego paska są rozwinięciem menu.

Aby zbudować nawigację opartą na zakładkach z paska akcji, należy utworzyć zakładki w podobny sposób jak w klasie `TabHost`. Jednak zamiast łączyć z każdą zakładką odrębną aktywność, można pracować w ramach jednej aktywności i stosować fragmenty. W przykładzie zmieniamy zawartość fragmentu. Zwróć jednak uwagę na to, że do metody `onTabSelected` przekazywany jest obiekt klasy `FragmentManager`. Dlatego w aktywności można wykonywać różne operacje na fragmentach, na przykład usuwać je lub zastępować innymi. Pasek akcji nie tylko jest ulepszeniem dawnego systemu menu, ale w połączeniu z fragmentami sprawia też, że porządkowanie kodu aplikacji jest prostsze, i daje przy tym więcej możliwości.

Ostatnia z podstawowych technik związanych z tabletami, przeciąganie, pozwala usprawnić interakcję użytkowników z aplikacją.

TECHNIKA 91. Przeciąganie

Odkąd Douglas Engelbart wymyślił mysz komputerową, menedżerowie produktu żądają od programistów dodawania funkcji przeciągania. W rozbudowanych frameworkach do tworzenia aplikacji desktopowych mechanizm przeciągania jest dostępny od wielu lat. W aplikacjach sieciowych przez długi czas występowały znaczne problemy z jego obsługą. Pracę programistom ułatwiały frameworki JavaScriptu, aż w końcu mechanizm przeciągania stał się częścią specyfikacji języka HTML5. W świecie urządzeń mobilnych do momentu pojawienia się Androida 3.0 przeciąganie było w frameworkach pomijane. Oczywiście, można było dodać jego obsługę za pomocą interfejsów API do obsługi dotknięć, jednak technikę tę wykorzystywano głównie w grach. Od wersji Honeycomb przeciąganie można stosunkowo łatwo dodać do aplikacji dowolnego rodzaju.

PROBLEM

Chcemy umożliwić użytkownikom bardziej intuicyjną interakcję z aplikacją przez udostępnienie przeciągania różnych elementów.

ROZWIĄZANIE

Aby umożliwić przeciąganie w aplikacji, wystarczy zastosować kilka interfejsów API wprowadzonych w Androidzie 3.0. W ramach przykładu przedstawiamy prostą aplikację z funkcją przeciągania. Program wyświetla na ekranie widoki

StackView (jest to nowa kontrolka wprowadzona w wersji Honeycomb) i umożliwia użytkownikom zmianę uporządkowania tych kontroltek przez ich przeciągnięcie. Wygląd aplikacji przedstawiono na rysunku 15.9.



Rysunek 15.9. Aplikacja z funkcją przeciągania

Jak widać, aplikacja wyświetla prostą siatkę z kilkoma kontrolkami StackView. Na listingu 15.10 znajduje się kod tego układu.

Listing 15.10. Plik XML z układem z siatką, używanym przez mechanizm przeciągania

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TableRow>
        <LinearLayout android:layout_width="640dp"
            android:layout_height="345dp"
            android:id="@+id/topLeft">
            <StackView android:id="@+id/stack"
                android:layout_width="250dp"
                android:layout_height="250dp
                android:clickable="true"
                android:loopViews="true"
                android:longClickable="true"
            />
        </LinearLayout>
        <LinearLayout android:layout_width="640dp"
            android:layout_height="345dp"
            android:id="@+id/topRight"
        />
    </TableRow>
</TableLayout>
```

```

<LinearLayout android:layout_width="640dp"
    android:layout_height="345dp"
    android:id="@+id/bottomLeft"
/>
<LinearLayout android:layout_width="640dp"
    android:layout_height="345dp"
    android:id="@+id/bottomRight">
    <StackView android:id="@+id/stack2"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:clickable="true"
        android:loopViews="true"
        android:longClickable="true"
    />
</LinearLayout>
</TableRow>
</TableLayout>

```

W kodzie z listingu 15.10 używamy układu `TableLayout` ❶. Nie ma w nim nic wyjątkowego. Tu pozwala łatwo zidentyfikować różne części ekranu na potrzeby pokazu przeciągania. Każda komórka tabeli obejmuje układ `LinearLayout` ❷. Także on nie ma żadnych specjalnych cech. Potrzebujemy tylko kontenera, do którego można przeciągać widoki `StackView` ❸. Również te widoki nie mają żadnych specjalnych cech w kontekście przeciągania. Co więcej, prościej byłoby ich nie używać, jednak są ciekawe wizualnie, dlatego warto z nich korzystać w aplikacjach na tablety.

Najważniejszą cechą aplikacji jest umożliwianie użytkownikom przeciągania widoków `StackView` do różnych kontenerów (układów `LinearLayout`) na ekranie. Aby rozpocząć przeciąganie, użytkownik musi dotknąć i przytrzymać stos (długie kliknięcie), co powoduje podświetlenie miejsc, w których można upuścić elementy. Na rysunku 15.10 pokazano takie miejsca.

Wszystkie pozostałe operacje w aplikacji wykonujemy programowo. Na listingu 15.11 znajduje się kod tworzący interfejs użytkownika.

Listing 15.11. Tworzenie interfejsu użytkownika

```

public class DndActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.grid);

        StackView stack = (StackView) findViewById(R.id.stack); ← ❶
        Bitmap[] bmps = new Bitmap[5];
        Resources res = getResources(); ← ❷
        bmps[0] = BitmapFactory.decodeResource(res, R.drawable.donut);
        bmps[1] = BitmapFactory.decodeResource(res, R.drawable.ecclair);
        bmps[2] = BitmapFactory.decodeResource(res, R.drawable.froyo);
        bmps[3] =
            BitmapFactory.decodeResource(res, R.drawable.gingerbread);
        bmps[4] =
            BitmapFactory.decodeResource(res, R.drawable.honeycomb);
    }
}

```



Rysunek 15.10. Aktywne obszary, w których można upuścić elementy

```

ImgAdapter adapter = new ImgAdapter(bmps, stack);
stack.setAdapter(adapter);

StackView stack2 = (StackView) findViewById(R.id.stack2);
stack2.setAdapter(new ImgAdapter(bmps, stack2));

findViewById(R.id.topLeft).setOnDragListener(
    new BoxDragListener());
findViewById(R.id.bottomLeft).setOnDragListener(
    new BoxDragListener());
findViewById(R.id.topRight).setOnDragListener(
    new BoxDragListener());
findViewById(R.id.bottomRight).setOnDragListener(
    new BoxDragListener());
    }
}

```

Pierwszą rzeczą, jaką trzeba zrobić w celu utworzenia interfejsu użytkownika, jest uzyskanie referencji do jednego z widoków `StackView` ❶. Następnie aplikacja wczytuje zasoby graficzne będące częścią aplikacji ❷. Na podstawie rysunków 15.9 i 15.10, a także kodu z listingu 15.11 można stwierdzić, że rysunki używane w widokach `StackView` to ikony reprezentujące różne wersje Androida (aż do wersji Honeycomb). Rysunki te są przekształcane na obiekty klasy `Bitmap` i przekazywane do niestandardowego adaptera widoku `StackView` ❸ (kod tego adaptera przedstawiono na listingu 15.12). `StackView` to oparta na adapterze kontrolka podobna do widoków `ListView` i `GridView`. Pobieramy uchwyty do kontenerów `LinearLayout` z listingu 15.10 i przekazujemy im obiekt typu `OnDragListener` ❹.

OnDragListener to nowy interfejs wprowadzony w Androidzie 3.0. Jego implementację przedstawiono na listingu 15.13. Teraz przyjrzyjmy się niestandardowemu adapterowi widoków StackView.

Listing 15.12. Adapter używany dla widoków StackView w aplikacji

```
class ImgAdapter extends BaseAdapter{ ← ❶
    private Bitmap[] bmps;
    private Context ctx = DndActivity.this;
    private ViewGroup owner;
    ImgAdapter(Bitmap[] bmps, ViewGroup owner){
        this.bmps = bmps;
        this.owner = owner;
    }
    @Override
    public int getCount() {
        return bmps.length;
    }
    @Override
    public Object getItem(int index) {
        return bmps[index];
    }
    @Override
    public long getItemId(int index) {
        return index;
    }
    @Override
    public View getView(int index, View recycledView, ViewGroup parent) {
        if (recycledView == null){
            recycledView = new ImageView(ctx);
        }
        ImageView imgView = (ImageView) recycledView;
        imgView.setOnLongClickListener(new OnLongClickListener(){
            @Override
            public boolean onLongClick(View view) { ← ❷
                ClipData data =
                    ClipData.newPlainText("foo", "bar");
                DragShadowBuilder sBuilder =
                    new DragShadowBuilder(owner);
                owner.startDrag(data, sBuilder, owner, 0); ← ❸
                return true;
            }
        });
        imgView.setImageBitmap(bmps[index]);
        return imgView;
    }
}
```

Wspomnieliśmy już, że widok StackView przypomina widoki ListView i GridView. Wszystkie te widoki korzystają z adaptera tego samego rodzaju, dlatego tu stworzymy klasę pochodną od BaseAdapter ❶. Jej kod jest podobny do kodu innych adapterów. Adapter ImgAdapter z listingu 15.12 można zastosować także dla widoku ListView lub GridView. Jest on przeznaczony nie tylko dla widoków StackView. Jedyny wyjątkowy kod służy do obsługi przeciągania. Najpierw z widokiem

StackView wiążemy odbiornik `OnLongClickListener` ❷. Przeciąganie chcemy inicjować w momencie długiego kliknięcia, wykrywanego właśnie przez ten odbiornik. Nie chcemy przeciągać poszczególnych widoków `ImageView` z widoku `StackView` — interesuje nas przeciąganie całego widoku `StackView`. To dlatego przechowujemy referencję `owner` do tego widoku. Dalej wywołujemy metodę `startDrag` ❸, dodaną do klasy `android.view.View` w wersji Honeycomb Androida. Metoda ta początkuje przeciąganie widoku `StackView`. Zauważ, że jednym z parametrów metody `startDrag` jest egzemplarz klasy `DragShadowBuilder`. Klasa ta odpowiada za wyświetlanie cienia widoku przeciąganego po ekranie. Tu używamy domyślnej wersji klasy `DragShadowBuilder`. Wyświetla ona widok przekazany do konstruktora tej klasy. Wyświetlanie takiego widoku może wymagać dużo zasobów. Jeśli tak jest, warto utworzyć klasę pochodną od `DragShadowBuilder` i wyświetlać w niej niestandardowy cień. Istnieje też inne rozwiązanie — można zastosować domyślną wersję klasy `DragShadowBuilder`, jednak nie przekazywać widoku do jej konstruktora. Wtedy klasa nie wyświetla żadnego cienia.

Do czego służy klasa `ClipData`?

Może zauważyłeś, że na listingu 15.13 przekazaliśmy do metody `startDrag` obiekt klasy `ClipData`. Jeśli tak, prawdopodobnie zwróciłeś uwagę, że umieściliśmy w tym obiekcie fikcyjne dane. Klasa `ClipData` jest przydatna, kiedy widok reprezentuje skomplikowane dane przeciągane w aplikacji. Widok może na przykład wyświetlać obiekt z danymi kontaktowymi, które aplikacja pomaga porządkować. Klasa `ClipData` pozwala w wygodny sposób powiązać z przeciąganym widokiem wskaźnik do złożonego obiektu. Ponadto klasa ta jest potrzebna do udostępniania niestandardowego mechanizmu kopiowania i wklejania.

Po zainicjowaniu przeciągania widoku `StackView` w aplikacji trzeba dodać obsługę upuszczania go w jednym z kontenerów `LinearLayout`. Na listingu 15.13 pokazano, jak to zrobić za pomocą ustawionego wcześniej odbiornika `OnDragListener`.

Listing 15.13. Implementacja interfejsu `OnDragListener` dla kontenerów

```
class BoxDragListener implements OnDragListener{
    boolean insideOfMe = false;           ← ❶
    Drawable border = null;
    Drawable redBorder = getResources().getDrawable(R.drawable.border3);
    @Override
    public boolean onDrag(View self, DragEvent event) {
        if (event.getAction() == DragEvent.ACTION_DRAG_STARTED){
            border = self.getBackground();
            self.setBackgroundDrawable(redBorder);           ← ❷
        } else if (event.getAction() == DragEvent.ACTION_DRAG_ENTERED){
            insideOfMe = true;
        } else if (event.getAction() == DragEvent.ACTION_DRAG_EXITED){
            insideOfMe = false;
        } else if (event.getAction() == DragEvent.ACTION_DROP){
            if (insideOfMe){
                View view = (View) event.getLocalState();
            }
        }
    }
}
```

```

        ViewGroup owner = (ViewGroup) view.getParent();
        owner.removeView(view);
        LinearLayout container = (LinearLayout) self;
        if (container.getChildCount() > 0){
            container.addView(view,
                container.getChildCount());
        } else {
            container.addView(view);
        }
    } else if (event.getAction() == DragEvent.ACTION_DRAG_ENDED){
        self.setBackgroundDrawable(border);
    }
    return true;
}

```

W każdym kontenerze w aplikacji używamy egzemplarza klasy z implementacją interfejsu `OnDragListener` (listing 15.13). Kontener musi śledzić, czy przeciągany widok się w nim znajduje. Informacja ta jest przechowywana w zmiennej logicznej ❶. W momencie rozpoczęcia przeciągania obramowanie kontenera ustawiamy na kolor czerwony ❷. Jest to informacja, że w danym kontenerze można upuścić przeciągany widok. Następnie śledzimy zdarzenia `ENTERED` i `EXIT` za pomocą zmiennych logicznych. Po zgłoszeniu upuszczenia trzeba zrobić dwie rzeczy. Najpierw należy usunąć przeciągany widok z poprzedniego kontenera ❸. Później trzeba dodać przeciągany widok do nowego kontenera ❹. Po wykryciu zakończenia przeciągania należy przywrócić domyślny wygląd obramowania ❺. Warto zauważyć, że nie ma tu znaczenia, jakiego rodzaju widok jest przeciągany do kontenera. Ponadto jeśli dany kontener nie ma obsługiwać mechanizmu przeciągania, wystarczy nie ustawiać odbiornika `OnDragListener`. Kontener bez tego odbiornika nie ma czerwonego obramowania i nie przyjmuje upuszczonych widoków.

OMÓWIENIE

Nie bez powodu programiści nieustannie są proszeni o dodawanie obsługi przeciągania do aplikacji. W programach dowolnego rodzaju przeciąganie pozwala w bardziej intuicyjny sposób komunikować się z aplikacją. W przeszłości przeciąganie odbywało się za pomocą myszy w aplikacjach desktopowych oraz sieciowych — i nawet przy korzystaniu z tego urządzenia operacja ta była intuicyjna.

Obecnie nastała era urządzeń dotykowych, z których jako pierwsze pojawiły się smartfony. Jednak przeciąganie rzadko było w nich stosowane, choć nie z powodu braku potrzebnych mechanizmów w Androidzie. Po prostu operacja ta na małych ekranach jest niewygodna. Miejsca, w których można dotknąć ekran, aby rozpocząć przeciąganie, są zwykle małe. Jednak na tabletach sytuacja wygląda zupełnie inaczej. Przeciąganie jest w nich dużo wygodniejsze niż w smartfonach. Tablety mają duże wyświetlacze, dlatego można precyzyjnie wskazać przeciągany element. Ponadto przeciąganie za pomocą interfejsów dotykowych jest

znacznie wygodniejsze niż posługiwanie się myszą. Przy przeciąganiu z wykorzystaniem myszy pojawia się ikona dłoni, co ma imitować przenoszenie elementów w rzeczywistym świecie. Przeciąganie na tabletach nie wymaga żadnego imitowania, ponieważ to prawdziwa, ludzka dłoń służy do interakcji z obiektami na ekranie.

Możliwości tworzenia intuicyjnych interfejsów opartych na przeciąganiu w aplikacjach na tablety są atrakcyjne i niemal nieskończone. Trzeba jednak pokonać kilka przeszkód. Ponieważ przeciąganie nie było popularne w urządzeniach dotykowych (smartfonach), nie istnieją powszechnie przyjęte sposoby informowania użytkowników o tym, że aplikacja obsługuje tę technikę. W przykładowej aplikacji początkiem przeciągania jest dotknięcie ekranu i przytrzymanie palca (czyli *długie kliknięcie*, jak określają to programiści aplikacji na Android). Nie chcemy rozpoczynać przeciągania w reakcji na każde dotknięcie widoku `StackView`. To uniemożliwiłoby użytkownikom przeglądanie rysunków z widoku `StackView`. Ponadto irytująca byłaby sytuacja, gdyby każde dotknięcie ekranu skutkowało wyświetlaniem czerwonego obramowania wokół kontenerów. Nie twierdzimy jednak, że długie kliknięcie powinno być powszechnie przyjętym sposobem początkowania przeciągania w aplikacjach na tablety. Inna możliwość to dodanie przycisku *Edycja* (na przykład na pasku akcji) do włączania trybu przeciągania, w którym dotknięcie początkuje przeciąganie obiektu.

15.3. Podsumowanie

Z pewnością wiesz, że Android nie był pierwszą platformą na smartfony, którą przeniesiono na tablety. Sukces Androida w dotykowych smartfonach sprawił, że zastosowanie go w tabletach było oczywiste. Jednak, jak zobaczyłeś w tym rozdziale, twórcy tej platformy nie spoczęli na laurach i wprowadzili szereg istotnych usprawnień z myślą o tabletach. Najważniejszą nowinką było zastosowanie fragmentów. Wspomnieliśmy już, że potrzeba tworzenia bardziej niezależnych komponentów w Androidzie nie jest warunkowana tylko tym, że pojawiły się tablety. Programiści odczuwali ją już wcześniej i wymyślali rozmaite rozwiązania problemu. Jednak na tabletach chciano tworzyć interfejsy użytkownika lepiej wykorzystujące dużą powierzchnię ekranu. Fragmenty to umożliwiły. Obecnie fragmenty (a także inne ważne mechanizmy programowania aplikacji na tablety — pasek akcji i przeciąganie) są dostępne także we wcześniejszych wersjach Androida poprzez pakiet `Android Compatibility`.

Warto zauważyć, że rozdział ten nie jest wyczerpującym omówieniem wszystkich zmian wprowadzonych w wersji `Honeycomb` Androida. Większość modyfikacji ma pomagać programistom pisać aplikacje na tablety, dlatego pasuje do tego rozdziału. Zdecydowaliśmy się jednak skoncentrować na najważniejszych technikach, które powinni znać autorzy wszystkich aplikacji na Android. Nie chcemy przy tym umniejszać znaczenia innych funkcji. Pominęliśmy tu na przykład kilka nowych rozwiązań (m.in. środowisko `Renderscript`) bardzo przydat-

nych dla twórców gier. Nie omówiliśmy też usprawnień klasy `RemoteView`, ulepszających obsługę kontrolek i powiadomień na ekranie głównym. Wszystkie te funkcje są istotne, a w niektórych sytuacjach niemal niezastąpione. Szczegółowy opis ulepszeń w interfejsach API i działaniu mechanizmów znajdziesz, jak zawsze, w dokumentacji Androida. W ten sposób doszliśmy do końca książki! Zacząłeś od utworzenia aplikacji `HelloAndroid` w rozdziale 1., przebrnąłeś przez wiele zaawansowanych zagadnień i ponad 80 technik, a na końcu zapoznałeś się z tabletami. W tym momencie masz bardzo solidne podstawy do tworzenia aplikacji na Android. Możesz sobie pogratulować!

A

- adapter, 91, 444, 459
 - MovieAdapter, 158
 - widoków StackView, 659
- adaptery niestandardowe, 92
- ADB, Android Debug Bridge, 665
- adres pocztowy, 425
- ADT, Android Development Tools, 34, 608
- AIDL, Android IDL, 208
- Ajax, Asynchronous JavaScript and XML, 389
- akcelerometr, 58
- akcja
 - ACTION_SEND, 99
 - MAIN, 102
- akcje spółek, 218
- aktualizowanie
 - informacji w tle, 217
 - lokalizacji, 416, 421
 - pamięci podręcznej, 216
- aktywności, 82
 - aplikacji LifecycleExplorer, 117
 - główne, 131
 - oparte na listach, 84
 - z pliku Main.java, 118
- aktywność, 39, 72
 - BrewLocationDetails, 432
 - DealList, 548
 - MapActivity, 427
 - OpenGLGreenScreenActivity, 498
 - ProviderDetail, 411
 - SlideshowActivity, 456, 460
- alarm, 226
- alarm systemowy, 225
- algorytm
 - decyzyjny, 394
 - najbliższego sąsiada, 517
 - wyboru folderu, 194
- Android, 30
- Android Asset Processing Tool, 43
- Android Debug Bridge, 64
- Android Market, 30, 435, 476
- animacja niestandardowa, 456
- animacje, 454
- ANR, Activity Not Responding, 237
- ANR, Application Not Responding, 572
- Ant, Another neat tool, 583–592
- Apache Ant, 35
- Apache Harmony, 46
- aparaty VGA, 437
- aplikacja
 - Barcode Scanner, 59
 - BrewMap, 422, 427
 - Browser, 58
 - Bubble, 59
 - Camera, 467
 - Coin Flip, 59
 - Compass, 59
 - DealDroid, 70, 106, 529, 674, 705
 - FileExplorer, 283, 287
 - Gallery, 332
 - GoodShares, 330
 - HelloAndroid, 35
 - HelloAnt, 619, 689
 - HelloMaven, 599
 - Hoccer, 59
 - ImageMash, 330, 333, 339
 - LifecycleExplorer, 117, 120

aplikacja
 Locale, 59
 MediaMogul, 436, 447, 468
 MyMovies, 138, 395
 MyMoviesDatabase, 296, 300, 324
 OpenGLDemo, 497
 StockPortfolio, 208

aplikacje
 natywne, 683
 otwarte, 33
 typu klient-serwer, 671
 w JavaScriptcie, 681
 wykorzystujące czujniki, 59

APNS, Apple Push Notification Service, 230

archiwizacja, 580

archiwum APK, 581

argument nazwany, keyword argument, 705

argumenty określające typy, 257

artefakty, 597

artefakty Mavena, 613

asynchroniczna praca, 256

asynchroniczne wywoływanie usługi, 338

asynchroniczne zadania, 264

atrapy, 556, 559

atrybut
 android:color, 181
 android:configChanges, 125
 android:listSelector, 177
 cacheColorHint, 171
 minSdkVersion, 189
 supports-screens, 191
 windowBackground, 173

atrybuty
 intencji, 102
 układu, 144
 widoków, 145

automatyczne dostosowywanie aplikacji, 186

automatyzacja budowania aplikacji, 631

AVD, Android Virtual Devices, 35, 45

awaria sieci, 393

azymut, 407

B

baza
 filmów IMDB, 158
 kontaktów, 349
 mymovies.db, 324

bazy danych, 57, 299

biblioteka
 Apache Commons Lang, 585, 591
 Apache HttpClient API, 47
 Apache HTTP Components, 366
 AWT, 47

Bionic, 51

Calculon, 553

cglib, 560

ddmlib, 706

EasyMock, 560

glibc, 51

GLUtils, 516

ignition, 375

java.io, 280, 285

JavaScript Object Notation, 48

Mockito, 566

OpenGL, 494, 500, 519

OpenGL ES, 56, 477, 493, 495

Robotium, 551–553

SGL, 56

SQLite, 56

Swing, 47

WebKit, 58

biblioteki
 natywne, 56
 uruchomieniowe, 685

blokada wzbudzająca, wake lock, 226

blokowanie bazy danych, 57

błąd OutOfMemoryError, 604

błędy ANR, 572

błędy w HttpURLConnection, 365

budowanie, 607, 615
 aplikacji, 577, 589
 macierzowe, matrix build, 613, 617, 625–629
 z wykorzystaniem Hudsona, 617

bufor wierzchołków, 512

C

C2DM, Cloud to Device Messaging, 230

cel, target, 578
 clean, 602
 distribution, 584
 emulator-start, 603
 install, 602, 604

ceny akcji, 218

certyfikat bezpieczeństwa, 580

ContentProvider, 75

CRUD, create, read, update, delete, 353

cykl życia, 108
 aktywności, 113, 460
 komponentów, 111

czas ustalenia lokalizacji, 419

czujnik, 58
 akcelerometr, 58
 ciśnienia, 58
 GPS, 58
 pola magnetycznego, 58
 sztucznego światła, 58

temperatury, 58
 zbliżeniowy, 58
 żyroskop, 58
 czujniki geoprzestrzenne, 407

D

Dalvik, 48, 50
 Dalvik Debug Monitor, 65
 dane środowiskowe, 92
 DAO, Data Access Object, 302
 debugowanie, 44, 665, 676
 debugowanie przez port usb, 666
 definicja
 motywu, 169
 stylu, 168
 definiowanie
 zadań, 131
 zasobów, 76
 deklaracja
 odbiornika, 206
 układu, 80
 usługi, 203
 DEX, Dalvik Executable, 579
 diagram encja-związek, 304
 długie kliknięcia, 662
 długość geograficzna, 405
 dodatek Google APIs Add-On, 423
 dokumentacja, 32
 Javadoc, 332
 ProGuarda, 687
 dołączanie układów, 152, 154
 DOM, Document Object Model, 47, 380
 domknięcia, 684
 dostawca
 gps, 411
 kontaktów, 350
 network, 411
 położenia, 411, 414
 treści, 73, 304, 349, 356
 treści niestandardowy, 353
 treści multimedialnych, 447
 dostęp do
 danych, 302, 303
 interfejsu API instrumentacji, 540
 zasobów, 79
 zdjęć, 444, 468
 drzewo katalogów, 52
 DSL, domain-specific languages, 525
 dzienniki budowania, 615
 dzienniki systemowe, 672
 dźwięk, 458

E

Eclipse, 34, 577
 efekt FILL, 486
 efekty
 dwuwymiarowe, 490
 dźwiękowe, 462
 graficzne, 490
 tekstowe, 492
 ekran, 189, 194
 aktywności CanvasDemo, 479
 Main, 119
 OLED, 357
 powitalny, 269, 270
 serwera budowania, 616
 właściwości projektu, 37
 ekrany aplikacji BrewMap, 422
 element
 <supports-screens>, 188, 194
 <uses-permission>, 227
 scope, 598
 uses-feature, 438
 uses-permissions, 438
 elementy
 obiektów graficznych, 178
 stylu, 166
 emulator, 409
 emulator Androida, 45, 65, 619
 etapy cyklu życia, 114

F

fabryka, 387
 FAT, File Allocation Table, 281
 filtrowanie, 89
 filtry intencji, 44, 96, 103, 452
 Firefoks
 rozszerzenie SQLiteManager, 325
 format
 *.9.png, 183
 DEX, 579
 JSON, 375, 389
 OASIS XLIFF, 78
 wymiany danych, 375
 XML, 375
 fragmenty, 642, 648, 650
 FrameLayout, 146
 framework, 33
 Apache Turbine, 592
 Google Guice, 560
 JUnit 3, 532
 JUnit 4, 565
 Robolectric, 561
 Robolectrica, 566

framework
 Robotium, 549, 551
 Spring, 560
 funkcja camera.front, 438
 funkcje pierwszej kategorii, 684

G

geokodowanie, 425, 427
 geolokalizowanie, 683
 gęstość ekranu, 190
 gęstość pikseli, 188, 194
 Google APIs Add-On, 423
 Google Geocoding API, 427
 Google Maps, 427
 Google Maps API Premier, 427
 GPRS, General Packet Radio Service, 357
 GPS, 58, 408
 GPU, graphics processing unit, 493
 gradient, 176
 grafika dwuwymiarowa, 478
 grafika trójwymiarowa, 493

H

Harmony, 46
 HDPI, high dots per inch, 192
 hierarchia ustawień, 297
 hierarchia widoków, 139, 140
 HTTP, HyperText Transfer Protocol, 358
 Hudson, 617–622

I

I, info, 673
 IDE, 34
 IDEA, 677
 identyfikator
 artifactId, 597
 R.string.deal_details, 79
 URI, 350
 identyfikatory
 rejestracyjne, 233
 specjalne, 110
 tekstur, 518
 użytkownika, 110
 tekstur, 516
 w układach, 151
 zarezerwowane zasobów, 86
 implementacja interfejsu OnDragListener, 660
 inflating, 86
 informacje o projekcie, 588
 inicjowanie testów, 543

instalowanie
 aplikacji, 65, 589
 wtyczek, 609
 instrukcja ALTER TABLE, 57
 instrukcje HTTP
 DELETE, 359
 GET, 359
 HEAD, 359
 OPTIONS, 359
 POST, 359
 PUT, 359
 TRACE, 359
 instrumentacja, 539
 integracja
 asynchroniczna, 338
 synchroniczna, 336
 integracyjna baza danych, 347
 IntelliJ, 677
 intencje, 73, 96–101, 329
 interfejs
 API, 48, 60, 221
 API fragmentów, 642
 ContentHandler, 381
 DataManager, 308, 318
 do pobierania danych, 354
 Handler.Callback, 246
 narzędzia android, 64
 obiektów DAO, 312
 OnItemLongClickListener, 377
 Parcelable, 211
 protokołu HTTP, 359
 renderscript, 495
 SAX, 379
 sprzętowy, 34
 użytkownika, 186, 190
 wygenerowany, 211
 interfejs wysokopoziomowy
 MediaPlayer, 462
 SoundPool, 462
 IoC, inversion of control, 560
 izolowanie połączeń, 373

J

JavaScript, 680
 JAXB, Java API for XML Binding, 47
 JDBC, 316
 JDT, Java Development Tools, 34
 Jenkins, 618
 język
 HTML5, 683
 IDL, 208
 Scala, 685
 XPath, 380

języki
 do testowania, 553
 DSL, 525, 550
 programowania, 683–686
 JRE, Java Runtime Environment, 45
 JSON, JavaScript Object Notation, 48, 389–392
 JVM, Java Virtual Machine, 45

K

kanal RSS, 87
 karta SD, 207, 281, 446
 katalog
 anttasks, 587
 assets, 78, 443
 cache, 55
 drawable-hdpi, 192
 drawable-ldpi, 192
 gen, 38
 główny, 53
 raw, 76
 res, 39
 src, 38
 tools, 61
 z narzędziami, 588
 kategoria
 Build, 624
 Configuration Matrix, 626
 LAUNCHER, 102
 klasa
 Activity3, 127
 ActivityInstrumentationTestCase2, 546
 ActivityUnitTestCase, 540, 542
 Adapter, 86, 91, 93
 AlarmManager, 208, 222, 226
 AlarmReceiver, 224
 android.app.AlarmManager, 222
 android.app.Notification, 218
 android.content.ContentProvider, 348
 android.content.Intent, 332
 android.content.pm.PackageManager, 438
 android.content.res.AssetManager, 442
 android.media.SoundPool, 462
 android.provider.MediaStore.Images, 450
 android.provider.MediaStore.Video, 450
 android.R.attr, 145, 167
 android.R.style, 167
 android.R.styleable, 167
 android.view.View, 82
 AndroidHttpClient, 374
 AndroidTestCase, 534
 ApplicationTestCase, 535
 ArrayAdapter, 87, 91–94
 AssetManager, 442
 AsyncTask, 87, 203, 256–268
 BasicHttpParams, 373
 Bomb, 699
 BrewLocationOverlay, 431, 433
 Bundle, 122
 Camera, 60, 474
 Canvas, 478, 483, 519
 CheckBoxPreference, 298
 ClipData, 660
 ColouredPyramid, 510, 512
 Configuration, 124
 ConnectivityManager, 398
 ContentProvider, 75
 Context, 91, 92
 CursorAdapter, 450
 CustomButton, 490, 492
 DataManager, 322
 DataManagerImpl, 319–322
 DealDetails.class, 101
 DealDroidApp, 104, 535
 DealExporterTest, 558
 DealList, 75, 84
 DealsAdapter, 88
 DealsApp, 636
 DefaultHandler, 381
 DefaultHttpClient, 370, 372
 DefaultHttpRequestRetryHandler, 394
 FileDescriptor, 294
 FileOutputStream, 294
 FileUtil, 288
 FragmentManager, 650
 FragmentTransaction, 655
 Geocoder, 425, 426
 GeoPoint, 432
 GetMovieRatingTask, 378
 getView, 94
 GridAdapter, 445
 Handler, 272, 417
 HashMap, 161
 HashSet, 459
 HttpClient, 367, 370
 HttpContext, 367
 URLConnection, 360, 364, 365
 IdleHandler, 276
 ImageHandler, 253
 InstrumentationTestRunner, 539
 Instrumentation, 544, 548
 InstrumentationTestCase, 534
 InstrumentationTestRunner, 670
 Intent, 96, 99
 IntentFilter, 96, 102, 103
 ItemizedOverlay, 430
 java.lang.Thread, 241
 java.util.Timer, 203
 JsonMovieParser, 391

klasa

JsonStringer, 612
 LayoutInflater, 141
 LifecycleActivity, 122
 ListActivity, 86
 ListView, 84
 LocationHelper, 416, 421
 LocationListener, 420
 LocationManager, 407
 Looper, 273–276
 Main, 39
 MapController, 429
 MapResults, 428, 430
 MapView, 430
 MediaPlayer, 459–465
 MediaRecorder, 474
 MediaScannerConnection, 468
 MediaStore, 447, 450
 Message, 246, 272
 MockOutputStream, 558, 559
 ModelBase, 306
 MonkeyDevice, 703
 MonkeyHelper, 708
 MonkeyImage, 703
 MonkeyRunner, 702
 Movie, 305
 MovieAdapter, 159
 MovieCategoryTable, 310
 MovieDao, 313–317
 MovieTable, 308
 MyOpenGLRenderer, 498
 obsługi protokołów, 360
 OpenGLPyramidActivity, 508
 OpenGLTexturedPyramidActivity, 514
 OpenGLTriangleActivity, 503
 OpenHelper, 308
 Paint, 491, 493
 Parcel, 211
 PendingIntent, 225
 PortfolioManagerService, 204, 212
 PortfolioStartupReceiver, 224
 PreferenceActivity, 296, 298
 ProviderTestCase2, 534
 Pyramid, 507
 R, 38
 RemoteViews, 221
 RobolectricTestRunner, 562
 SchemeRegistry, 372
 Section, 92
 Service, 696
 ServiceTestCase, 534
 ShapesAndTextView, 485
 ShareActivity, 332
 SharedPreferences, 295
 Solo, 553

Song, 449
 SQLiteOpenHelper, 304, 306
 Stock, 210
 StrictMode, 674, 675
 Stub, 212
 Surface, 472
 SurfaceHolder, 472
 TabActivity, 86
 TableLayout, 149
 TestCase, 532
 TexturedPyramid, 514, 517
 Thread, 110, 269
 ThreadPoolExecutor, 250
 ThreadSafeClientConnManager, 371, 373
 Timer, 223, 271
 TimerTask, 223
 Triangle, 501, 503
 Typeface, 487
 UpdateNoticeTask, 368
 Uri, 450
 URL, 360
 URLConnection, 364
 VideoView, 464
 View, 41
 ViewHolder, 160
 WakeLock, 228
 WebView, 678
 XMLHttpRequest, 683
 XmlPullMovieParser, 386

klasy

aktywności, 39, 84, 97
 animacji, 454
 anonimowe wewnętrzne, 87
 pakietu com.google.android.maps, 428
 pamięci, storage classes, 309
 zastępcze, shadow class, 561

klucz API, 424, 429

klucze obce, 311

kod

5xx, 396
 bajtowy, 48, 579
 kreskowy, 469
 obiektu graficznego, 179
 QR, 31, 469
 źródełowy, 579
 kolejka komunikatów, 246, 275
 kolejność testów, 537
 kolor płótna, 480
 kolory, 171
 kompilacja, 683
 kompilator
 dx, 62
 JIT, 50
 komponent obsługi, handler, 245

komponenty
 aplikacji, 72
 platformy, 31
 komunikacja
 międzyprocesowa, 205, 333
 z aplikacją, 218
 z serwerem HTTP, 367
 z usługą, 208
 komunikat o błędzie, 669
 komunikaty, 246, 272
 HTTP, 369
 rozgłoszeniowe, 398
 konfiguracja
 ekranu, 189
 procesu budowania, 621
 ProGuarda, 689
 sieci, 398
 skryptu budowania, 587
 wyświetlacza, 191
 konfiguracje
 testów, 555
 zadań, 622
 konfigurowanie
 menu, 89
 obiektu klienta, 372
 projektów testowych, 563
 rejestrowania, 473
 kontener, 661
 kontener LinearLayout, 80
 kontrolka, 82
 ListView, 81, 89, 141, 170
 Spinner, 81, 87
 kształty, 176, 178
 kursor, 449
 kwalifikatory zasobów, 194

L

LDPI, low dots per inch, 192
 liczby zmiennoprzecinkowe, 501
 licznik egzemplarza, 128
 LinearLayout, 147
 Linux, 51
 lista, 84
 ofert, 645
 utworów, 458
 wtyczek, 621
 lokalizacja, 411

Ł

łańcuchy znaków, 77
 łączenie
 atrybutów widoków, 168
 Eclipse z Mavenem, 607
 elementów projektu, 43

M

macierz
 konfiguracji, 627
 rzutowania, 509
 manifest aplikacji, 73
 manifest BrewMap, 424
 manipulowanie macierzami, 509
 mapy, 422
 marginesy
 wewnętrzne, padding, 145, 184
 zewnętrzne, margin, 145
 maska uprawnień, 344
 maszyna
 stanowa, 496
 wirtualna, 33
 wirtualna Dalvik, 45, 48, 671
 wirtualna Zygote, 49, 670
 Maven, 592–607
 Maven Central, 594
 mechanizm przeciągania, 656
 mechanizm uruchamiania testów, 562
 menedżer
 LocationManager, 407, 409, 411
 połączeń, 371
 układu, 143, 146
 FrameLayout, 146
 LinearLayout, 147
 RelativeLayout, 150
 TableLayout, 149
 metoda
 Activity.getApplication, 546
 addToPortfolio, 212
 AsyncTask.get, 547
 bindService, 213
 Context.getCacheDir, 292
 Context.startActivity, 102
 createLowPriceNotification, 220
 createPackageContent, 345
 DealDroidApp.onCreate, 536
 DealList.onCreate, 570
 debugEvent, 122
 doInBackground, 260
 drawArrays, 502
 find, 318
 finish, 123
 getActionBar, 654
 getApplication, 536
 getExternalStoragePublicDirectory, 445
 getFrontFacingCamera, 439
 getGpsStatus, 421
 getInstrumentation, 548
 getLastNonConfigurationInstance, 130, 266
 getPortfolio, 216
 getView, 250

metoda

- glBindTexture, 518
- glClearColor, 499
- glDrawArrays, 512
- glRotatf, 508
- gluPerspective, 509
- handleMessage, 254
- HttpClient.execute, 239
- Instrumentation.runOnMainSync, 549
- invokeMenuActionSync, 543
- ListActivity.onListItemClick, 161
- ListView.setChoiceMode, 161
- Menu.add, 90
- Movie.imdbLookup, 382
- myClickHandler, 692
- notifyDataSetChanged, 95
- obsługi kliknięcia, 693
- onActivityResult, 332
- onCreate, 83, 115, 205, 416
- onCreateOptionsMenu, 90, 98
- onCreateView, 646
- onDestroy, 115, 205
- onDraw, 492
- onDrawFrame, 499, 510, 517
- onMessage, 233
- onOptionsItemSelected, 90
- onPause, 83, 90, 115, 116
- onRestart, 115
- onRestoreInstanceState, 127
- onResume, 83, 115, 116, 414
- onRetainNonConfigurationInstance, 268
- onServiceConnected, 213
- onStart, 115, 205
- onStartCommand, 225
- onStop, 115
- onTabSelected, 654
- openDealInBrowser, 100
- ParseFeedTask.execute, 87
- populate, 432
- query, 350
- resetListItems, 88
- retryRequest, 394
- saveImage, 334
- scheduleAtFixedRate, 272
- sendLocationToHandler, 419
- setHighPriceNotification, 221
- setLastEventInfo, 221
- setLastFocusedIndex, 432
- setListAdapter, 86
- setTextViewText, 221
- shareDealUsingChooser, 99–102, 653
- SimpleCursorAdapter.ViewBinder, 450
- startActivity, 543
- startRecording, 473
- sync, 294

- testThatAllFieldsAreSetCorrectly, 543

- texImage2D, 516
- Thread.sleep, 275
- Thread.start, 241
- toString, 342, 556
- toString, 343
- TouchUtils.clickView, 549
- updateStockData, 216, 223
- waitAndUpdate, 547
- waitForIdleSync, 549

metody

- cyklu życia, 39, 114, 120
- opt*, 392
- testowe, 543

model

- ACID, 57
- DOM, 683
- Movie, 305
- OSI, 358
- POM, 593, 615

- model-widok-kontroler, 84, 95

modyfikator

- in, 209
- inout, 209
- out, 209

modyfikowanie

- dźwięku, 462
- wyglądu listy, 170
- monkeyrunner, 710
- kod źródłowy, 702
- skrypty, 703
- uruchamianie, 701

- motywy, 167, 168

- multimedia, 436, 475

N

- nagrywanie, *Patrz* rejestrowanie

- nakładka itemizedoverlay, 432

- narożniki, 195

narzędzia

- cURL, 374
- dla platformy, 61
- GLU, 509
- pakietu SDK, 61, 579
- podstawowe, 61
- uruchamiane z wiersza poleceń, 62

narzędzie

- aapt, 63, 76, 579
- Activity Manager, 669
- adb, 64, 65, 666
- AIDL, 209
- android, 63
- Apache Ant, 582, 584

ddms, 65, 110, 325, 706
 hierarchyviewer, 140
 logcat, 124
 maven-android-sdk-deployer, 610, 613
 Monkey, 568–572
 Monkeyrunner, 665, 701
 ProGuard, 687
 sqlite3, 324
 zipalign, 581
 nawiasy klamrowe, 390
 nawigacja, 655
 nazwy w liczbie mnogiej, 77
 NDK, Native Development Kit, 34
 niepowodzenie testu, 537
 notacja ?, 173
 notacja @, 173

O

obiekt
 IntentFilter, 75
 POJO, 377
 R.layout.main, 39
 SQLiteDatabase, 312
 obiekty DAO, 312
 obiekty graficzne, 41, 172–185
 dziewięciopółowe, 185
 elementy, 178
 kształty, 176
 predefiniowane, 176
 selektory, 179
 skalowanie, 182
 stany, 181
 obiekty klasy
 Application, 103
 HttpClient, 371
 MediaPlayer, 465
 SharedPreferences, 295
 obiekty modelu, 303
 Movie, 305
 obiekty typu
 DefaultHttpClient, 369
 Parcelable, 333, 685
 Runnable, 460
 obiekty w komunikacji HTTP, 367
 obrót ekranu, 124
 obsługa
 animacji, 457
 awarii sieci, 393
 CSS3, 58
 czujników, 58
 dotknięć, 432
 ekranów, 189, 194
 grafiki, 477

HTML-u, 58
 intencji, 339
 JavaScriptu, 58
 kluczy obcych, 311
 map, 423
 multimediiów, 56
 ponawiania żądań, 395
 protokołów, 360
 rejestracji, 232
 stanu egzemplarza, 129
 zarządzania zasobami, 638
 odbiornik
 AlarmReceiver, 228
 LocationListener, 414, 416
 OnSharedPreferenceChangeListener, 296
 TabListener, 654
 typu BroadcastReceiver, 73, 75, 111
 odczyt
 danych, 295
 pliku, 284
 odinstalowanie aplikacji, 33
 odświeżanie danych, 216
 odtwarzanie
 dźwięku, 462
 filmów, 463
 multimediiów, 453
 odwrotne geokodowanie, 425
 odwrócenie sterowania, 560
 odwzorowywanie UV, UV mapping, 514
 określanie położenia, 150, 414, 612
 opcja
 anyDensity, 190
 Create Activity, 39
 START_NOT_STICKY, 225
 OpenGL, Open Graphics Library, 494, 500, 519
 OpenGL ES, 493, 495
 operacje na plikach, 288
 operacje wejścia-wyjścia, 440
 operator APN, 397
 opis celów, 605
 orientacja, 640
 orientacja pionowa, 648

P

pakiet
 android.bluetooth, 60
 android.database, 301
 android.database.sqlite, 301
 android.graphics, 60
 android.hardware, 60
 android.location, 60
 android.media, 60
 android.opengl, 60

- pakiet
 - android.provider, 352
 - android.telephony, 60
 - android.widget, 82
 - APK, 70
 - com.google.android.maps, 428
 - GLUT, 509
 - java.io, 47
 - java.lang, 47
 - java.net, 47
 - java.nio, 47
 - java.sql, 47
 - java.util, 47
 - javax.sql, 47
 - NDK, 60
 - SDK, 30, 59
- pakiety
 - dla programistów Javy, 34
 - najwyższego poziomu, 46
- pakowanie, 683
- pamięć
 - podręczna, 214, 216
 - VRAM, 488
 - wewnętrzna, 280
 - zewnętrzna, 280
- parametry układu, 144, 148
- parser
 - JSON, 391
 - SAX, 381, 382, 388
 - SAXMovieParser, 383
 - StAX, 380
 - typu pull, 47
 - XmlPull, 380, 385, 388
- parsery, 380
 - strumieniowe, 380, 384, 392
 - typu pull, 380
 - typu push, 380
- partycja
 - systemowa, 33
 - specjalna, 281
- pary
 - klucz-wartość, 670
 - nazwa-wartość, 234
- pasek akcji, Action Bar, 650–652
- perspektywa
 - DDMS, 346
 - Hierarchy View, 140
- pętle komunikatów, 272
- piksele, 196
 - niezależne od gęstości, 195, 196
 - niezależne od skali, 196
- piramida, 505
 - kolorowanie, 510
 - tekstura, 513
- planowanie wykonania usługi, 223, 224
- plik
 - .nomedia, 291
 - Activity3.java, 127
 - AIDL, 338
 - android.jar, 529, 598, 691
 - android.R.styleable, 168
 - AndroidManifest.xml, 39, 43, 74, 230, 270, 531, 639
 - build.xml, 583–586
 - button_bar.xml, 153
 - classes.dex, 580
 - colors.xml, 172
 - DealDetails.java, 97
 - DealDroidApp.java, 104
 - DealFragment.java, 647
 - DealList.java, 84, 88–90
 - deallist.xml, 80
 - deals.txt, 557
 - DealsAdapter.java, 93
 - DetailsActivity.java, 651, 653
 - DownloadTask.java, 258
 - droid.gif, 41
 - dziennika, 672
 - FilmstripFragment.java, 649
 - ImageHandler.java, 254
 - InternalStorage.java, 284
 - IStockService.aidl, 209
 - LifecycleActivity.java, 120
 - list_selector.xml, 176, 179
 - Main.java, 39, 118
 - main.xml, 40, 242
 - main_rules.xml, 588
 - mapping.txt, 691
 - maps.jar, 610, 612
 - modelu POM, 596, 599
 - movies.xml, 158
 - MoviesAdapter.java, 252
 - mymovies.db, 324
 - MyMovies.java, 157, 362
 - plugin.jar, 709
 - plurals.xml, 78
 - pom.xml, 593
 - Preferences.java, 297
 - preferences.xml, 297
 - proguard.cfg, 694, 698
 - R.java, 38, 42
 - resources.arsc, 580
 - SectionDetailsFragment.java, 645, 646
 - seeds.txt, 691
 - settings.xml, 601
 - ShareActivity.java, 331, 332
 - SimpleImageDownload.java, 241
 - Stock.aidl, 210
 - strings.xml, 41, 77

- styles.xml, 166, 270
- title.9.png, 185
- usage.txt, 691
- pliki
 - .aidl, 209
 - .class, 45, 48, 62
 - .dax, 63
 - .dex, 49, 63, 684
 - .jar, 74
 - .ttf, 487
 - APK, 109
 - dziennika, 673, 691
 - dźwiękowe, 457
 - graficzne, 489
 - multimedialne, 440, 443
 - PNG, 183
 - typu SharedPreferences, 344
 - układu, 144
 - z bazą, 326
 - z kluczem, 581
 - z zasadami, 589
 - zasobów, 77, 78
- plótno, 481
- pobieranie danych kontaktowych, 350
- podgląd filmu, 472
- podpisywanie plików
 - APK, 581
 - JAR, 581
- podwójne buforowanie, 488
- podział układu, 155
- POJO, plain old Java object, 377
- pokaz slajdów, 455
- pole widzenia, 509
- polecenie
 - logcat, 672
 - ls, 53
 - mount, 281
 - ps, 112
 - sqlite3, 324
- połączenie HttpURLConnection, 363
- POM, Project Object Model, 593
- pomiary widoków, 142
- ponawianie żądań, 393, 395
- porządkowanie widoków, 143
- powiadomienia, 217, 229
- powiadomienia typu toast, 217
- powłoka poleceń, 324
- powłoka urządzenia, 667
- poziomy komunikatów, 673
- priorytety procesów, 113
- proces budowania, 578, 589, 604
- procesor graficzny, 493
- procesy, 110, 214
- program wyboru, chooser, 98, 100
- programowaniem natywne, 60
- ProGuard, 687, 694, 700
 - dane wyjściowe, 689
 - konfiguracja, 688
 - opcje, 698
 - reguły, 691, 695, 696
- projekt
 - apache ivy, 592
 - brewmap, 422
 - canvasdemo, 478
 - dealdroidmonkeyrunner, 706
 - dealdroidrobolectrictest, 562
 - dealdroidrobotiumtest, 551
 - dealdroidtest, 529
 - dealdroidwithexport, 557
 - fileexplorer, 283
 - handlingactivityinterruptions, 265
 - helloant, 584
 - helloanttest, 619
 - hellomaven, 596
 - hellomavenwithmaps, 614
 - imagedownloadwithmessagepassing, 247
 - lifecycleexplorer, 118
 - locationinfo, 408
 - mediamogul, 441
 - mymovies, 138
 - mymoviesdatabase, 296
 - mymovieswithhttpclient, 366
 - mymovieswithimages, 250
 - mymovieswithimagesasynctask, 257
 - mymovieswithsplash-screen, 269
 - mymovieswithupdatenotice, 361
 - opengl demo, 497
 - producerconsumerwithlooper, 273
 - proguarded, 692
 - ruboto, 685
 - simpleimagedownload, 240
 - stockportfolio, 202
- promień narożnika, 195
- protokół
 - FTP, 358
 - HTTP, 360
 - HTTP/1.1, 369
 - XVNC, 624
- przebieg
 - pomiarowy, 141
 - rozieszczania, 141
- przechowywanie
 - danych, 635
 - ustawień, 294
- przechwytywanie wyjątków, 260
- przeciąganie, 655
- przeciąganie StackView, 657, 660
- przeglądanie pliku z bazą, 326
- przekazywanie
 - informacji między wątkami, 249
 - komunikatów, 247, 329

przekształcanie
 adresu pocztowego, 425
 układów na klasy, 86
 przekształcenia afiniczne, 330
 przełącznik kształtów, 180
 przenośność, 186
 przestrzeń barw ARGB, 219
 przesyłanie
 danych, 331
 komunikatów, 272
 przetwarzanie
 dokumentu XML, 381
 danych, 379
 przezroczystość, 171
 pseudolosowość, 571
 pula wątków, 250, 259
 punkt montowania, 53, 280
 punkty rozszerzeń, 584

R

RC, Remote Control, 553
 referencja do
 aktywności, 263, 264
 klucza obcego, 310
 obiektu, 295
 widoku StackView, 658
 zasobów, 79
 reguły ProGuarda, 692
 rejestracja, 231
 rejestrowanie
 dźwięku, 470
 filmów, 470, 473
 zdjęć, 465
 relacje, 57
 relacyjne bazy danych, 299
 RelativeLayout, 150
 renderowanie
 obiektów, 506
 obrazu, 499
 w odrębnym wątku, 499
 repozytorium Maven Central, 594, 610
 rodzaje
 zdarzeń, 572
 połączeń, 371
 rozmieszczanie widoków, 142
 rozszerzalny obszar, 184
 rozszerzenie
 Google APIs Add-On, 423
 SQLiteManager, 325
 rozwijanie układu do klasy, 141
 RPC, remote procedure call, 336
 rysowanie, 519
 figur, 487
 kształtów, 481, 482

rysunek tytułowy, 182, 185
 rzadkie macierze konfiguracji, 630
 rzutowanie, 506
 ortogonalne, 506
 perspektywiczne, 506
 rzutowanie perspektywiczne, 506

S

SAX, Simple API for XML, 47
 scalanie układów, 152
 scena trójwymiarowa, 505
 schemat architektury, 32
 SD, Secure Digital, 281
 SDK, Software Development Kit, 29
 selektor
 listy, 177, 180
 obiektów graficznych, 179–181
 przezroczysty, 177
 serializowanie danych, 375
 Service, 75
 serwer XVNC, 624
 serwery budowania, 615, 616
 sieć
 3G, 393
 Wi-Fi, 393
 silnik
 SQLite, 57
 V8, 58
 WebKit, 58
 silniki
 bazodanowe, 56
 motywów, 165
 skalowanie, 195
 skalowanie widoków, 182
 sklep Android Market, 31, 435, 476
 skrypt monkeyrunnera, 703
 słowo kluczowe
 synchronizacji, 244
 volatile, 244
 sortowanie, 89
 specyfikacja
 SAX, 376
 XmlPull, 376
 sprawdzanie cen, 218
 SQL, Structured Query Language, 300
 SQLite, 56, 299–301, 311
 SQLiteDatabase, 307
 SQLiteManager, 325
 stan egzemplarza, instance state, 108, 126, 129
 aktywności, 125
 niezwiązany z konfiguracją, 129
 stan
 GPS-u, 421
 testowy, test fixture, 543

- trwały, 126
 - widoczności, 114
 - widoku, 181
 - StAX, Streaming API for XML, 47, 380
 - sterowanie zdarzeniami, 572
 - stopniowe wzbogacanie, 439
 - stos aktywności, 83, 123, 132, 133
 - struktura
 - katalogu głównego, 53
 - obiektów graficznych, 174
 - projektu, 38
 - projektu wtyczki, 707
 - układu, 144
 - strumień
 - InputStream, 468
 - java.io.InputStream, 446
 - styl STROKE, 491
 - styl tła, 170, 174
 - style, 166, 171
 - symbol
 - #, 53
 - \$, 53
 - /, 53
 - @, 79
 - symulowanie połączenia telefonicznego, 66
 - synchroniczne wywoływanie usługi, 336
 - system
 - Apache Maven, 592
 - DBMS, 57
 - ext4, 293
 - X11, 624
 - systemy
 - budowania, 622
 - operacyjne, 33
 - plików, 281
 - plików z księgowaniem, 293
 - szerokość geograficzna, 404
 - szybkość, 407
- ## Ś
- ścieżka
 - bezwzględna, 52
 - względna, 52
 - środowisko
 - IDE, 37, 596
 - IDE Eclipse, 34, 577
 - uruchomieniowe, 45
 - uruchomieniowe Dalvik, 33
- ## T
- tabele, 304
 - tablety, 641
 - TDD, test-driven development, 523, 526
 - technologia
 - JDBC, 316
 - stax, 380
 - tekst, 172
 - tekstury, 494, 513
 - telefon wielofunkcyjny, 436
 - test jednostkowy, 533
 - testowanie
 - aktywności, 539
 - aplikacji, 525, 534
 - dostawców treści, 534
 - klas aplikacji, 536
 - scenariuszy, 527
 - usług, 534
 - testy, 523
 - funkcjonalne, 526, 544
 - jednostkowe, 525, 539
 - JUnit, 561
 - losowe, 554
 - obciążeniowe, 567
 - oparte na Javie, 528
 - Robolectrica, 563
 - z instrumentacją, 546, 554, 555
 - tło okna, 173
 - TMDb, The Movie Database, 377
 - trójkąt, 500
 - tryb
 - automatycznego skalowania, 190
 - letterbox, 190
 - pełnoekranowy, 480
 - tworzenie
 - adaptera, 92
 - aktywności MapActivity, 427
 - aplikacji, 35
 - aplikacji BrewMap, 424
 - aplikacji na tablety, 635, 638
 - baz danych, 303, 306
 - identyfikatora, 151
 - interfejsu użytkownika, 186, 458, 657
 - motywów, 165
 - obiektów DAO, 312
 - obiektu trójwymiarowego, 507
 - paska akcji, 652
 - piramidy, 504
 - plików multimedialnych, 465
 - pliku plugin.jar, 709
 - pokazu slajdów, 455
 - powiadomień, 217–220
 - powiązania z aktywnością, 266
 - projektu testowego, 530, 531
 - referencji współużytkowanej, 275
 - stosu aktywności, 132
 - trójkąta, 502
 - usługi, 201, 202
 - wtyczki, 706
 - zakładek paska akcji, 653

typ
 alarmu, 226
 serializowany, 210
 SurfaceView, 82

typy
 animacji, 454
 blokad, 227
 intencji, 101
 rzutowania, 506
 w AIDL-u, 209
 zasobów, 79

U

układ, 73, 80, 143
 aplikacji MyMovies, 182
 FrameLayout, 140, 455
 LinearLayout, 140
 LinearView, 139
 pokazu slajdów, 455
 RelativeLayout, 94

układu
 atrybuty, 144
 dołączanie, 154
 parametry, 145, 148
 scalanie, 152
 struktura, 144
 wbudowane menedżery, 146

układy
 niestandardowe, 89, 221
 pionowe, 644
 poziome, 643
 z siatką, 656

uprawnienia, 54, 75, 281, 344

uruchamianie
 aktywności, 669
 aplikacji, 44, 668
 emulatora, 623, 624
 Hudsona, 620
 komponentów, 669
 usługi, 204, 206, 207

urządzenia dotykowe, 633, 661

usługa, 73
 Apple Push Notification Service, 230
 Cloud to Device Messaging, 222, 229, 234
 Device Messaging, 208
 do zarządzania akcjami, 215
 LAYOUT_INFLATER_SERVICE, 94
 PortfolioManagerService, 203
 TMDb, 381, 391
 typu IntentService, 233
 usługi, 201, 202, 214
 komunikacja, 208
 sieciowe, 375

systemowe, 223
 uruchamianie, 204, 206
 wywoływanie, 213
 znacznik, 204
 ustawianie języka systemu, 670
 ustawienia supports-screens, 191
 usuwanie aktywności, 123, 130
 usypianie wątków, 275

W

wady
 parserów SAX, 385
 Robolectrica, 566
 środowiska Eclipse, 576
 WAP, Wireless Access Protocol, 357
 warstwa
 pośrednia, 33
 prezentacji, 40
 transportowa, 376

wartość
 @null, 173
 le6, 419
 wątek, 240, 262
 główny, 110
 interfejsu użytkownika, 244, 273
 konsumenta, 273
 obsługi, 245
 producenta, 273
 roboczy, 262
 usługowy, daemon thread, 272
 z pętlą, 276
 zegara, 272

wątki
 aplikacji, 110
 robocze, 244
 wczytywanie pliku PNG, 489
 WebKit, 58
 wiązanie aktywności z usługą, 212
 widok, 73, 139, 143
 android.view.SurfaceView, 464
 CanvasView, 484
 CheckBox, 181
 GLSurfaceView, 498
 GridView, 446
 ImageView, 260, 454
 ListView, 164
 LogCat, 275
 MapView, 429
 ShapesAndTextView, 488
 StackView, 657
 SurfaceView, 464
 TextView, 41
 VideoView, 464
 WebView, 678

- widoki
 - hierarchia, 140
 - listy, 170
 - nagłówka, 161
 - pomiary, 142
 - porządkowanie, 143
 - rozmieszczanie, 142
 - stopki, 162
 - wyświetlanie, 139, 141
 - złożone, 89
 - wielkość ekranu, 188
 - wielozadaniowość, 55, 199
 - wiersz poleceń, 62
 - wierzchołki trójkąta, 502
 - właściwości
 - Androida, 670
 - Javy, 670
 - nowego projektu, 37
 - projektu HelloAndroid, 37
 - systemowe, 670
 - właściwość
 - adb.device.arg, 628
 - Build Target, 37
 - Create Activity, 38
 - jar.libs.dir, 591
 - włączanie lampek LED, 219
 - WML, Wireless Markup Language, 357
 - wprowadzanie zależności, 560
 - współbieżność, 238, 244
 - współczynnik proporcji, 509
 - współdzielenie kontekstu, 341
 - współrzedne
 - geograficzne, 404
 - GPS, 672
 - przestrzenne, 510
 - znormalizowane, 514
 - współużytkowanie danych, 327–329, 341, 637
 - wstępne skalowanie, prescaling, 191
 - wtyczka, 706
 - ADT, 34, 39, 44, 608
 - Android Emulator Hudsona, 618, 619
 - Google'a, 36
 - Gradle, 606
 - Green Balls, 620
 - JDT, 608
 - m2eclipse, 607, 608, 610
 - m2eclipse--android-integration, 608
 - Mavena, 596, 607, 689
 - maven-android-plugin, 601
 - SBT, 606
 - wybieranie pliku, 451
 - wyciekanie pamięci, 263
 - wydajność parserów, 388
 - wyjątek
 - ANR, 237, 250
 - IOException, 394
 - NotFoundException, 608
 - RuntimeException, 570
 - SecurityException, 75, 410
 - wymagania
 - aplikacji, 639
 - funkcjonalne, 524
 - sprzętowe, 437
 - wynik testu, 533
 - wysokość, 407
 - wyszukiwanie, 448
 - wyszukiwanie numerów, 349
 - wyświetlacze, 187
 - wyświetlanie
 - bitmap, 489
 - celów, 588
 - cienia, 660
 - ekranów powitalnych, 268
 - elementów OverlayItems, 430
 - filmów, 462
 - losowego koloru, 479
 - oferty, 646
 - podłączonych urządzeń, 65
 - procesów, 112
 - tekstu, 485
 - usług, 205
 - widoków, 139, 141
 - zdjęć, 454
 - wywołanie
 - Anta, 629
 - metody HttpClient.execute, 239
 - zwrotne, 264
 - wywoływanie
 - asynchroniczne, 338
 - synchroniczne, 336
 - zdalne, 336
 - wywoływanie usługi, 213
 - wzorzec
 - DAO, 302
 - MVC, 96
 - ViewHolder, 160
- ## X
- XML, Extensible Markup Language, 42, 376
- ## Z
- zaciemnianie
 - kodu, 698
 - nazw metod, 700

- zadania
 - asynchroniczne, 256
 - testowe, test cases, 532, 534, 550
- zalety tabletów, 639
- zamknięcie procesu, 111
- zapis
 - danych, 295
 - pliku, 284, 287
- zarządzanie
 - akcjami, 215
 - bazami danych, 299
 - testami, 529
 - wątkami, 249
 - wątkami roboczymi, 265
 - zasilaniem, 226–228
 - zasobami, 638
- zasoby, 73, 76
 - DealDroid, 77
 - multimedialne, 441
- zasób układu, 40
- zawieszanie się aplikacji, 239
- zdalna usługa, 214
- zdalne wywołanie procedur, 335
- zdarzenie
 - BOOT_COMPLETED, 207
 - CONNECTIVITY_ACTION, 398
 - parsera SAX, 382
- zdjęcia, 466
- zegar, 271
- zgłaszanie zdarzeń, 571
- zintegrowane środowisko programowania, 34
- złączenia, 300

- zmiana
 - konfiguracji, 125
 - konfiguracji sieci, 399
 - orientacji, 125
 - sieci, 396
- zmienna środowiskowa \$PATH, 36, 61, 588
- zmniejszanie zużycia energii, 59
- znacznik
 - <merge>, 155
 - czasu, 571
 - kierunkowy, directional tag, 209
 - usługi, 204
- znak
 - @, 41
 - +, 151
 - zachęty, 53
- znaki @+id, 151
- zrzut stosu, 699

Ż

- żądanie
 - GET, 368
 - HTTP, 359
 - rejestracji, 231
- żyroskop, 58

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

Nietrudno jest znaleźć informacje potrzebne do stworzenia pierwszej aplikacji na Androida. Ale co dalej? Im głębiej zanurzymy się w świat urządzeń mobilnych, tym częściej będziemy trafiać na pułapki i ślepe uliczki. Na ratunek przychodzi książka *Android w praktyce*. Nie znajdziesz tu jednak żadnych banalnych informacji. Ta książka to bogate źródło wskazówek, sztuczek i najlepszych praktyk z obszaru tworzenia aplikacji na Androida, obejmująca ponad dziewięćdziesiąt pomysłowych i przydatnych technik, dzięki którym staniesz się lepszym programistą.

W trakcie lektury zobaczysz, jak tworzyć precyzyjne (co do piksela) elementy graficzne, zarządzać zadaniami wykonywanymi w tle oraz równoległymi wątkami. Ponadto sprawdzisz, jak współużytkować dane między aplikacjami oraz komunikować się z usługami sieciowymi. To tylko niektóre z tematów poruszonych w tej wyjątkowej książce, poświęconej platformie Android. Przeznaczona zarówno dla początkujących, jak i zaawansowanych użytkowników, pomoże Ci ona zrozumieć, jak budować doskonałe aplikacje, które przez lata będą odnosić sukcesy na platformie Android.

Sięgnij po tę książkę i...

- poznaj najlepsze praktyki tworzenia aplikacji na platformę Android
- wykorzystaj potencjał wielordzeniowych procesorów
- twórz grafikę dwu- i trzymiarymową
- znajdź rozwiązania Twoich problemów

Obowiązkowa pozycja na półce programisty aplikacji mobilnych!

Charlie Collins jest programistą aplikacji mobilnych i sieciowych w firmie MOVIL, współtwórcą kilku projektów o otwartym dostępie do kodu źródłowego, a także współautorem książek *GWT in Practice* i *Unlocking Android*.

Michael Galpin jest programistą w firmie Bump Technologies, gdzie pracował nad dwiema spośród najbardziej popularnych aplikacji ze sklepu Android Market (Bump i eBay Mobile).

Matthias Kaeppler jest inżynierem odpowiedzialnym za Androida i interfejsy API w firmie Qype.

helion.pl
księgarnia
internetowa

Nr katalogowy: 10480



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

• <http://helion.pl/novosci>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-246-4810-8



9 788324 648108

Cena: 99,00 zł

Informatyka w najlepszym wydaniu