

Joseph Annuzzi Jr.
Lauren Darcey
Shane Conder



Android™

Wprowadzenie do programowania aplikacji

Wydanie V

The Helion logo, featuring the word "Helion" in white text on a blue rectangular background, followed by a white icon of a stylized 'H' or a similar symbol.

Helion



Tytuł oryginału: Introduction to Android Application Development: Android Essentials, Fifth Edition

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-2612-5

Authorized translation from the English language edition, entitled: Introduction TO ANDROID APPLICATION DEVELOPMENT: ANDROID ESSENTIALS, Fifth Edition; ISBN 013438945X; by Joseph Annuzzi; and by Lauren Darcey; and by Shane Conder; published by Pearson Education, Inc, publishing as Addison-Wesley Professional.

Copyright © 2016 Joseph Annuzzi, Jr., Lauren Darcey, and Shane Conder

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 Pearson Education, Inc.

Polish language edition published by HELION SA. Copyright © 2016.

The following are registered trademarks of Google:

Android™, Chrome™, Google Play™, Nexus™, Dalvik™, Google Maps™, Google+™, Google TV™, Google and the Google logo are registered trademarks of Google Inc.

ARM is a registered trademark of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

Altium® and Altium Designer® are trademarks or registered trademarks of Altium Limited or its subsidiaries.

Qualcomm and Snapdragon are trademarks of Qualcomm Incorporated, registered in the United States and other countries.

Cyanogen is a trademark of Cyanogen Inc., registered in certain countries.

CyanogenMod is a trademark of CyanogenMod, LLC, registered in the United States.

JetBrains® and IntelliJ®, are registered trademarks owned by JetBrains s.r.o.

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/anwpp5>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

O autorach	25
Wprowadzenie	27
Kto powinien przeczytać tę książkę	27
Kluczowe pytania, na jakie odpowiada ta książka	28
Struktura książki	29
Opis zmian wprowadzonych w tym wydaniu książki	30
Środowiska programistyczne wykorzystane w tej książce	32
Dostępne materiały dodatkowe	33
Konwencje stosowane w książce	33
Gdzie szukać dodatkowych informacji	34
Kontakt z autorami	35
Część I	
Ogólne informacje o platformie Android	37
Rozdział 1	
Prezentacja systemu Android	39
Android Open Source Project (AOSP)	39
Open Handset Alliance	40
Google staje się mobilny	40
Prezentacja Open Handset Alliance	40
Dołączanie do Open Handset Alliance	41
Producenci — projektowanie urządzeń dla Androida	41
Operatorzy — dostarczanie wrażeń	42
Aplikacje napędzają sprzedaż urządzeń: tworzenie aplikacji na Androida	43
Wykorzystanie wszystkich możliwości Androida	43
Android: stan obecny	43
Co wyróżnia platformę Android?	44
Android — nazwy kodowe	45
Darmowy i otwarty	45
Znane i niedrogie narzędzia programistyczne	46
Rozsądny stopień trudności nauki programowania	47

Umożliwianie tworzenia potężnych aplikacji	47
Bogate i bezpieczne możliwości integracji aplikacji	47
Brak kosztownych przeszkód utrudniających publikację	48
„Otwarty rynek” aplikacji	48
Rozwijająca się platforma	49
Platforma Android	50
Architektura Androida	50
Bezpieczeństwo i uprawnienia	52
Pisanie aplikacji na Androida	53
Android poza OHA i GMS	56
Fire OS firmy Amazon	57
Cyanogen OS i CyanogenMod	57
Maker Movement oraz Open-Source Hardware	57
Szersze spojrzenie	58
Podsumowanie	58
Pytania kwizowe	59
Ćwiczenia	59
Odwołania i inne źródła informacji	59
Rozdział 2 Przygotowywanie środowiska programistycznego	61
Konfiguracja środowiska programistycznego	61
Konfiguracja własnego systemu do debugowania urządzenia	64
Konfiguracja urządzenia do debugowania	65
Aktualizacja Android Studio	66
Aktualizacja Android SDK	67
Problemy z Android Studio	67
Problemy z Android SDK	68
IntelliJ IDEA jako alternatywa dla Android Studio	68
Poznajemy Android SDK	69
Zrozumienie postanowień licencji	69
Korzystanie z dokumentacji Android SDK	70
Prezentacja szkieletu programowania aplikacji	70
Poznanie narzędzi Android SDK	73
Poznanie aplikacji przykładowych	76
Podsumowanie	79
Pytania kwizowe	79
Ćwiczenia	79
Odwołania i inne źródła informacji	80
Rozdział 3 Pierwsza aplikacja na Androida	81
Testowanie środowiska programistycznego	81
Importowanie aplikacji przykładowej BorderlessButtons do Android Studio	82

Korzystanie z preinstalowanego AVD do uruchamiania aplikacji BorderlessButtons	84
Uruchamianie aplikacji BorderlessButtons w emulatorze Androida	86
Budowanie pierwszej aplikacji na Androida	88
Tworzenie i konfiguracja nowego projektu aplikacji	89
Wyjaśnienie symbolicznego widoku Android oraz klasycznego widoku Project	93
Podstawowe pliki i katalogi aplikacji na Androida	94
Uruchamianie aplikacji w emulatorze	95
Debugowanie aplikacji w emulatorze	96
Dodawanie mechanizmów rejestracji do aplikacji	100
Debugowanie aplikacji na fizycznym urządzeniu	101
Podsumowanie	104
Pytania kwizowe	105
Ćwiczenia	106
Odwołania i inne źródła informacji	106

Część II Kluczowe informacje o konstrukcji aplikacji na Androida107

Rozdział 4 Prezentacja komponentów aplikacji109

Opanowanie najważniejszej terminologii	109
Kontekst aplikacji	110
Pobieranie kontekstu aplikacji	111
Stosowanie kontekstu aplikacji	111
Realizacja działań przy użyciu aktywności	112
Cykl życia aktywności	113
Organizowanie komponentów aplikacji przy użyciu fragmentów	118
Zarządzanie zmianami aktywności przy użyciu intencji	120
Przechodzenie pomiędzy aktywnościami przy użyciu intencji	120
Organizacja aktywności, fragmentów i intencji w menu nawigacyjnym aplikacji	122
Praca z usługami	123
Odbieranie oraz rozgłaszanie intencji	124
Podsumowanie	125
Pytania kwizowe	125
Ćwiczenia	126
Odwołania i inne źródła informacji	126

Rozdział 5 Definiowanie manifestu127

Konfiguracja aplikacji na Androida przy użyciu pliku manifestu	127
Edycja pliku manifestu	128
Zarządzanie tożsamością aplikacji	130
Określanie nazwy oraz ikony aplikacji	131

Określanie wymagań systemowych aplikacji	131
Ograniczenia związane z platformą	132
Ustawienia związane z konfiguracją innych aplikacji i filtrów	134
Rejestracja aktywności w pliku manifestu	134
Określanie aktywności będącej głównym punktem wejścia aplikacji przy użyciu filtra intencji	135
Konfiguracja innych filtrów intencji	135
Rejestracja innych komponentów aplikacji	136
Stosowanie uprawnień	136
Rejestracja uprawnień wymaganych przez aplikację	137
Rejestracja uprawnień definiowanych przez aplikację	139
Poznananie innych ustawień podawanych w manifestie	140
Podsumowanie	141
Pytania kwizowe	141
Ćwiczenia	141
Odwołania i inne źródła informacji	142
Rozdział 6 Zarządzanie zasobami aplikacji	143
Czym są zasoby?	143
Przechowywanie zasobów aplikacji	144
Typy wartości zasobów	144
Programowy dostęp do zasobów	148
Określanie wartości prostych zasobów w Android Studio	148
Praca z różnymi typami zasobów	152
Łańcuchy znaków	152
Stosowanie łańcuchów z zasobów do formatowania tekstów	153
Praca z łańcuchami wyrażającymi liczbę mnogą	154
Praca z tablicami łańcuchów znaków	155
Praca z wartościami logicznymi	156
Praca z liczbami całkowitymi	157
Praca z kolorami	157
Praca z wymiarami	158
Programowe stosowanie zasobów z wymiarami	159
Praca z zasobami graficznymi	159
Praca z obrazami	161
Praca z listami stanów kolorów	163
Praca z animacjami	164
Praca z menu	166
Praca z plikami XML	168
Praca z nieprzetworzonymi plikami	168
Odwołania do zasobów	169
Praca z układami	170
Projektowanie układów w Android Studio	172
Programowe korzystanie z zasobów definiujących układy	174

Odwołania do zasobów systemowych	175
Podsumowanie	176
Pytania kwizowe	176
Ćwiczenia	177
Odwołania i inne źródła informacji	177
Rozdział 7 Prezentacja elementów interfejsu użytkownika	179
Prezentacja widoków i układów	179
Widoki	179
Kontrolki systemu Android	179
Prezentacja układów	180
Wyświetlanie tekstów przy użyciu TextView	181
Konfiguracja układu oraz określanie wymiarów	181
Umieszczanie w tekście kontekstowych odnośników	182
Pobieranie danych od użytkowników za pomocą pól tekstowych	184
Pobieranie danych przy użyciu kontrolki EditText	185
Ograniczanie możliwości wprowadzania danych z zastosowaniem filtrów	186
Wspomaganie wpisywania przy użyciu automatycznego uzupełniania	187
Kontrolka Spinner — zapewnianie możliwości wyboru	189
Stosowanie przycisków i przełączników	191
Stosowanie zwyczajnych przycisków	192
Stosowanie pól wyboru i przełączników	194
Stosowanie kontrolki RadioGroup oraz RadioButton	195
Pobieranie daty, godziny i liczb	197
Prezentacja postępów i aktywności przy użyciu wskaźników	199
Prezentacja postępów za pomocą paska postępu	199
Sygnalizowanie aktywności za pomocą pasków aktywności oraz kołowych znaczników aktywności	202
Modyfikacja postępu przy użyciu kontrolki SeekBar	202
Inne użyteczne kontrolki interfejsu użytkownika	203
Wyświetlanie oceny przy użyciu kontrolki RatingBar	203
Prezentacja upływu czasu za pomocą minutnika	204
Wyświetlanie czasu	205
Odtwarzanie wideo przy użyciu kontrolki VideoView	206
Podsumowanie	207
Pytania kwizowe	208
Ćwiczenia	208
Odwołania i inne źródła informacji	208
Rozdział 8 Umiejscawianie elementów z użyciem układów	211
Tworzenie interfejsów użytkownika w systemie Android	211
Definiowanie układów w zasobach aplikacji	211
Programowe tworzenie układów	213

Organizacja interfejsu użytkownika aplikacji	215
Stosowanie klas potomnych ViewGroup do projektowania układów	216
Stosowanie klas potomnych ViewGroup jako pojemników	216
Stosowanie wbudowanych układów	217
Układ LinearLayout	218
Układ RelativeLayout	220
Układ FrameLayout	223
Układ TableLayout	226
Stosowanie układu GridLayout	228
Stosowanie wielu układów jednocześnie	231
Stosowanie wbudowanych klas pojemników	232
Pojemniki działające na podstawie danych	232
Dodawanie możliwości przewijania	237
Prezentacja innych rodzajów pojemników	237
Podsumowanie	239
Pytania kwizowe	239
Ćwiczenia	239
Odwołania i inne źródła informacji	240
Rozdział 9 Dzielenie interfejsu aplikacji z użyciem fragmentów	243
Wyjaśnienie pojęcia fragmentu	243
Opis cyklu życia obiektów Fragment	245
Zarządzanie modyfikacjami fragmentów	246
Stosowanie specjalnych typów fragmentów	247
Projektowanie aplikacji korzystających z fragmentów	248
Stosowanie pakietu biblioteki wsparcia	256
Dodawanie wsparcia dla fragmentów do starych aplikacji	256
Stosowanie fragmentów w nowych aplikacjach przeznaczonych dla starszych wersji platformy	257
Dołączanie pakietu Support Library do aplikacji	257
Dodatkowe sposoby stosowania fragmentów	259
Fragmenty funkcjonalne pozbawione interfejsu użytkownika	259
Poznanie zagnieżdżonych fragmentów	259
Podsumowanie	259
Pytania kwizowe	260
Ćwiczenia	260
Odwołania i inne źródła informacji	261

Część III Kluczowe zagadnienia projektowania aplikacji263

Rozdział 10 Określanie architektury z użyciem wzorców265

Określanie architektury nawigacji	265
Scenariusze nawigacyjne aplikacji na Androida	266
Uruchamianie zadań i poruszanie się po stosie cofnięć	269
Nawigacja a fragmenty	269
Relacje pomiędzy ekranami	269
Wzorce projektowe nawigacji stosowane w Androidzie	270
Zachęcanie do wykonywania akcji	276
Menu	277
Paski akcji	278
Pływający przycisk akcji	282
Akcje zależne od zawartości aplikacji	282
Okna dialogowe	283
Podsumowanie	291
Pytania kwizowe	291
Ćwiczenia	292
Odwołania i inne źródła informacji	292

Rozdział 11 Stosowanie stylów do poprawy wizualnej atrakcyjności aplikacji295

Style ze wsparciem bibliotek	295
Motywy i style	296
Definiowanie domyślnego motywu aplikacji	296
Dziedziczenie motywów i stylów	297
Kolory	298
Układ	300
Elementy <merge> i <include>	300
Widżet TextInputLayout	301
Przycisk FloatingActionButton	301
Widżet Toolbar jako dolny pasek aplikacji	302
Określanie markowego wyglądu aplikacji	303
Separatory i odstępy	304
Menu	305
Uzyskane wyniki	305
Typografia	305
Podsumowanie	306
Pytania kwizowe	307
Ćwiczenia	307
Odwołania i inne źródła informacji	308

Rozdział 12 Stosowanie Material Design	311
Zrozumienie Material Design	311
Domyślny motyw Material	312
Aplikacja SampleMaterial	312
Implementacja aplikacji SampleMaterial	312
Zależności	313
Style wspomagające Material Design	313
Wyświetlanie zbioru danych na liście	314
Podsumowanie	336
Pytania kwizowe	336
Ćwiczenia	336
Odwołania i inne źródła informacji	337
Rozdział 13 Projektowanie zgodnych aplikacji	339
Maksymalizacja zgodności aplikacji	339
Projektowanie interfejsów użytkownika pod kątem zgodności	342
Stosowanie fragmentów	343
Stosowanie API dostępnych w bibliotekach wsparcia	344
Wsparcie dla konkretnych typów ekranów	344
Stosowanie elastycznej grafiki typu Nine-Patch	345
Wykorzystywanie zasobów alternatywnych	345
Przedstawienie sposobu wyznaczania zasobów alternatywnych	346
Organizowanie zasobów alternatywnych z użyciem kwalifikatorów	347
Stosowanie zasobów dla różnych orientacji	353
Programowe stosowanie zasobów alternatywnych	354
Efektywna organizacja zasobów aplikacji	354
Przygotowywanie aplikacji dla tabletów i telewizorów	357
Aplikacje na tablety	357
Aplikacje na telewizory	359
Rozszerzanie zasięgu aplikacji na zegarki i samochody	360
Zapewnianie zgodności z SafetyNet	361
Podsumowanie	362
Pytania kwizowe	363
Ćwiczenia	363
Odwołania i inne źródła informacji	363

Część IV Kluczowe zagadnienia programowania aplikacji na Androida	365
Rozdział 14 Stosowanie preferencji	367
Korzystanie z preferencji aplikacji	367
Określanie, kiedy stosowanie preferencji jest właściwe	367
Zapisywanie w preferencjach wartości różnych typów	368
Tworzenie prywatnych preferencji	368
Tworzenie wspólnych preferencji używanych przez większą liczbę aktywności	368
Przeszukiwanie i odczyt preferencji	369
Dodawanie, aktualizacja oraz usuwanie preferencji	369
Reagowanie na zmiany w preferencjach	371
Odnajdywanie danych preferencji w systemie plików Androida	371
Tworzenie łatwych do zarządzania preferencji użytkownika	372
Tworzenie pliku zasobów preferencji	372
Stosowanie klasy PreferenceActivity	374
Organizowanie preferencji dzięki wykorzystaniu nagłówków	376
Automatyczna kopia zapasowa aplikacji na Androida	380
Podsumowanie	381
Pytania kwizowe	381
Ćwiczenia	381
Odwołania i inne źródła informacji	382
Rozdział 15 Dostęp do plików i katalogów	383
Korzystanie z danych aplikacji na urządzeniu	383
Dobre praktyki związane z zarządzaniem plikami	384
Wyjaśnienie kwestii uprawnień do plików w Androidzie	385
Praca z plikami i katalogami	386
Badanie katalogów aplikacji	386
Praca z innymi katalogami i plikami w systemie plików Androida	391
Podsumowanie	394
Pytania kwizowe	394
Ćwiczenia	394
Odwołania i inne źródła informacji	395
Rozdział 16 Zapisywanie informacji w bazach danych SQLite	397
Dodawanie bazy SQLite do aplikacji SampleMaterial	397
Praca z bazami danych	398
Zapewnianie dostępu do danych	399
Aktualizacja klasy SampleMaterialActivity	400
Aktualizacja konstruktora klasy SampleMaterialAdapter	401
Usuwanie operacji na bazie z głównego wątku interfejsu użytkownika	401

Tworzenie kart w bazie danych	402
Pobieranie wszystkich kart	403
Dodawanie nowej karty	404
Aktualizacja kart	405
Usuwanie karty	406
Podsumowanie	407
Pytania kwizowe	408
Ćwiczenia	408
Odwołania i inne źródła informacji	408
Rozdział 17 Stosowanie dostawców treści	411
Prezentacja dostawców treści	411
Stosowanie dostawcy treści MediaStore	412
Stosowanie dostawcy danych CallLog	414
Korzystanie z dostawcy treści CalendarContract	416
Stosowanie dostawcy treści UserDictionary	417
Korzystanie z dostawcy treści VoicemailContract	417
Stosowanie dostawcy treści Settings	417
Prezentacja dostawcy treści ContactsContract	417
Modyfikacja danych dostawców treści	419
Dodawanie rekordów	419
Aktualizacja rekordów	421
Usuwanie rekordów	421
Korzystanie z dostawców treści innych firm	422
Podsumowanie	423
Pytania kwizowe	423
Ćwiczenia	423
Odwołania i inne źródła informacji	424
 Część V Kluczowe zagadnienia	
rozpowszechniania aplikacji	425
 Rozdział 18 Proces tworzenia oprogramowania mobilnego	427
Prezentacja procesu tworzenia oprogramowania mobilnego	427
Wybór metodologii tworzenia oprogramowania	428
Zrozumienie niebezpieczeństw metody kaskadowej	428
Zrozumienie znaczenia powtarzania	429
Gromadzenie wymagań aplikacji	429
Określanie wymagań projektowych	429
Tworzenie przypadków użycia aplikacji na Androida	432
Dołączanie wymagań i zaleceń innych podmiotów	433
Zarządzanie bazą danych urządzeń	433

Szacowanie ryzyka związanego z projektem	437
Określanie urządzeń docelowych	437
Pozyskiwanie urządzeń docelowych	439
Określanie możliwości zaspokojenia wymagań aplikacji	440
Rozumienie ryzyka związanego z zapewnianiem jakości	440
Pisanie ważnej dokumentacji projektowej	442
Tworzenie planów testowania na potrzeby kontroli jakości	442
Dostarczanie dokumentacji wymaganej przez inne podmioty	443
Dokumentacja na potrzeby utrzymania i przenoszenia	443
Korzystanie z systemów zarządzania konfiguracjami	443
Wybór systemu zarządzania kodem źródłowym	443
Implementacja działającego systemu numeracji wersji aplikacji	444
Projektowanie aplikacji na Androida	445
Znajomość ograniczeń urządzeń z Androidem	445
Poznanie wspólnych architektur aplikacji na Androida	445
Projektowanie aplikacji pod kątem jej rozszerzania i pielęgnacji	446
Projektowanie pod kątem możliwości współdziałania aplikacji	447
Tworzenie aplikacji na Androida	448
Testowanie aplikacji na Androida	448
Kontrola wersji testowych aplikacji	449
Wdrażanie aplikacji na Androida	450
Określanie rynków docelowych	450
Wsparcie i pielęgnacja aplikacji na Androida	450
Śledzenie i weryfikacja informacji o awariach	451
Testowanie aktualizacji oprogramowania układowego	451
Prowadzenie odpowiedniej dokumentacji aplikacji	451
Wprowadzanie zmian na działającym serwerze	452
Określanie możliwości przenoszenia aplikacji obciążonego niewielkim ryzykiem	452
Selekcja możliwości aplikacji	452
Podsumowanie	452
Pytania kwizowe	453
Ćwiczenia	453
Odwołania i inne źródła informacji	453
Rozdział 19 Planowanie doświadczeń użytkowników	455
Myślenie o celach	455
Cele użytkowników	456
Cele twórców aplikacji	456
Cele innych zainteresowanych stron	457
Techniki konkretyzowania wysiłków wkładanych w rozwój projektu	457
Persony	457
Mapowanie historii użytkowników	458

Wykrywanie i organizacja encji	458
Planowanie interakcji użytkowników	459
Wyrażanie tożsamości aplikacji	460
Projektowanie układów ekranów	462
Szkice	462
Szkielety	462
Kompozycje projektowe	462
Właściwe reagowanie z wykorzystaniem wizualnych informacji zwrotnych	463
Obserwowanie docelowej grupy odbiorców w celu poprawy użyteczności aplikacji	463
Tworzenie atrapy aplikacji	464
Testowanie wersji finalnej aplikacji	465
Podsumowanie	465
Pytania kwizowe	465
Ćwiczenia	466
Odwotania i inne źródła informacji	466

Rozdział 20 Projektowanie i tworzenie niezawodnych aplikacji na Androida467

Najlepsze praktyki projektowania niezawodnych aplikacji na Androida	467
Zaspokajanie wymagań użytkowników urządzeń z Androidem	468
Projektowanie interfejsu użytkownika aplikacji na Androida	468
Projektowanie stabilnych i szybko reagujących aplikacji mobilnych	470
Projektowanie bezpiecznych aplikacji na Androida	472
Projektowanie aplikacji na Androida w celu maksymalizacji zysków	473
Korzystanie z wytycznych dotyczących zachowania jakości podczas projektowania aplikacji na Androida	474
Stosowanie standardów jakości firm trzecich	475
Projektowanie aplikacji pod kątem prostoty ich utrzymania i aktualizacji	476
Projektowanie aplikacji przy wykorzystaniu narzędzi Androida	478
Unikanie głupich błędów podczas projektowania aplikacji na Androida	478
Najlepsze praktyki stosowane przy tworzeniu wysokiej jakości aplikacji na Androida	479
Określanie procesu produkcyjnego dostosowanego do tworzenia oprogramowania mobilnego	479
Wczesne i częste testowanie możliwości wykonania projektu	480
Stosowanie standardów kodowania, weryfikacji i testów jednostkowych w celu poprawienia jakości kodu	480
Obsługa usterek występujących na jednym urządzeniu	483

Korzystanie z narzędzi Androida przy pisaniu aplikacji	484
Unikanie głupich błędów podczas tworzenia aplikacji na Androida	484
Podsumowanie	485
Pytania kwizowe	485
Ćwiczenia	485
Odwołania i inne źródła informacji	486
Rozdział 21 Testowanie aplikacji na Androida	487
Najlepsze praktyki testowania oprogramowania mobilnego	487
Projektowanie systemu rejestracji defektów na potrzeby tworzenia oprogramowania mobilnego	487
Zarządzanie środowiskiem testowym	489
Maksymalizacja pokrycia testów	492
Stosowanie narzędzi Android SDK do testowania aplikacji na Androida	500
Unikanie głupich błędów podczas testowania aplikacji na Androida	502
Podstawowe informacje o testowaniu aplikacji na Androida	502
Testy jednostkowe z użyciem JUnit	503
Prezentacja aplikacji PasswordMatcher	504
Określanie, czego powinny dowieść testy jednostkowe	507
Tworzenie konfiguracji uruchomieniowej na potrzeby kodu testowego	507
Pisanie testów	511
Wykonywanie pierwszego testu z wykorzystaniem narzędzi Android Studio	513
Analiza wyników testów	513
Dodawanie kolejnych testów	514
Inne programy i API do automatyzacji testów aplikacji na Androida	517
Podsumowanie	518
Pytania kwizowe	519
Ćwiczenia	519
Odwołania i inne źródła informacji	519
Rozdział 22 Rozpowszechnianie aplikacji na Androida	521
Wybór odpowiedniego modelu dystrybucji	521
Ochrona swojej własności intelektualnej	522
Zachowanie zgodności z regulaminem Google Play	523
Pobieranie opłat od użytkowników	523
Przygotowywanie aplikacji do publikacji	525
Przygotowanie kodu do utworzenia pakietu instalacyjnego	525
Tworzenie pakietu aplikacji i jego podpisywanie	527
Testowanie publikowanej wersji pakietu aplikacji	531
Dołączanie wszystkich niezbędnych zasobów	531

Przygotowanie serwerów i usług	531
Dystrybucja aplikacji	531
Publikowanie aplikacji w Google Play	532
Rejestracja w Google Play w celu publikowania aplikacji	532
Przesyłanie aplikacji do Google Play	535
Przesyłanie materiałów marketingowych aplikacji	537
Konfiguracja szczegółowych informacji dotyczących opłat oraz dystrybucji aplikacji	537
Konfigurowanie innych opcji aplikacji	539
Zarządzanie pozostałymi opcjami Developer Console	539
Publikowanie aplikacji w Google Play	539
Zarządzanie aplikacją w sklepie Google Play	540
Mechanizm wdrażania etapami	541
Publikowanie aplikacji w prywatnym kanale Google Play	542
Tłumaczenie aplikacji	542
Publikowanie aplikacji w alternatywnych kanałach dystrybucji	543
Samodzielne publikowanie aplikacji	544
Podsumowanie	545
Pytania kwizowe	546
Ćwiczenia	546
Odwołania i inne źródła informacji	546

Dodatki 549

Dodatek A Wskazówki i sztuczki: Android Studio 551

Organizacja przestrzeni roboczej w Android Studio	551
Integracja z usługami kontroli kodów źródłowych	551
Zmiana położenia okien w Android Studio	552
Zmiana wielkości okna edytora	552
Zmiana wielkości okna Tools	553
Wyświetlanie okien edytora jedno przy drugim	553
Wyświetlanie dwóch fragmentów tego samego pliku	553
Zamykanie niepotrzebnych kart	554
Zachowywanie kontroli nad oknami edytora	555
Tworzenie niestandardowych filtrów dzienników	556
Przeszukiwanie projektu	557
Organizowanie zadań Android Studio	558
Pisanie kodu w Javie	558
Stosowanie automatycznego uzupełniania	558
Tworzenie nowych klas i metod	559
Organizowanie instrukcji importu	559
Formatowanie kodu	559
Możliwość modyfikowania niemal wszystkich nazw	560
Refaktoryzacja kodu	560

Reorganizacja kodu	562
Narzędzie Intention Actions	562
Przygotowywanie dokumentacji Javadoc	563
Rozwiązywanie tajemniczych błędów budowy	563
Podsumowanie	563
Pytania kwizowe	563
Ćwiczenia	564
Odwolania i inne źródła informacji	564
Dodatek B Krótki przewodnik po emulatorze Androida	565
Symulacja rzeczywistości — przeznaczenie emulatora	565
Korzystanie z różnych urządzeń wirtualnych (AVD)	567
Stosowanie programu Android Virtual Device Manager	568
Tworzenie AVD	570
Tworzenie AVD z niestandardowymi ustawieniami komponentów sprzętowych	574
Uruchamianie emulatora z użyciem konkretnego AVD	575
Zapewnianie wydajności pracy emulatora	576
Uruchamianie emulatora w celu wykonania aplikacji	577
Uruchamianie emulatora z poziomu programu Android Virtual Device Manager	580
Konfiguracja położenia GPS w emulatorze	581
Symulowanie przychodzących połączeń telefonicznych na emulatorze Androida	583
Przesyłanie SMS-ów do emulatora Androida	584
Interakcja z emulatorem z poziomu konsoli	585
Wykorzystanie konsoli do symulowania odbieranych połączeń	586
Stosowanie konsoli do symulowania wiadomości SMS	587
Stosowanie konsoli do przesyłania współrzędnych GPS	589
Stosowanie konsoli do monitorowania transmisji sieciowych	590
Stosowanie konsoli do modyfikowania ustawień zasilania	590
Inne polecenia konsoli emulatora	591
Personalizacja emulatora	591
Ograniczenia emulatora	592
Podsumowanie	593
Pytania kwizowe	593
Ćwiczenia	594
Odwolania i inne źródła informacji	594
Dodatek C Krótki przewodnik po programie Device Monitor	595
Korzystanie z programu Device Monitor w Android Studio oraz jako niezależnej aplikacji	595
Szybka prezentacja kluczowych możliwości Device Monitora	597

Obsługa procesów, wątków i sterty	597
Dołączanie debugera do aplikacji	598
Zatrzymywanie procesu	598
Monitorowanie aktywności wątku aplikacji	598
Monitorowanie operacji wykonywanych na sterce	599
Wymuszenie oczyszczenia pamięci	600
Tworzenie i stosowanie plików HPROF	601
Stosowanie karty Allocation Tracker	602
Przeglądanie statystyk wykorzystania sieci	603
Zarządzanie plikami	604
Przeglądanie systemu plików w emulatorze lub na urządzeniu ...	605
Kopiowanie plików z emulatora lub urządzenia	606
Kopiowanie plików do emulatora lub urządzenia	606
Usuwanie plików na emulatorze lub urządzeniu	607
Stosowanie zakładki Emulator Control	607
Zmiana stanu telefonii	607
Symulowanie przychodzących połączeń telefonicznych	608
Symulowanie nadsyłanych wiadomości SMS	609
Przesyłanie współrzędnych geograficznych	609
Korzystanie z karty System Information	609
Robienie zrzutów ekranu z emulatora i rzeczywistych urządzeń	610
Korzystanie z mechanizmów rejestracji komunikatów	611
Podsumowanie	612
Pytania kwizowe	612
Ćwiczenia	613
Odwołania i inne źródła informacji	613
Dodatek D Kurs mistrzowski: narzędzia Android SDK	615
Stosowanie dokumentacji Androida	615
Korzystanie z emulatora Androida	619
Przeglądanie dzienników aplikacji z użyciem narzędzia logcat	620
Debugowanie aplikacji z użyciem monitora urządzenia	620
Stosowanie ADB	621
Stosowanie edytora układów	622
Stosowanie podglądu hierarchii	622
Uruchamianie narzędzia Hierarchy Viewer	624
Stosowanie trybu Layout View	624
Optymalizacja interfejsu użytkownika	625
Stosowanie trybu Pixel Perfect	626
Stosowanie formatu graficznego Nine-Patch Stretchable Graphics ...	626
Korzystanie z innych narzędzi Android SDK	628

Podsumowanie	631
Pytania kwizowe	632
Ćwiczenia	632
Odwołania i inne źródła informacji	632
Dodatek E Krótki przewodnik po systemie budowy Gradle	635
Pliki budowy Gradle	635
Ustawienia projektu	637
Ustawienia modułów	637
Stosowanie Android Studio	
do konfigurowania procesu budowy aplikacji	641
Synchronizacja projektu	641
Konfiguracja właściwości Androida	642
Określanie opcji podpisywania	643
Konfiguracja budowy różnych wersji aplikacji	644
Konfigurowanie różnych typów budowy	645
Konfiguracja zależności aplikacji	647
Dodawanie zależności od bibliotek	647
Budowanie różnych wersji plików APK	648
Wykonywanie różnych zadań budowy Gradle	650
Podsumowanie	652
Pytania kwizowe	652
Ćwiczenia	652
Odwołania i inne źródła informacji	653
Dodatek F Odpowiedzi na pytania kwizowe	655
Rozdział 1. „Prezentacja systemu Android”	655
Rozdział 2. „Przygotowywanie środowiska programistycznego”	655
Rozdział 3. „Pierwsza aplikacja na Androida”	656
Rozdział 4. „Prezentacja komponentów aplikacji”	656
Rozdział 5. „Definiowanie manifestu”	656
Rozdział 6. „Zarządzanie zasobami aplikacji”	656
Rozdział 7. „Prezentacja elementów interfejsu użytkownika”	657
Rozdział 8. „Umiejscawianie elementów z użyciem układów”	657
Rozdział 9. „Dzielenie interfejsu aplikacji z użyciem fragmentów”	657
Rozdział 10. „Określanie architektury z użyciem wzorców”	658
Rozdział 11. „Stosowanie stylów	
do poprawy wizualnej atrakcyjności aplikacji”	658
Rozdział 12. „Stosowanie Material Design”	658
Rozdział 13. „Projektowanie zgodnych aplikacji”	659
Rozdział 14. „Stosowanie preferencji”	659
Rozdział 15. „Dostęp do plików i katalogów”	659
Rozdział 16. „Zapisywanie informacji w bazach danych SQLite”	659
Rozdział 17. „Stosowanie dostawców treści”	660

Rozdział 18. „Proces tworzenia oprogramowania mobilnego”	660
Rozdział 19. „Planowanie doświadczeń użytkowników”	660
Rozdział 20. „Projektowanie i tworzenie niezawodnych aplikacji na Androida”	661
Rozdział 21. „Testowanie aplikacji na Androida”	661
Rozdział 22. „Rozpowszechnianie aplikacji na Androida”	662
Dodatek A. „Wskazówki i sztuczki: Android Studio”	662
Dodatek B. „Krótki przewodnik po emulatorze Androida”	662
Dodatek C. „Krótki przewodnik po programie Device Monitor”	663
Dodatek D. „Kurs mistrzowski: narzędzia Android SDK”	663
Dodatek E. „Krótki przewodnik po systemie budowy Gradle”	663
Skorowidz	665

Dzielenie interfejsu aplikacji z użyciem fragmentów

Tradycyjnie każdy ekran w aplikacji przeznaczonej na Androida był skojarzony z jedną, konkretną aktywnością. Niemniej jednak w systemie Android 3.0 (Honeycomb) wprowadzono pojęcie fragmentu (reprezentowanego przez klasę `Fragment`). Następnie obsługę fragmentów dodano do biblioteki wsparcia systemu Android, tak by można ich było używać w Androidzie, zaczynając od wersji 1.6 (API poziomu 4). Fragmenty zapewniają separację komponentów interfejsu użytkownika lub zachowań (zarówno tych dysponujących interfejsem użytkownika, jak i tych, które są go pozbawione) od cyklu życia konkretnej aktywności. Z kolei klasy aktywności mogą dowolnie łączyć i stosować komponenty interfejsu użytkownika bądź zachowania, tworząc z ich pomocą znacznie bardziej elastyczny interfejs użytkownika. W tym rozdziale zostanie wyjaśnione, czym są fragmenty oraz jak można je tworzyć i stosować. Przedstawione zostanie także zagadnienie fragmentów zagnieżdżonych.

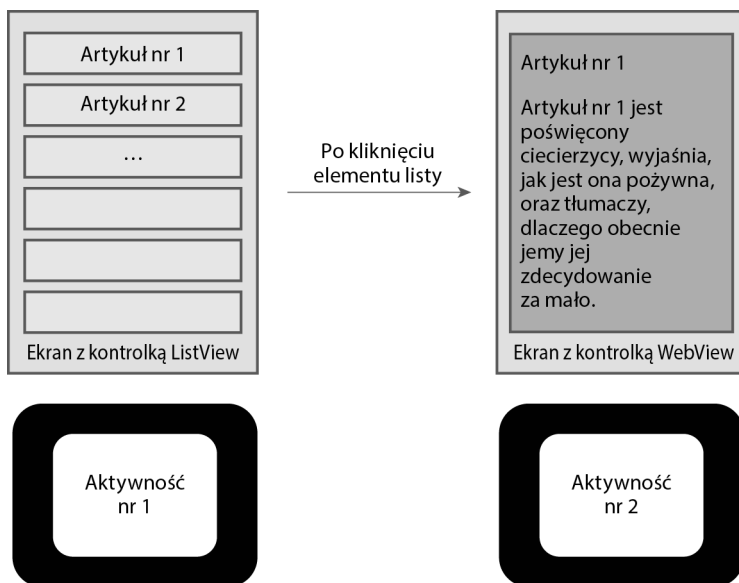
Wyjaśnienie pojęcia fragmentu

Fragmenty zostały dodane do Android SDK w kluczowym momencie, kiedy na rynku ogromnie wzrosły popularność i liczba dostępnych urządzeń z systemem Android. Obecnie stosowane są nie tylko smartfony, lecz także urządzenia z większymi ekranami, takie jak tablety i telewizory, także działające pod kontrolą Androida. Urządzenia te dysponują znacznie większą powierzchnią ekranu, którą mogą wykorzystać programiści. Na przykład typowy, dopracowany i elegancki interfejs użytkownika aplikacji, przygotowany z myślą o smartfonach, będzie na tabletach zazwyczaj wyglądał na zbyt uproszczony. Jednak wprowadzając do projektu interfejsu użytkownika komponenty fragmentów, można napisać jedną aplikację, która będzie dostosowana do wszystkich tych cech charakterystycznych i orientacji ekranu, a nie do wielu wersji aplikacji przystosowanych do konkretnych typów urządzeń. Takie rozwiązanie znacznie zwiększa możliwości wielokrotnego stosowania kodu, upraszcza proces testowania aplikacji oraz ułatwia publikowanie pakietu aplikacji oraz zarządzanie nim.

Zgodnie z tym, co podano we wstępie do tego rozdziału, jedną z podstawowych reguł, których trzymali się programiści aplikacji przeznaczonych na Androida, było tworzenie jednej aktywności obsługującej jeden, konkretny ekran aplikacji. Rozwiązanie to w bardzo ścisły i bezpośredni sposób

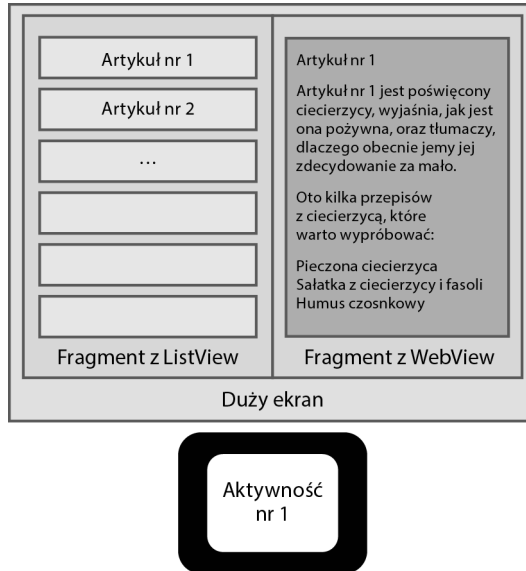
kojarzy zadania realizowane przez klasę aktywności z interfejsem użytkownika. Jednak okazało się, że w momencie, gdy na rynku pojawiły się urządzenia z większymi ekranami, takie rozwiązanie zaczęło przysparzać problemów. Kiedy na jednym ekranie pojawiło się więcej miejsca, umożliwiające wykonywanie większej liczby czynności, konieczne stało się implementowanie odrębnych klas aktywności, przeznaczonych do obsługi tych rozbudowanych ekranów. Niemniej jednak funkcje owych prostszych oraz rozbudowanych aktywności w przeważającej mierze były identyczne. Problem ten można rozwiązać dzięki zastosowaniu fragmentów, gdyż pozwalają one hermetyzować funkcje udostępniane w obszarach ekranu w formie komponentów, których można wielokrotnie używać i, w ramach tej samej aktywności, łączyć z innymi fragmentami.

Poniżej przeanalizowano teoretyczny przykład aplikacji. Jest to tradycyjna aplikacja na smartfony składająca się z dwóch ekranów, przeznaczona na przykład do prezentowania najnowszych informacji z witryny internetowej. Jej pierwszy ekran tworzy aktywność typu `ListActivity` zawierająca kontrolkę `ListView`. Każdy element tej listy `ListView` reprezentuje artykuł z witryny, który użytkownik może przeczytać. Ponieważ jest to aplikacja internetowa, zatem kliknięcie konkretnego artykułu powoduje przejście na drugi ekran, na którym treść wybranego artykułu jest prezentowana w kontrolce `WebView`. Ta tradycyjna architektura aplikacji internetowej została przedstawiona na rysunku 9.1.



Rysunek 9.1. Tradycyjne powiązania pomiędzy ekranami w aplikacji, która nie korzysta z fragmentów

Taka architektura interfejsu użytkownika aplikacji sprawdza się na niewielkich smartfonach, jednak na dużych tabletach lub telewizorach będzie przykładem marnowania dostępnego miejsca. W przypadku tych większych urządzeń należałoby raczej umieścić listę artykułów oraz treść wybranego artykułu na tym samym ekranie. Gdyby kontrolki `ListView` oraz `WebView` umieścić w dwóch niezależnych fragmentach, bez trudu można by zdefiniować układ, który prezentowałby je na tym samym ekranie wówczas, gdyby było na nim dostatecznie dużo miejsca. Teoretyczną postać takiej aplikacji przedstawiono na rysunku 9.2.



Rysunek 9.2. Tradycyjne powiązania pomiędzy ekranami w aplikacji, która nie korzysta z fragmentów

Opis cyklu życia obiektów Fragment

Cykl życia aktywności (obiektów `Activity`) został opisany w rozdziale 4. „Prezentacja komponentów aplikacji”. W tym punkcie rozdziału zostanie opisane, jak do schematu działania aktywności można dopasować fragmenty. Przede wszystkim fragment (obiekt klasy `Fragment`) musi być umieszczony w aktywności. Fragmenty mają swój własny cykl życia, niemniej jednak nie są one niezależnymi komponentami, które mogą istnieć poza kontekstem aktywności.

W przypadku gdy cały stan interfejsu użytkownika zostaje przeniesiony do poszczególnych fragmentów, odpowiedzialność klas `Activity` ulega znacznemu ograniczeniu i uproszczeniu. Klasy `Activity`, w których układach znajdują się same fragmenty, nie muszą już tracić czasu na zapisywanie oraz odtwarzanie stanu, gdyż obiekt aktywności zarządza wszelkimi dołączonymi do niego obiektami `Fragment`. Z kolei komponenty `Fragment` same zarządzają własnym stanem, wykorzystując do tego swój własny cykl życia. Oczywiście nic nie stoi na przeszkodzie, by w klasach aktywności dowolnie łączyć fragmenty z innymi kontrolkami `View`. W takich sytuacjach klasy `Activity` będą odpowiedzialne za zarządzanie używanymi kontrolkami `View`, jak działa się to do tej pory.

Aktywności muszą się także koncentrować na zarządzaniu używanymi w nich fragmentami. Koordynację działań między aktywnością a jej fragmentami upraszcza klasa `FragmentManager` (`android.app.FragmentManager`). Obiekt `FragmentManager` można pobrać przy użyciu metody `getFragmentManager()`, dostępnej w klasach `Activity` oraz `Fragment`. W przypadku korzystania z biblioteki wsparcia koordynację tę ułatwia klasa `FragmentManager` (`android.support.v4.app.FragmentManager`), której obiekt można pobrać przy użyciu metody `getSupportFragmentManager()`, dostępnej w API klasy `FragmentActivity`.

Definiowanie fragmentów

Implementacje klasy `Fragment`, które zostały zdefiniowane w aplikacji jako normalne klasy, można dodawać do plików układów przy użyciu znacznika XML `<fragment>`, a następnie wczytywać w aktywności, stosując standardową metodę `setContentView()`, która zazwyczaj jest wywoływana w metodzie `onCreate()` aktywności.

W przypadku odwoływania się do klas `Fragment` zdefiniowanych w pakiecie aplikacji w plikach XML układów należy korzystać ze znacznika `<fragment>`. Posiada on kilka ważnych atrybutów. Przede wszystkim w atrybucie `android:name` należy podać pełną nazwę klasy `Fragment`. Oprócz tego należy przypisać fragmentowi unikalny identyfikator, używając w tym celu atrybutu `android:id`; dzięki czemu, w razie konieczności, będzie można odwołać się do fragmentu w kodzie programu. Oprócz tego, podobnie jak we wszystkich innych kontrolkach umieszczanych w układach, trzeba także określić wymiary fragmentu za pomocą atrybutów `layout_width` oraz `layout_height`. Poniżej przedstawiona została prosta, przykładowa postać znacznika `<fragment>`, odwołującego się do klasy o nazwie `VeggieGardenListFragment`, zdefiniowanej jako klasa Javy należąca do pakietu:

```
<fragment
    android:name="com.introtoandroid.simplefragments.VeggieGardenListFragment"
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Zarządzanie modyfikacjami fragmentów

Jak można się przekonać, w sytuacjach gdy w ramach jednej aktywności jest umieszczonych na jednym ekranie kilka fragmentów, interakcje wykonywane przez użytkownika w jednym fragmencie (takim jak lista doniesień — `ListViewFragment` — przedstawionej wcześniej, przykładowej aplikacji) często będą powodować, że aktywność zmodyfikuje inny fragment (taki jak `WebViewFragment` przykładowej aplikacji). Aktualizacje lub modyfikacje fragmentów są wykonywane z wykorzystaniem obiektów `FragmentManager` (`android.app.FragmentManager` lub `android.support.v4.app.FragmentManager`). Korzystając z czynności udostępnianych przez klasę `FragmentManager`, można wykonywać na fragmentach kilka różnych operacji, takich jak:

- dołączanie fragmentu do nadrzędnej aktywności lub odłączanie od niej,
- ukrywanie lub wyświetlanie fragmentu w widoku.

W tym momencie można się zacząć zastanawiać, jak wygląda współpraca między przyciskiem *Wstecz* a takim właśnie interfejsem użytkownika korzystającym z fragmentów. Okazuje się, że obecnie klasa aktywności dysponuje własnym stosem cofnięć (ang. *back stack*). Programista może obecnie zdecydować, które operacje `FragmentManager` warto umieścić na tym stosie przy użyciu metody `addToBackStack()` klasy `FragmentManager`, a które można zignorować. Na przykład w przypadku przedstawionej wcześniej hipotetycznej aplikacji operacje wyświetlania artykułu we fragmencie `WebViewFragment` mogą być dodawane do stosu cofnięć, tak by klikanie przycisku *Wstecz* powodowało wyświetlanie poprzednio oglądanych artykułów, a dopiero potem wyjście z aplikacji.

Dołączanie fragmentów do aktywności i odłączanie ich od niej

Kiedy już zostanie utworzony fragment, który ma być używany w klasie aktywności, pojawi się problem zarządzania jego cyklem życia. Poniżej przedstawiono metody zwrotne służące do tego

celu; należy przy tym pamiętać, że fragmenty są tworzone wtedy, kiedy staną się potrzebne, a usuwane wtedy, gdy nie są już używane. Wiele spośród przedstawionych w tym miejscu metod odpowiada analogicznym metodom cyklu życia aktywności:

- Metoda zwrotna `onAttach()` jest wywoływana podczas pierwszego dołączania fragmentu do aktywności.
- Metoda zwrotna `onCreate()` jest wywoływana w momencie pierwszego tworzenia fragmentu.
- Metoda zwrotna `onCreateView()` jest wywoływana w chwili, gdy powinien być utworzony układ bądź hierarchia kontrolki `View` zawierająca dany fragment.
- Metoda zwrotna `onActivityCreated()` jest wywoływana w celu „poinformowania” fragmentu o zakończeniu wykonywania metody `onCreate()` nadrzędnej aktywności, do której został on dołączony.
- Metoda zwrotna `onStart()` jest wywoływana, kiedy interfejs użytkownika fragmentu został już utworzony, lecz jeszcze nie jest aktywny.
- Metoda zwrotna `onResume()` jest wywoływana, kiedy interfejs użytkownika fragmentu stanie się aktywny i gotowy do prowadzenia interakcji z użytkownikiem po wcześniejszym wznowieniu aktywności lub aktualizacji fragmentu przy użyciu obiektu `FragmentManager`.
- Metoda zwrotna `onPause()` jest wywoływana po wstrzymaniu wykonywania nadrzędnej aktywności lub po aktualizacji fragmentu z wykorzystaniem obiektu `FragmentManager`. Jej wywołanie oznacza, że fragment nie jest już dłużej aktywny bądź że jest obecnie wyświetlany w tle.
- Metoda zwrotna `onStop()` jest wywoływana po zatrzymaniu wykonywania nadrzędnej aktywności lub po aktualizacji fragmentu za pomocą obiektu `FragmentManager`. Jej wywołanie oznacza, że fragment nie jest już widoczny.
- Metoda zwrotna `onDestroyView()` jest wywoływana w celu wyczyszczenia zasobów skojarzonych z danym fragmentem, używanych przez układ lub hierarchię kontrolki `View`.
- Metoda zwrotna `onDestroy()` jest wywoływana w celu wyczyszczenia wszelkich pozostałych zasobów używanych przez fragment.
- Metoda zwrotna `onDetach()` jest wywoływana bezpośrednio przed odłączeniem fragmentu od aktywności.

Stosowanie specjalnych typów fragmentów

W rozdziale 8., zatytułowanym „Umiejscawianie elementów z użyciem układów”, przedstawiono kilka szczególnych wersji klasy `Activity`, przeznaczonych do zarządzania pewnymi często używanymi typami interfejsów użytkownika. Na przykład klasa `ListActivity` upraszcza tworzenie aktywności zarządzających kontrolką `ListView`. I analogicznie klasa `PreferenceActivity` ułatwia tworzenie aktywności zarządzających współdzielonymi preferencjami. A jak pokazał przedstawiony wcześniej przykład hipotetycznej aplikacji czytnika informacji, często będzie się pojawiać konieczność umieszczania kontrolki, takich jak `ListView` lub `WebView`, we fragmentach.

Ponieważ fragmenty zostały opracowane w celu odseparowania tej możliwości funkcjonalnej od klas aktywności, obecnie można znaleźć klasy pochodne klasy `Fragment`, odpowiadające wyspecjalizowanym klasom aktywności. Spośród nich warto dokładniej przyjrzeć się następującym klasom:

- **ListFragment** (`android.app.ListFragment`). Podobnie jak `ListActivity`, ta klasa `Fragment` ułatwia zarządzanie kontrolką `ListView`.
- **PreferenceFragment** (`android.preference.PreferenceFragment`). Podobnie jak `PreferenceActivity`, ta klasa `Fragment` ułatwia zarządzanie preferencjami użytkownika.
- **WebViewFragment** (`android.webkit.WebViewFragment`). Ten typ fragmentu zawiera kontrolkę `WebView`, dzięki czemu bez trudu można w nim wyświetlać treści z internetu. Aby aplikacja miała dostęp do internetu, wciąż będzie jej potrzebne uprawnienie `android.permission.INTERNET`.
- **DialogFragment** (`android.app.DialogFragment`). Odseparowanie możliwości funkcjonalnych związanych z interfejsem użytkownika od aktywności oznacza także, że obecnie nie jest pożądane także zarządzanie oknami dialogowymi z poziomu aktywności. Zamiast tego klasa ta pozwala na gromadzenie i zarządzanie kontrolkami w oknie dialogowym w sposób charakterystyczny dla fragmentów. Zagadnienia związane z oknami dialogowymi zostały opisane w rozdziale 10. „Określanie architektury z użyciem wzorców”.

Uwaga

Pełną listę wyspecjalizowanych rodzajów fragmentów można znaleźć w dokumentacji klasy `Fragment`, w sekcji prezentującej jej klasy pochodne, na stronie <http://d.android.com/reference/android/app/Fragment.html>.

Projektowanie aplikacji korzystających z fragmentów

W ostatecznym rozrachunku tworzenia aplikacji korzystających z fragmentów najlepiej jest się uczyć na przykładzie. Właśnie z tego powodu w dalszej części rozdziału zostanie przedstawiony stosunkowo prosty przykład, którego zadaniem będzie wyjaśnienie oraz praktyczne przedstawienie wielu opisanych tu pojęć. W celu uproszczenia rozwiązania przykład ten będzie przeznaczony dla konkretnej wersji platformy Android, a ściślej: Android Marshmallow. Niemniej jednak niebawem okaże się, że dzięki zastosowaniu bibliotek wsparcia aplikacje korzystające z fragmentów można tworzyć na niemal wszystkie urządzenia działające pod kontrolą Androida.

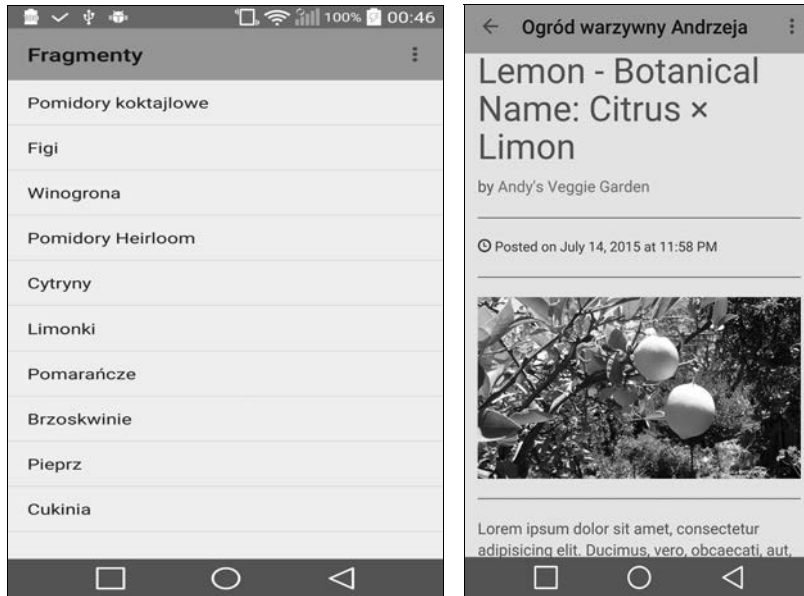
Wskazówka

Wiele fragmentów kodu przedstawionych w tym rozdziale pochodzi z aplikacji `SimpleFragments`. Jej kod źródłowy jest dostępny w przykładach dołączonych do książki, które można pobrać z serwera FTP wydawnictwa Helion pod adresem <ftp://ftp.helion.pl/przyklady/anwpp5.zip>.

Robi (fikcyjny robot) jest wielkim miłośnikiem prac ogrodowych i w swoim ogródku hoduje wiele owoców i warzyw. Przedstawiona tu przykładowa aplikacja będzie zawierać kontrolkę `ListView` prezentującą ich nazwy. Kliknięcie elementu tej listy będzie powodować wczytanie kontrolki `WebView` i wyświetlenie w niej strony poświęconej wybranej roślinie. Aby uprościć rozwiązanie, nazwy owoców i warzyw oraz adresy URL odpowiadających im stron WWW będą przechowywane w zasobach aplikacji, w tablicach łańcuchów znaków. (Pełna implementacja tej aplikacji jest dostępna w kodach dołączonych do książki).

A jak będą działać fragmenty używane w tej aplikacji? Lista zostanie zaimplementowana za pomocą fragmentu klasy `ListFragment`, natomiast za wyświetlanie stron będzie odpowiadał fragment typu `WebViewFragment`. W układzie pionowym na ekranie będzie wyświetlany tylko jeden fragment,

przez co konieczne będzie użycie dwóch klas aktywności, jak pokazano na rysunku 9.3. W przypadku tej przykładowej aplikacji aktywności zostaną zaimplementowane z wykorzystaniem klasy `AppCompatActivity` (`android.support.v7.AppCompatActivity`).



Rysunek 9.3. Sytuacja, w której każdy fragment jest umieszczony w osobnej aktywności i wyświetlany pojedynczo na ekranie

W układzie poziomym oba fragmenty będą wyświetlane na tym samym ekranie w ramach jednej aktywności `AppCompatActivity`. Tę sytuację przedstawia zrzut zamieszczony na rysunku 9.4.



Rysunek 9.4. W układzie poziomym oba fragmenty są widoczne na ekranie jednocześnie i działają w ramach jednej aktywności

Implementacja fragmentu ListFragment

W pierwszej kolejności przedstawiona zostanie implementacja klasy pochodnej klasy ListFragment, która w prezentowanej aplikacji będzie nosić nazwę VeggieGardenListFragment. Fragment ten będzie odpowiadał za wyświetlanie listy nazw warzyw i owoców. Wspomniana klasa musi określić, czy należy wczytać drugi fragment — VeggieGardenWebViewFragment — czy też kliknięcie elementu listy ma powodować uruchomienie drugiej aktywności (VeggieGardenViewActivity):

```
public class VeggieGardenListFragment extends ListFragment implements
FragmentManager.OnBackStackChangeListener {

    private static final String DEBUG_TAG = "VeggieGardenListFragment";
    int mCurPosition = 1;
    boolean mShowTwoFragments;

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
        String[] veggies = getResources().getStringArray(R.array.veggies_array);
        setListAdapter(new ArrayAdapter<>(getActivity(),
            android.R.layout.simple_list_item_activated_1, veggies));

        View detailsFrame = getActivity().findViewById(R.id.veggieentry);
        mShowTwoFragments = detailsFrame != null
            && detailsFrame.getVisibility() == View.VISIBLE;

        if (savedInstanceState != null) {
            mCurPosition = savedInstanceState.getInt("curChoice", 0);
        }

        if (mShowTwoFragments == true || mCurPosition != 1) {
            viewVeggieInfo(mCurPosition);
        }

        getFragmentManager().addOnBackStackChangeListener(this);
    }

    @Override
    public void onBackStackChanged() {
        VeggieGardenWebViewFragment details =
            (VeggieGardenWebViewFragment) getFragmentManager()
                .findFragmentById(R.id.veggieentry);
        if (details != null) {
            mCurPosition = details.getShownIndex();
            getListView().setItemChecked(mCurPosition, true);

            if (!mShowTwoFragments) {
                viewVeggieInfo(mCurPosition);
            }
        }
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putInt("curChoice", mCurPosition);
    }
}
```

```

@Override
public void onItemClick(ListView l, View v, int position, long id) {
    viewVeggieInfo(position);
}

void viewVeggieInfo(int index) {
    mCurrentPosition = index;

    if (mShowTwoFragments == true) {
        // Sprawdzamy, który fragment jest aktualnie wyświetlany, i w razie potrzeby go zastępujemy.
        VeggieGardenWebViewFragment details =
            (VeggieGardenWebViewFragment) getFragmentManager()
                .findFragmentById(R.id.veggieentry);
        if (details == null || details.getShownIndex() != index) {

            VeggieGardenWebViewFragment newDetails =
                VeggieGardenWebViewFragment.newInstance(index);

            FragmentManager fm = getFragmentManager();
            FragmentTransaction ft = fm.beginTransaction();
            ft.replace(R.id.veggieentry, newDetails);
            if (index != 1) {
                String[] veggies =
                    getResources().getStringArray(R.array.veggies_array);
                String strBackStackTagName = veggies[index];
                ft.addToBackStack(strBackStackTagName);
            }

            ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
            ft.commit();
        }
    } else {
        Intent intent = new Intent();
        intent.setClass(getActivity(), VeggieGardenViewActivity.class);
        intent.putExtra("index", index);
        startActivity(intent);
    }
}
}

```

Większość operacji związanych z inicjowaniem fragmentu jest wykonywana w metodzie zwrotnej `onActivityCreated()` — dzięki jej zastosowaniu zainicjowanie kontrolki `ListView` zostanie wykonane tylko jeden raz. Następnie fragment określa, w którym trybie wyświetlania ma działać — w tym celu sprawdza, czy w układzie jest dostępny drugi fragment. Pozostałe szczegóły związane z wyświetleniem fragmentu zostają przekazane do metody pomocniczej o nazwie `viewVeggieInfo()`, która będzie wywoływana także po każdym kliknięciu elementu listy `ListView`.

Logika działania metody `viewVeggieInfo()` uwzględnia oba tryby wyświetlania aplikacji. Jeśli urządzenie działa w układzie pionowym, to przy użyciu obiektu `Intent` uruchamiana jest aktywność `VeggieGardenViewActivity`. Z kolei w układzie poziomym konieczne jest wykonanie pewnych operacji na fragmentach.

Konkretnie rzecz biorąc, za pomocą obiektu klasy `FragmentManager` i na podstawie unikalnego identyfikatora (a konkretnie `R.id.veggieentry` zdefiniowanego w pliku układu) odnajdywany jest fragment `VeggieGardenWebViewFragment`. Następnie tworzona jest nowa instancja fragmentu `VeggieGardenWebViewFragment`, w której zostanie wyświetlona strona WWW nowego, wybranego

z listy owocu lub warzywa. Kolejną operacją jest utworzenie transakcji (obiektu `FragmentTransaction`), w ramach której poprzedni fragment `VeggieGardenWebViewFragment` zostaje zastąpiony nowym. Poprzedni fragment zostaje także umieszczony na stosie cofnięć, tak by naciśnięcie przycisku *Wstecz* działało prawidłowo. Oprócz tego zostanie określona animacja przejścia pomiędzy fragmentami, po czym transakcja zostanie zatwierdzona, co spowoduje asynchroniczną aktualizację ekranu.

Dodatkowo stos cofnięć jest monitorowany przez wywołanie metody `addOnBackStackChangeListener()`. Metoda zwrotna `onBackStackChanged()` aktualizuje listę, dodając do niej obecnie wybrany element. Rozwiązanie to stanowi solidny sposób synchronizacji elementów wybieranych z listy `List<View>` z zawartością wyświetlaną we fragmencie, i to zarówno w przypadku dodawania nowego fragmentu do stosu cofnięć, jak i podczas usuwania fragmentów z tego stosu, na przykład gdy użytkownik naciśnie przycisk *Wstecz*.

Implementacja fragmentu `WebViewFragment`

Kolejnym etapem tworzenia aplikacji jest zaimplementowanie klasy pochodnej `WebViewFragment` o nazwie `VeggieGardenWebViewFragment`, reprezentującej fragment, w którym będzie wyświetlana strona poświęcona wybranemu owocowi lub warzywu. Działanie tej klasy ogranicza się niemal jedynie do określenia adresu URL strony i wyświetlenia jej w kontrolce `WebView`:

```
public class VeggieGardenWebViewFragment extends WebViewFragment {

    private static final String DEBUG_TAG = "VGWebViewFragment";

    public static VeggieGardenWebViewFragment newInstance(int index) {
        Log.v(DEBUG_TAG, "Tworzenie nowej instancji: " + index);
        VeggieGardenWebViewFragment fragment = new VeggieGardenWebViewFragment();
        Bundle args = new Bundle();
        args.putInt("index", index);
        fragment.setArguments(args);
        return fragment;
    }

    public int getShownIndex() {
        int index = 1;
        Bundle args = getArguments();
        if (args != null) {
            index = args.getInt("index", -1);
        }
        if (index == -1) {
            Log.e(DEBUG_TAG, "Brak takiego indeksu w tablicy.");
        }

        return index;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        String[] veggieUrls = getResources().getStringArray(R.array.veggieurls_array);
        int veggieUrlIndex = getShownIndex();

        WebView webview = getWebView();
        webview.setPadding(0, 0, 0, 0);
    }
}
```

```

webView.getSettings().setLoadWithOverviewMode(true);
webView.getSettings().setUseWideViewPort(true);

if (veggieUrlIndex != 1) {
    String veggieUrl = veggieUrls[veggieUrlIndex];
    webView.loadUrl(veggieUrl);
} else {
    String veggieUrl = "http://andys-veggie-garden." +
        "appspot.com/cherrytomatoes";
    webView.loadUrl(veggieUrl);
}
}
}

```

Także w tym fragmencie przeważająca większość czynności związanych z jego inicjowaniem jest wykonywana w metodzie zwrotnej `onActivityCreated()`, dzięki czemu będzie je trzeba wykonać tylko jeden raz. Domyślna konfiguracja kontrolki `WebView` nie zapewnia dostatecznie dobrych efektów, dlatego metoda wprowadza w niej pewne zmiany; konkretnie rzecz biorąc, usuwa wypełnienia wokół kontrolki i zapewnia, że obszar przeglądarki będzie idealnie pasować do obszaru dostępnego na ekranie. Jeśli zostało przekazane żądanie wyświetlenia strony konkretnego owocu lub warzywa, to odnajdywany jest jej adres URL, po czym strona zostaje wczytana; w przeciwnym razie wyświetlana jest „domyślna” strona WWW aplikacji, poświęcona pomidorom koktajlowym.

Definiowanie plików układów

Po zaimplementowaniu klas obu fragmentów można dodać te fragmenty do odpowiednich plików zasobów z definicjami układów. Konieczne będzie utworzenie dwóch takich plików. Jeśli aplikacja będzie działać w układzie poziomym, stosowany będzie jeden plik układu, *activity_simple_fragments.xml*, w którym zostaną umieszczone oba fragmenty. Z kolei w przypadku gdy aplikacja będzie działać w układzie pionowym, zastosowany zostanie podobny plik układu zawierający wyłącznie fragment `ListFragment`. Interfejs użytkownika zawierający fragment `WebViewFragment` będzie generowany programowo w trakcie działania aplikacji.

W pierwszej kolejności przedstawiony zostanie plik układu używanego w układzie poziomym — *res/layout-land/activity_simple_fragments.xml*. Warto zwrócić uwagę, że plik ten jest umieszczony w specjalnym katalogu przeznaczonym dla układów używanych wyłącznie w układzie poziomym. Więcej informacji dotyczących przechowywania alternatywnych zasobów można znaleźć w rozdziale 13. „Projektowanie zgodnych aplikacji”. Jak na razie wystarczy powiedzieć, że układ ten będzie wczytywany automatycznie, zawsze gdy urządzenie będzie działać w układzie poziomym.

```

<?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">

    <include
        android:id="@+id/toolbar"
        layout="@layout/tool_bar" />

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:baselineAligned="false">

```

```

<fragment
    android:name="com.introtoandroid.simplefragments.VeggieGardenListFragment"
    android:id="@+id/list"
    android:layout_weight="1"
    android:layout_width="200dp"
    android:layout_height="match_parent" />
<FrameLayout
    android:id="@+id/veggieentry"
    android:layout_weight="4"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
</LinearLayout>
</LinearLayout>

```

W tej sytuacji mamy do czynienia ze stosunkowo prostym układem typu `LinearLayout`, wewnątrz którego jest umieszczony drugi układ `LinearLayout` zawierający dwie kontrolki podrzędne. Pierwszą z nich jest statyczny komponent `Fragment`, odwołujący się do zaimplementowanego wcześniej fragmentu `ListFragment`. Jeśli natomiast chodzi o drugą kontrolkę, reprezentującą obszar, w którym będą przedstawiane fragmenty `WebViewFragment`, to zostanie ona zdefiniowana jako układ `FrameLayout`. W trakcie działania aplikacji układ ten będzie programowo zastępowany obiektami `VeggieGardenWebViewFragment`.

Zasoby umieszczone w zwyczajnym katalogu układów będą używane w sytuacjach, gdy urządzenie nie będzie pracować w układzie pionowym (czyli, innymi słowy, gdy będzie trzymane w układzie pionowym). W takim przypadku konieczne jest zdefiniowanie dwóch układów. W pierwszej kolejności zostanie przedstawiony statyczny układ z fragmentem `ListFragment`, zapisany w pliku `res/layout/activity_simple_fragments.xml`. Jest on bardzo podobny do poprzedniego układu, lecz nie występuje w nim kontrolka `FrameLayout`:

```

<?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"
    tools:context=".SimpleFragmentActivity">
    <include
        android:id="@+id/toolbar"
        layout="@layout/tool_bar" />
    <fragment
        android:name="com.introtoandroid.simplefragments.VeggieGardenListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        tools:layout="@layout/activity_simple_fragments" />
</LinearLayout>

```

Definiowanie klas aktywności

To już prawie wszystko. Kolejnym krokiem jest napisanie klas aktywności, w których zostaną umieszczone przygotowane wcześniej fragmenty. Potrzebne będą dwie takie klasy: główna oraz dodatkowa, która będzie używana wyłącznie w celu wyświetlenia fragmentu `VeggieGardenWebViewFragment`, w sytuacji gdy urządzenie będzie działać w układzie pionowym. Główna klasa aktywności będzie nosić nazwę `SimpleFragmentsActivity`, a druga — `VeggieGardenActivity`.

Zgodnie z zamieszczonymi wcześniej informacjami przeniesienie całej logiki obsługi interfejsu użytkownika do klas fragmentów ogromnie upraszcza implementację klas aktywności. Poniżej przedstawiony został kompletny kod klasy SimpleFragmentsActivity:

```
public class SimpleFragmentsActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_simple_fragments);
        Toolbar toolbar;
        Toolbar = (Toolbar) findViewById(R.id.toolbar);
        getSupportActionBar(toolbar);
    }
}
```

No właśnie! To cały niezbędny kod! Klasa VeggieGardenViewActivity jest tylko nieco bardziej interesująca:

```
public class VeggieGardenViewActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (getResources().getConfiguration().orientation ==
            Configuration.ORIENTATION_LANDSCAPE) {
            finish();
            return;
        }
        if (savedInstanceState == null) {
            setContentView(R.layout.activity_simple_fragments);
            Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
            getSupportActionBar(toolbar);
            getSupportActionBar().setDisplayHomeAsUpEnabled(true);
            VeggieGardenWebViewFragment details = new VeggieGardenWebViewFragment();
            details.setArguments(getIntent().getExtras());

            FragmentManager fm = getFragmentManager();
            FragmentTransaction ft = fm.beginTransaction();
            ft.replace(R.id.list, details);
            ft.commit();
        }

        @Override
        public boolean onOptionsItemSelected(MenuItem item) {
            if (item.getItemId() == android.R.id.home) {
                onBackPressed();
                return true;
            }
            return super.onOptionsItemSelected(item);
        }
    }
}
```

Powyższa aktywność w pierwszej kolejności sprawdza, czy urządzenie znajduje się w odpowiednim ułożeniu, i na tej podstawie określa, czy w ogóle można wykonywać dalsze operacje. Potem tworzona jest instancja fragmentu VeggieGardenWebViewFragment, która następnie zostaje użyta do zastąpienia widoku R.id.list, będącego głównym widokiem domyślnego układu każdej aktywności typu ListActivity. To wszystkie czynności, które trzeba wykonać w celu zapewnienia prawidłowego

działania prezentowanej tu prostej aplikacji korzystającej z fragmentów. Dodatkowo warto zwrócić uwagę, że używa ona paska narzędzi, komponentu `ToolBar`, jako paska akcji.

Stosowanie pakietu biblioteki wsparcia

Fragmenty są tak ważne dla przyszłości platformy Android, że twórcy systemu zdecydowali się na przygotowanie specjalnej biblioteki, dzięki której programiści mogą, jeśli tylko zechcą, zaktualizować swoje stare aplikacje, przeznaczone nawet dla systemu Android 1.6, i wykorzystać w nich fragmenty. Biblioteka ta była początkowo określana jako *Compatibility Package* (pakiet zgodności), jednak obecnie określa się ją jako pakiet *Android Support Library* (pakiet biblioteki wsparcia Androida).

Dodawanie wsparcia dla fragmentów do starych aplikacji

Decyzja o tym, czy warto aktualizować stare aplikacje, czy nie, należy jedynie do ich twórców. Aplikacje, w których fragmenty nie są używane, powinny bez żadnych błędów działać przez całą przewidywalną przyszłość, a to głównie ze względu na prowadzoną przez twórców Androida politykę maksymalnie długiego wspierania starszych aplikacji w kolejnych wydawanych wersjach systemu. Poniżej przedstawiono kilka uwag, które twórcy aplikacji powinni przemyśleć, rozważając, czy warto aktualizować ich istniejący kod:

- Stare aplikacje można pozostawić w niezmienionej postaci, gdyż konsekwencje, z jakimi będzie się to wiązać, nie będą katastrofalne. Takie aplikacje nie będą co prawda wykorzystywać najnowszych i najlepszych możliwości oferowanych przez system Android (co użytkownicy na pewno zauważą), niemniej jednak będą cały czas działać tak jak wcześniej, bez żadnego dodatkowego nakładu pracy ze strony ich twórców. Jeśli twórcy nie mają zamiaru aktualizować ani rozwijać aplikacji, to takie rozwiązanie będzie całkowicie uzasadnione. Pewnym problemem może być mało optymalne wykorzystanie dostępnego obszaru ekranu, choć nie powinno to spowodować nowych błędów.
- Jeśli aplikacja jest bardzo popularna na rynku, a jej twórcy aktualizowali ją wraz z rozwojem platformy Android, to będą też zapewne bardziej skłonni do rozważenia zastosowania pakietu `Support Library`. Mogą tego także wymagać użytkownicy aplikacji. Oczywiście można kontynuować wsparcie dla starej aplikacji i jednocześnie napisać jej nową, poprawioną wersję korzystającą z możliwości nowszych wersji platformy. Jednak oznacza to konieczność zorganizowania dwóch wersji kodu i dwóch pakietów aplikacji oraz zarządzania nimi, a także komplikuje publikowanie aplikacji i zgłaszanie błędów, nie wspominając nawet o dodatkowych problemach związanych z utrzymaniem aplikacji oraz z ich reklamowaniem. Znacznie lepszym rozwiązaniem będzie unowocześnienie już istniejącej aplikacji i wykorzystanie w niej pakietu `Support Library` oraz dołożenie wszelkich starań w celu zachowania możliwości utrzymania jednej bazy kodu aplikacji. Ważnymi czynnikami przy podejmowaniu takiej decyzji będą na pewno wielkość i zasoby organizacji.
- Samo rozpoczęcie korzystania z pakietu `Support Library` w aplikacji nie oznacza wcale, że od razu trzeba w niej wykorzystywać wszystkie nowe możliwości, takie jak fragmenty, mechanizmy wczytywania (ang. *loaders*) czy paski narzędzi. Można wybrać tylko te spośród

możliwości, które mają sens w przypadku danej aplikacji, a pozostałe dodawać stopniowo wraz z jej rozwojem, o ile zespół twórców będzie miał niezbędne zasoby i chęci.

- Rezygnacja z aktualizowania aplikacji i wprowadzania w niej nowych kontrolek może sprawić, że w porównaniu z innymi aplikacjami będzie ona wyglądać staro. Jeśli tworzona aplikacja jest w bardzo dużym stopniu dostosowana do konkretnych potrzeb i wymagań i nie korzysta z żadnych domyślnych kontrolek — co często dzieje się w grach lub innych aplikacjach o wyszukanej formie graficznej — może się okazać, że wcale nie trzeba jej aktualizować. Jeśli jednak aplikacja używa standardowych kontrolek i standardowego sposobu prezentacji i obsługi, to nadanie jej nowoczesnego, świeżego wyglądu może mieć bardzo duże znaczenie.

Stosowanie fragmentów w nowych aplikacjach przeznaczonych dla starszych wersji platformy

W przypadku rozpoczynania pracy nad nową aplikacją, która jednak ma działać także w starszych wersjach platformy Android, podjęcie decyzji o zastosowaniu fragmentów może być znacznie prostsze. W takiej sytuacji po prostu trudno znaleźć argumenty przemawiające za niewykorzystaniem nowych możliwości, natomiast bez kłopotu można znaleźć powody, by je wykorzystywać:

- Niezależnie od tego, z myślą o których konkretnie platformach i urządzeniach jest tworzona aplikacja, w przyszłości i tak pojawią się kolejne, które obecnie trudno jeszcze przewidzieć. Fragmenty zapewniają elastyczność pozwalającą na łatwe dostosowywanie interfejsu użytkownika do zmieniających się sposobów pracy, bez konieczności ponownego pisania i testowania całego kodu aplikacji.
- Wczesne zaadaptowanie pakietu Support Library w aplikacji oznacza, że jeśli w przyszłości zostaną dodane nowe możliwości, wystarczy wówczas zaktualizować biblioteki i będzie można z tych możliwości skorzystać.
- Dzięki wykorzystaniu pakietu Support Library aplikacja będzie znacznie wolniej wykazywać objawy starzenia się, gdyż będzie korzystała z nowszych możliwości platformy i udostępnić je także użytkownikom używającym starszych wersji systemu Android.

Dołączanie pakietu Support Library do aplikacji

Pakiet Android Support Library jest po prostu zestawem statycznych bibliotek wsparcia (udostępnianym w formie pliku *.jar*), który wystarczy dołączyć do aplikacji, by móc go używać. Pakiet ten można pobrać za pomocą programu Android SDK Manager, a następnie dodawać do wybranych projektów. Jest to pakiet opcjonalny i domyślnie nie jest dołączony do żadnych projektów. Pakiety Android Support Library są udostępniane w wersjach, podobnie jak wszystkie inne elementy Android SDK, i od czasu do czasu są wzbogacane o nowe możliwości oraz, co ważniejsze, uzupełniane o poprawki wykrytych błędów.

Wskazówka

Więcej informacji na temat najnowszej wersji tego pakietu można znaleźć w witrynie *Android Developer*, a konkretnie na stronie <http://d.android.com/tools/support-library/index.html>.

Dostępnych jest siedem wersji pakietu Support Library: v4, v7, v8, v13, v17, Annotation oraz Design. Wersja v4 pakietu obejmuje nowe klasy wprowadzone w systemie Honeycomb oraz następnym i udostępnia je w wersjach platformy Android zgodnych z API poziomu 4 (Android 1.6). To właśnie tego pakietu należy używać, aby zapewnić możliwość stosowania fragmentów w starych aplikacjach. Wersja v7 pakietu obejmuje dodatkowe klasy nieobecne w wersji v4 i jest przeznaczona do udostępniania nowych możliwości w wersjach platformy zgodnych z API poziomu 7 (Android 2.1). Jest ona podzielona na następujące grupy: `appcompat`, `cardview`, `gridlayout`, `mediarouter`, `palette` oraz `recyclerview`. Wersja v8 udostępnia pakiet `renderscript` wspomagający obliczenia `RenderScript` w wersjach platformy zgodnych z API poziomu 8 (Android 2.2). W wersji v13 znalazły się bardziej wydajne implementacje niektórych klas, takich jak `FragmentCompat`, przeznaczone dla wersji platformy zgodnych z API poziomu 13 i nowszych. Wersja v17 zawiera widżety przeznaczone do tworzenia interfejsów użytkownika aplikacji służących do uruchamiania na telewizorach, takie jak `BrowseFragment`, `DetailsFragment`, `PlaybackOverlayFragment` czy też `SearchFragment`. Pakiet `Annotation` zapewnia możliwość dodawania do kodu adnotacji z metadanymi, natomiast pakiet `Design` — stosowanie wzorców projektowych oraz elementów użytkownika typowych dla Material Design.

Aby dodać do aplikacji pakiet Android Support Library, należy wykonać następujące czynności:

1. W przypadku korzystania z Android Studio trzeba pobrać Android Support Repository, używając programu Android SDK Manager. Element Android Support Library jest przeznaczony do użycia w Eclipse.
2. Następnie należy znaleźć plik modułu `build.gradle` (nie plik projektu `build.gradle`) projektu wyświetlony w panelu *Project* Android Studio i otworzyć go.
3. W sekcji zależności (`dependencies`) pliku trzeba dodać wybrane elementy pakietu Support Library, które będą używane w projekcie, podając ich prawidłowe identyfikatory oraz numery wersji. W przypadku prezentowanej tu aplikacji `SimpleFragments` dodane zostały następujące pakiety: `support-v4`, `appcompat-v7` oraz `design`, przy czym każdy z nich, jak pokazano na poniższym przykładzie, w wersji 23.0.0:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile "com.android.support:support-v4:23.0.0"
    compile "com.android.support:appcompat-v7:23.0.0"
    compile 'com.android.support:design:23.0.0'
}
```

4. Należy rozpocząć korzystanie z nowych API dodanych do projektu. Na przykład w celu napisania klasy dziedziczącej po `FragmentActivity` należy zaimportować klasę `android.support.v4.app.FragmentActivity`.

Uwaga

Istnieje kilka różnic między API udostępnianymi przez pakiet Android Support Library a tymi, które można znaleźć w nowszych wersjach Android SDK. Niemniej jednak w pakiecie tym jest kilka klas, których nazwy zmieniono w celu uniknięcia konfliktów, a poza tym nie wszystkie klasy oraz ich możliwości zostały zaimplementowane prawidłowo.

Dodatkowe sposoby stosowania fragmentów

Fragmenty idealnie nadają się do tworzenia komponentów interfejsu użytkownika przeznaczonych do wielokrotnego stosowania; istnieją jednak także inne możliwości wykorzystania fragmentów w aplikacjach. Na przykład można tworzyć komponenty funkcjonalne pozbawione interfejsu użytkownika bądź też zagnieżdżać jedne fragmenty wewnątrz innych.

Fragmenty funkcjonalne pozbawione interfejsu użytkownika

Fragmenty nie służą wyłącznie do zapewniania separacji interfejsu użytkownika od kodu aktywności. Można ich także używać do implementacji określonych możliwości funkcjonalnych czy zachowań aplikacji, w formie odrębnych komponentów nadających się do wielokrotnego użycia. W takiej sytuacji, dodając lub zastępując fragment, należy zamiast identyfikatora zasobu podać unikalny łańcuch znaków pełniący rolę znacznika. Ponieważ nie wiąże się to z dodawaniem kolejnego widoku do układu, nigdy nie jest wywoływana metoda `onCreateView()`. Trzeba także pamiętać, by takie fragmenty pobierać z aktywności, stosując metodę `findFragmentByTag()`.

Poznanie zagnieżdżonych fragmentów

W systemie Android 4.2 (API poziomu 17) wprowadzona została możliwość zagnieżdżania fragmentów w innych fragmentach. Analogiczna możliwość została także dodana do pakietu Android Support Library, dzięki czemu z zagnieżdżonych fragmentów można korzystać nawet w systemie Android 1.6 (API poziomu 4). Aby umieścić jeden fragment wewnątrz innego, należy wywołać metodę `getChildFragmentManager()` klasy `Fragment`, która zwraca obiekt typu `FragmentManager`. Po pobraniu tego obiektu można rozpocząć transakcję fragmentu (`FragmentTransaction`), wywołując w tym celu metodę `beginTransaction()`, po czym wywołać metodę `add()`, przekazując do niej fragment oraz jego układ, a na koniec zatwierdzić transakcję z wykorzystaniem metody `commit()`. W zagnieżdżonym fragmencie można nawet wywołać metodę `getParentFragment()`, by pobrać fragment zewnętrzny i wykonać na nim jakieś operacje.

Wprowadzenie fragmentów zagnieżdżonych otwiera nowe możliwości tworzenia dynamicznych komponentów, doskonale nadających się do wielokrotnego stosowania. Jako potencjalne przykłady wykorzystania można tu podać fragmenty zawierające karty umieszczane w innych fragmentach z kartami, przechodzenie z jednego fragmentu ze szczegółami do innego z użyciem kontrolki `ViewPager`, stosowanie kontrolki `ViewPager` do przeglądania fragmentów w ramach fragmentu z kartami, zagnieżdżanie fragmentów funkcjonalnych wewnątrz fragmentów posiadających własny interfejs użytkownika i tak dalej.

Podsumowanie

Fragmenty zostały wprowadzone w Android SDK, aby ułatwić obsługę różnych wielkości ekranów, które muszą być obsługiwane teraz i będą musiały być obsługiwane w przyszłości. `Fragment`, obiekt typu `Fragment`, jest po prostu samodzielnym, niezależnym elementem interfejsu użytkownika, bądź też implementacją pewnego zachowania, który to element posiada własny cykl życia i może być niezależny od jakiegóż jednej, konkretnej aktywności. Fragmenty muszą być umieszczane

w klasach aktywności, przy czym zapewniają programistom znaczną elastyczność pod względem możliwości podziału ekranów aplikacji na komponenty, które można łączyć ze sobą na różne sposoby w zależności od dostępnej wielkości ekranu. Fragmenty zostały wprowadzone w systemie Android 3.0, jednak dzięki pakietowi Android Support Library można ich używać także w starszych aplikacjach, nawet tych przeznaczonych na platformę zgodną z API poziomu 4 (Android 1.6). Jeszcze większą elastyczność podziału interfejsu użytkownika na komponenty zapewniają API pozwalające na tworzenie fragmentów zagnieżdżonych.

Pytania kwizowe

1. Która klasa ułatwia koordynację działań pomiędzy aktywnością a umieszczonymi w niej fragmentami?
2. Którą metodę należy wywołać w celu pobrania obiektu klasy ułatwiającej koordynację działań pomiędzy aktywnością a umieszczonymi w niej fragmentami?
3. Co powinno być wartością atrybutu `android:name` znacznika XML `<fragment>`?
4. Prawda czy fałsz? Metoda zwrotna `onActivityAttach()` jest wywoływana, gdy fragment zostanie po raz pierwszy dołączony do aktywności.
5. Podaj nazwy kilku klas pochodnych klasy `Fragment` (`android.app.Fragment`).
6. Jaką kontrolkę zawiera w sobie fragment typu `ListFragment` (`android.app.ListFragment`)?
7. Fragmenty zostały wprowadzone w API poziomu 11 (Android 3.0). W jaki sposób należałoby dodać możliwość stosowania fragmentów w aplikacji, która ma działać na urządzeniach z systemem Android zgodnym z API poziomu 11?

Ćwiczenia

1. Korzystając z dokumentacji Androida, dowiedz się, jak można dodawać fragmenty do stosu cofnięć. Napisz prostą aplikację korzystającą z układu składającego się z jednego fragmentu służącego do podawania liczby (fragment wyświetlany po uruchomieniu aplikacji powinien wyświetlać liczbę 1) oraz przycisku umieszczonego poniżej. Kliknięcie przycisku powinno spowodować zastąpienie pierwszego fragmentu drugim, w którym będzie wyświetlona liczba 2. Aplikacja ma pozwalać na takie zastępowanie fragmentów aż do wyświetlenia liczby 10, przy czym każdy z fragmentów ma być dodawany do stosu cofnięć, by zapewnić użytkownikowi możliwość powrotu do wcześniej wyświetlanej wartości.
2. Utwórz nowy projekt typu *Phone and Tablet*, używając Android Studio i wykorzystując kreator nowego projektu — na stronie *Add an activity to Mobile* zaznacz opcję *Master/Detail Flow*, następnie wykonaj kolejne kroki kreatora, a na końcu kliknij przycisk *Finish*. Uruchom wygenerowaną aplikację zarówno na telefonie, jak i urządzeniu z ekranem o wielkości typowej dla tabletów, by sprawdzić, jak będzie wyglądać w obu przypadkach; następnie przeanalizuj jej kod, by przekonać się, jak są w niej używane fragmenty.

3. Utwórz układ składający się z dwóch fragmentów, z których oba są generowane i wstawiane do niego programowo w trakcie działania aplikacji. Każdy z fragmentów powinien zajmować 50% dostępnego obszaru ekranu i mieć inny kolor.

Odwołania i inne źródła informacji

Szkolenie: *Building a Dynamic UI with Fragments*¹:

<http://d.android.com/training/basics/fragments/index.html>

Przewodnik dla programistów: *Fragments*²:

<http://d.android.com/guide/components/fragments.html>

Dokumentacja Android SDK dotycząca klasy `Fragment`:

<http://d.android.com/reference/android/app/Fragment.html>

Dokumentacja Android SDK dotycząca klasy `ListFragment`:

<http://d.android.com/reference/android/app/ListFragment.html>

Dokumentacja Android SDK dotycząca klasy `PreferenceFragment`:

<http://d.android.com/reference/android/preference/PreferenceFragment.html>

Dokumentacja Android SDK dotycząca klasy `WebViewFragment`:

<http://d.android.com/reference/android/webkit/WebViewFragment.html>

Dokumentacja Android SDK dotycząca klasy `DialogFragment`:

<http://d.android.com/reference/android/app/DialogFragment.html>

Narzędzia platformy Android: *Support Library*:

<http://d.android.com/tools/support-library/index.html>

Blog twórców systemu Android: *The Android 3.0 Fragments API*:

<http://android-developers.blogspot.com/2011/02/android-30-fragments-api.html>

¹ Tworzenie dynamicznych interfejsów użytkownika z użyciem fragmentów — *przyj. tłum.*

² Fragmenty — *przyj. tłum.*

Skorowidz

A

- ADB, Android Debug Bridge, 621
- akcje zależne, 282
- aktualizacja, 476
 - Android SDK, 67
 - Android Studio, 66
 - kart, 405
 - klasy `SampleMaterialActivity`, 400
 - preferencji, 369
 - rekordów, 421
- aktywności, 109, 112
 - cykl życia, 113
- aktywność
 - `AppCompatActivity`, 249
 - edycji karty, 331
 - `FirstChildActivity`, 270
 - `GreenBrandActivity`, 304, 306
 - `SampleMaterialActivity`, 326
 - `TransitionAddActivity`, 326
- analiza wyników testów, 513
- Android Runtime, 50
- Android SDK, 62, 67–69
 - aktualizacja, 66
 - dokumentacja, 70, 71
 - licencja, 69
 - narzędzia, 73
 - pakiety, 71
 - problemy, 68
- Android SDK Manager, 63, 73–76
- Android Studio, 66–68, 73
- Android Studio Setup, 63
- animacja, 164
 - kołowa, 321
 - przewijania, 320
- animacje przejść, 165, 166
- AOSP, Android Open Source Project, 39
- AOT, ahead-of-time compilation, 50
- API, 344, 517
- aplikacja
 - `Borderless Buttons`, 83
 - `BorderlessButtons`, 82, 84, 86
 - `PasswordMatcher`, 504
 - `SampleMaterial`, 312, 397
- aplikacje
 - aktualizacja, 477
 - aktualizacje, 476
 - bezpieczne, 472
 - debugowanie, 527
 - diagnostyka, 476
 - dokumentacja, 451
 - dołączanie zasobów, 531
 - dystrybucja, 531
 - Google Play, 532, 539
 - kanały dystrybucji, 543
 - kontrola wersji testowych, 449
 - korzystające z sieci, 445
 - maksymalizacja zysków, 473
 - na samochody, 360
 - na tablety, 357
 - na telewizory, 359
 - na zegarki, 360
 - najlepsze praktyki, 479
 - najlepsze praktyki projektowania, 467
 - niezależne, 445

- aplikacje
 - pielęgnacja, 446, 450
 - poprawa użyteczności, 463
 - prototypy, 464
 - przenoszenie, 452
 - przygotowanie serwerów i usług, 531
 - publikacja, 525
 - publikowanie, 521
 - rejestracja, 527
 - rodzime, 54
 - rozszerzanie, 446, 477
 - samodzielne publikowanie, 544
 - selekcja możliwości, 452
 - statystyki, 524
 - szybko reagujące, 470
 - testowanie, 448, 480, 487
 - tłumaczenie, 542
 - tworzenie, 448
 - tworzenie atrapy, 464
 - tworzenie pakietu, 527
 - układ ekranów, 462
 - wdrażanie, 450
 - wdrażanie etapami, 541
 - wersja finalna, 465
 - weryfikacja uprawnień, 527
 - wsparcie, 450
 - współdziałanie, 447
 - wyrażanie tożsamości, 460
 - zachowanie jakości, 474
 - Application Sandbox, 50
 - architektura
 - Androida, 50
 - nawigacji, 265
 - platformy, 51
 - ART, Android Runtime, 50
 - asercje, assertions, 512
 - atrapa aplikacji, 464
 - atrybut
 - colorBackground, 303
 - quantity, 155
 - textColorSecondary, 303
 - atrybuty
 - klasy Dialog, 287
 - kontrolki ViewGroup, 218
 - układów LinearLayout, 219, 220
 - układów RelativeLayout, 221, 222, 223
 - układu FrameLayout, 224
 - układu GridLayout, 229
 - układu TableLayout, 227
 - układu TableRow, 227
 - automatyczna kopia zapasowa, 380
 - automatyczne uzupełnianie, 187, 558
 - automatyzacja testów, 493, 517
 - AVD, Android Virtual Device, 84, 567
 - AVD Manager, 73
- ## B
- badanie katalogów, 386
 - baza danych
 - SQLite, 397
 - urządzeń, 433
 - bazy danych
 - pobieranie kart, 403
 - tworzenie kart, 402
 - usuwanie karty, 406
 - bezpieczeństwo, 52
 - bezpieczne aplikacje, 472
 - biblioteka wsparcia, 256, 344, 638
 - błędy
 - budowy, 563
 - projektowania, 478
 - testowania, 502
- ## C
- cele
 - twórców aplikacji, 456
 - użytkowników, 456
 - CRM, 121
 - CVS, 444
 - Cyanogen OS, 57
 - CyanogenMod, 57
 - cykl życia
 - aktywności, 113, 114
 - obiektów, 286
 - obiektów Fragment, 245
 - czcionka casual, 307

D

dane

- aplikacji, 383
- preferencji, 371
- użytkowników, 472

debugowanie, 65, 100, 527

- aplikacji, 96, 101, 620
- urządzenia, 64

defekty, 488

definiowanie

- fragmentów, 246
- klas aktywności, 254
- klasy DialogFragment, 287
- kształtu, 297
- list stanów kolorów, 164
- manifestu, 127
- plików układów, 253
- przycisku, 164
- standardów kodowania, 481
- układów, 211
- uprawnień aplikacji, 52
- zasobów menu, 166
- zasobów z kolorami, 157
- zasobów z liczbami całkowitymi, 157
- zasobów z wartościami logicznymi, 156
- zasobów z wymiarami, 158
- zasobu XML, 168

Device Monitor, 595

diagnostyka

- aplikacji, 476
- kodu, 482

dodawanie

- bazy SQLite, 397
- karty, 404
- mechanizmów rejestracji, 100
- paska postępu, 201
- preferencji, 369
- rekordów, 419
- testów, 514
- uprawnień, 416
- wsparcia dla fragmentów, 256
- zależności od bibliotek, 647

dokumentacja, 615

- Android SDK, 70, 71

aplikacji, 451

- Javadoc, 563
- projektowa, 442

dołączanie

- fragmentów do aktywności, 246
- pakietu Support Library, 257
- wymagań, 433

domyślny motyw

- aplikacji, 296
- Material, 312

dostawca

- danych CallLog, 414
- treści, 411, 412
 - CalendarContract, 416
 - CallLog, 416
 - ContactsContract, 417, 418, 419
 - innych firm, 422
 - MediaStore, 412
 - Settings, 417
 - UserDictionary, 417
 - VoicemailContract, 417

dostęp

- do plików i katalogów, 112, 383
- do preferencji aplikacji, 111
- do zasobów, 148, 399

dystrybucja aplikacji, 531

dziedziczenie

- motywów, 297
- stylów, 297

dzielenie interfejsu aplikacji, 243

E

Eclipse Android Developer Tools, 62

edycja

- karty, 327
- pliku manifestu, 128

edytor

- układów, 622

ekran typu informacje główne, 123

ekrany z listą opcji, 122

elastyczna grafika, 345

element, *Patrz* znacznik

elementy docelowe, 270

- emulator, 565, 619
 - debugowanie aplikacji, 96
 - konfiguracja położenia GPS, 581
 - kopiowanie plików, 606
 - ograniczenia, 592
 - personalizacja, 591
 - podłączenie debugera, 99
 - polecenia konsoli, 591
 - przesyłanie SMS-ów, 584
 - symulowanie połączeń przychodzących, 582
 - uruchamianie, 575, 577, 580
 - uruchamianie aplikacji, 95
 - usuwanie plików, 606
 - wydajność pracy, 576
- emulator Androida, 74, 77, 86, 87
- encje, 458

F

- filtr
 - danych, 190
 - dzienników, 556
 - InputFilter, 187
 - intencji, 135
 - intencji, intent filter, 121
- Fire OS, 57
- firmware, 49
- format Nine-Patch Stretchable Graphics, 626
- formatowanie
 - kodu, 559
 - tekstów, 153
- formaty graficzne, 161
- fragment, 110, 118, 243, 343
 - ListFragment, 250
 - WebViewFragment, 252
- fragmenty
 - funkcjonalne, 259
 - zagnieżdżone, 259

G

- Git, 444
- GMS, Google Mobile Services, 41
- Google Play
 - aktualizowanie aplikacji, 541

- karta Ceny i dystrybucja, 538
- konfigurowanie opcji aplikacji, 539
- prywatny kanał, 542
- przesyłanie aplikacji, 535
- przesyłanie materiałów marketingowych, 537
- publikowanie aplikacji, 539
- rejestracja, 532
- rejestracja programistów, 53
- usuwanie aplikacji, 541
- zarządzanie aplikacją, 540
- zwroty aplikacji, 540
- Google Play Game Services, 539
- GPS, 581
- Gradle, 635
 - opakowanie, 640
 - typy budowy, 645
 - ustawienia modułów, 637
 - ustawienia projektu, 637
 - wersje plików APK, 648
- graficzne prezentacje interfejsu użytkownika, 464
- grafika typu Nine-Patch, 345
- grupy uprawnień, permission groups, 140

I

- IDE, 62
- ikona
 - aplikacji, 131, 278, 526
 - listy aplikacji, 97
- implementacja
 - aplikacji SampleMaterial, 312
 - fragmentu ListFragment, 250
 - fragmentu WebViewFragment, 252
 - klasy Dialog, 284
 - klasy ViewHolder, 319
- importowanie aplikacji, 82
- informacje
 - o awariach, 451
 - o defektach, 488
 - o problemach, 98
 - o testowaniu, 502
 - o urządzeniach, 436
 - zwrotne, 463

- inicjowanie kart, 317
- integracja aplikacji, 47
- IntelliJ IDEA, 62, 68
- intencja, 110, 120
 - przekazywanie informacji, 122
- interakcja
 - użytkowników, 459
 - z emulatorem, 585
- interfejs
 - android.content.SharedPreferences, 369
 - InputFilter, 187
- interfejs użytkownika, 179, 211, 342, 468
 - graficzne prezentacje, 464
 - optymalizacja, 625

J

- jakość aplikacji, 474
- Java, 558
 - automatyczne uzupełnianie, 558
 - formatowanie kodu, 559
 - import, 559
 - refaktoryzacja kodu, 560
 - reorganizacja kodu, 562
 - tworzenie klas, 559
 - tworzenie metod, 559
- jawne definiowanie uprawnień, 52
- JDK, Java Development Ki, 61
- jednostki, 159
- język Java, 558
- języki programowania, 53

K

- karta, 273, 317, 402–406
 - Allocation Tracker, 602
 - File Explorer, 605
 - Network Statistics, 604
 - System Information, 609
- kasowanie aktywności, 116
- katalog, 94, 386, 391
- katalogi zasobów, 144

- klasa
 - ActionBar, 281
 - Activity, 112
 - android.content.Context, 387
 - android.util.Log, 100
 - android.widget.Button, 192
 - AppCompatActivity, 117, 316
 - BitmapDrawable, 162
 - Card, 316
 - CheckBox, 194
 - DeleteCardTask, 407
 - Dialog, 284, 286
 - DialogFragment, 286, 287, 289
 - Fragment, 118
 - FrameLayout, 216
 - GridLayout, 216
 - GridView, 216
 - ImageSwitcher, 216
 - LinearLayout, 216
 - ListView, 216
 - MediaStore, 413
 - PreferenceActivity, 374
 - RecyclerView, 216, 317
 - RelativeLayout, 216
 - SampleMaterialActivity, 326, 400
 - SampleMaterialAdapter, 401
 - ScrollView, 216
 - TableLayout, 216
 - TransitionAddActivita, 322
 - ViewHolder, 319
- klasy
 - aktywności, 254
 - pojemników, 232
- klasyczny widok, 93
- kod źródłowy systemu, 39
- kojarzenie danych, 235
- kolory, 157, 298
- kołowe znaczniki aktywności, 202
- komponent ViewPager, 273
- komponenty, 109
- kompozycje projektowe, 462
- komunikaty aplikacji, 102
- konfiguracja
 - aplikacji, 127
 - filtrów intencji, 135

- konfiguracja
 - położenia GPS, 581
 - projektu aplikacji, 89
 - środowiska programistycznego, 61
 - układu, 181
 - urządzenia do debugowania, 65
 - urządzeń, 489
 - własnego systemu, 64
 - właściwości Androida, 642
 - zależności aplikacji, 647
 - konsola, 589
 - konta użytkowników, 53
 - kontekst, 109
 - aplikacji, 110
 - kontekstowy tryb akcji, 282
 - kontrola
 - jakości, 442
 - wersji testowych aplikacji, 449
 - kontrolka, 179
 - AnalogClock, 206
 - Button, 218
 - CheckBox, 191
 - Chronometer, 205
 - DatePicker, 198
 - EditText, 185
 - ListFragment, 236
 - ListView, 236
 - ListView oraz GridView, 232
 - RadioButton, 191, 195
 - RadioGroup, 195
 - RatingBar, 203
 - RecyclerView, 238
 - Spinner, 189
 - SwipeRefreshLayout, 237
 - Switch, 191, 195
 - TextClock, 205
 - TextView, 214, 516
 - TimePicker, 198
 - ToggleButton, 191
 - ToolBar, 237
 - VideoView, 206
 - View, 216, 278
 - kontrolki, 179
 - interfejsu użytkownika, 203
 - SeekBar, 202
 - ViewGroup, 218
 - kopia zapasowa aplikacji, 380
 - kursywa, 153
 - kwalfikatory, 347
 - katalogów, 347
 - zasobów alternatywnych, 348–352
- ## L
- layout, 170
 - licencja, 46
 - liczby całkowite, 157
 - lista aplikacji, 98
 - listy stanów kolorów, 163
- ## Ł
- łańcuchy znaków, 152
- ## M
- Maker Movement, 57
 - maksymalizacja
 - pokrycia testów, 492
 - zgodności aplikacji, 339
 - zysków, 473
 - manifest, 127, 526
 - mapa ekranów, 460
 - mapowanie historii użytkowników, 458
 - Material Design, 311
 - materiały marketingowe, 537
 - mechanizm wdrażania etapami, 541
 - menu, 166, 305
 - główne, 122
 - kontekstowe, 277
 - nadmiarowe paska akcji, 279
 - nawigacyjne, 122
 - opcji, 277
 - wyskakujące, 278
 - Mercurial, 444
 - metoda
 - addCard(), 326
 - dismiss(), 286
 - doInBackground(), 407
 - getAll(), 403
 - getQuantityString(), 155

- onActivityCreated(), 247
- onActivityResult(), 326
- onAttach(), 247
- onCreate(), 114, 214, 247
- onCreateView(), 247
- onDestroy(), 117, 247
- onDestroyView(), 247
- onPause(), 115, 247
- onResume(), 115
- onSaveInstanceState(), 116
- onStart(), 115, 247
- onStop(), 247
- requestPermissions(), 138
- setAdapter(), 235
- setContentViews(), 174
- setFilters(), 187
- setOnClickListener(), 193
- setText(), 214
- show(), 286
- startActivity(), 120
- testPreConditions(), 515
- metodologie tworzenia oprogramowania, 428
- metody
 - kaskadowe, 428
 - klasy Activity, 113
 - klasy android.content.Context, 387
 - klasy android.util.Log, 100
 - wprowadzania, 132
 - zwrotne, 247
- minutnik, 204
- model dystrybucji, 521
- modelowanie
 - dziedzin, 459
 - klas, 459
 - relacji encji, 459
- moduł jądra SELinux, 52
- modyfikacja
 - danych dostawców treści, 419
 - postępu, 202
 - ustawień zasilania, 590
- monitorowanie
 - aktywności wątku aplikacji, 598
 - operacji, 599

- motyw, 296
 - aplikacji, 296
 - Material, 312
- możliwości urządzenia, 132

N

- nagłówki, 376
 - preferencji, 376
- narzędzia, 478, 484
 - Android SDK, 46, 73, 615, 628
 - do testowania, 443, 500
 - programistyczne, 46
- narzędzie
 - Android SDK Manager, 73, 75, 76
 - Android Studio, 73
 - Android Virtual Device Manager, 74
 - AVD Manager, 73
 - Hierarchy Viewer, 624
 - Intention Actions, 562
 - logcat, 620
- nawigacja, 123, 265
 - a fragmenty, 269
 - hierarchiczna w górę, 268
 - na zewnątrz, 268
 - poprzeczna, 266
 - wejścia, 266
 - wstecz, 267
 - zstępująca, 267
- nazwa aplikacji, 131, 526
- numeracja wersji aplikacji, 444

O

- obiekt
 - Card, 323
 - File, 392
- obiekty
 - AdapterView, 235
 - ArrayAdapter, 233
 - CursorAdapter, 234
 - Fragment, 245
- obrazy, 161
 - typu Nine-Patch Stretchable Graphics, 162

- obsługa
 - prywatnych danych, 472
 - zdarzeń, 235
 - zmian konfiguracji, 357
 - oczekiwanie na podłączenie debugera, 99
 - oczyszczanie pamięci, 600
 - odbieranie intencji, 124
 - odczyt
 - plików XML, 390
 - preferencji, 369
 - z pliku, 388, 389
 - odnośniki, 182
 - odstępny, 304
 - odtworzenie wideo, 206
 - odwołania do zasobów, 169, 175
 - ograniczenia
 - emulatora, 592
 - urządzeń, 445
 - HA, Open Handset Alliance, 40
 - licencja GMS, 41
 - możliwości Androida, 43
 - operatorzy, 42
 - producenci urządzeń, 41
 - okienko informacyjne, 193
 - okna
 - dialogowe, 283, 284
 - edytora, 555
 - okno
 - AlertDialog, 285
 - Choose Components, 63
 - CharacterPickerDialog, 286
 - DatePickerDialog, 285
 - Generate Signed APK, 528, 530
 - New Key Store, 529
 - ProgressDialog, 285
 - Select Deployment Target, 102
 - TimePickerDialog, 285
 - określanie
 - obsługiwanych metod wprowadzania, 132
 - obsługiwanych wielkości ekranu, 133
 - rynków docelowych, 450
 - urządzeń docelowych, 437
 - wartości zasobów, 148
 - wymaganych możliwości urządzenia, 132
 - wymagań systemowych, 131
 - opcje
 - Developer Console, 539
 - podpisywania, 643
 - profilu sprzętowego, 574
 - opcjonalne SDK, 72, 73
 - Open-Source Hardware, 57
 - operatorzy telekomunikacyjni, 42
 - opłaty od użytkowników, 523
 - oprogramowanie
 - mobilne, 427
 - układowe, 451
 - wbudowanego, firmware, 49
 - optymalizacja interfejsu użytkownika, 625
 - organizacja
 - interfejsu użytkownika, 215
 - zasobów aplikacji, 354
 - organizowanie
 - preferencji, 376
 - zadań, 558
- P**
- pakiet
 - Android SDK, 71
 - Android Support Library, 256
 - android.widget, 179, 180
 - SDK, 46
 - Support Library, 257
 - pakiety języka Java, 54
 - pasek
 - akcji, 201, 278, 281
 - narzędzi, 281
 - postępu, 199
 - personalizacja emulatora, 591
 - piaskownica, 50
 - pielęgnacja aplikacji, 446, 450
 - pierwsza aplikacja, 81, 88
 - pisanie aplikacji, 53
 - planowanie
 - doświadczeń użytkowników, 455
 - interakcji użytkowników, 459
 - testowania, 442
 - platforma Android, 44, 50

- plik, 386, 391
 - AndroidManifest.xml, 128
 - build.gradle, 526, 637
 - manifestu, 127, 526
 - MyFirstAndroidApp.java, 101
 - R.java, 150
 - strings.xml, 151
- pliki, 94
 - .apk, 527
 - budowy Gradle, 635
 - HPROF, 601
 - nieprzetworzone, 168
 - układów, 253
 - XML, 168, 390
 - zasobów preferencji, 372
- pływający przycisk akcji, 282
- pobieranie
 - danych, 184
 - daty, 197
 - godziny, 197
 - kontekstu aplikacji, 111
 - liczb, 197
 - wszystkich kart, 403
 - zasobów aplikacji, 111, 112
- podgląd hierarchii, 622
- podkreślenie, 153
- podpisywanie, 52, 643
- pogrubienie, 153
- pojemnik, 216, 232, 237
 - CardView, 238
 - ViewPager, 238
- pole
 - tekstowe, 184
 - wyboru, 191
- polecenie Organize imports, 101
- połączenie USB, 103
- powiązania pomiędzy ekranami, 244, 245
- poziom
 - PROTECTION_DANGEROUS, 137
 - PROTECTION_NORMAL, 137
- preferencje, 111, 367
 - aktualizacja, 369
 - dodawanie, 369
 - nagłówki, 376
 - odnajdywanie danych, 371
 - pliki zasobów, 372
 - reagowanie na zmiany, 371
 - usuwanie, 369
 - zarządzanie, 372
- prezentacja układów, 180
- problemy
 - z Android SDK, 68
 - z Android Studio, 67
- proces
 - produkcyjny, 479
 - tworzenia oprogramowania, 427
- profil AVD, 85
- profile ograniczone, 53
- program
 - Android Virtual Device Manager, 568, 580
 - Device Monitor, 595
- programowanie, 47
- programowanie na podstawie testów, TDD, 504
- programowe
 - stosowanie zasobów alternatywnych, 354
 - tworzenie układów, 213
- projektowanie
 - interfejsów użytkownika, 342
 - interfejsu użytkownika, 468
 - testów, 511
 - testów wstępnych, 493
 - układów, 172, 216
 - układów ekranów, 462
 - układu, 173
 - urządzeń, 41
 - zgodnych aplikacji, 339
- prototypy, 464
- przechowywanie
 - grafik, 146
 - plików, 146
 - zasobów aplikacji, 144
 - zasobów innych typów, 147
- przeglądanie
 - dzienników, 620
 - systemu plików, 603
- przejścia, 165
- przejście do edycji karty, 330
- przełączniki, 191
- przenoszenie aplikacji, 452

- przepląwy użytkowników, 459
- przesyłanie prywatnych danych, 472
- przeszukiwanie
 - preferencji, 369
 - projektu, 557
- przewijanie, 237
- przycisk
 - akcji, 191, 279, 282
 - FloatingActionButton, 301, 322, 324
- przypadki użycia, 432
- publikacja aplikacji, 48, 521, 525, 532

R

- refaktoryzacja kodu, 560
- regulamin Google Play, 523
- rejestracja, 100, 527
 - aktywności, 134
 - komponentów aplikacji, 136
 - komunikatów, 611
 - programistów, 53
 - uprawnień, 137
 - w Google Play, 532
 - wymaganych uprawnień, 139
- rekordy
 - aktualizacja, 421
 - dodawanie, 419
 - usuwanie, 421
- relacja jeden-do-jednego, 118
- relacje pomiędzy ekranami, 269
- reorganizacja kodu, 562
- RGB, 157
- rodzaje pojemników, 237
- rozgłaszanie intencji, 124
- rozszerzanie aplikacji, 446
- ryzyko, 437, 440

S

- SafetyNet, 361
- scenariusze nawigacyjne, 266
- scenorysy, 464
- sekwencja animacji przejść, 166
- selekcja możliwości aplikacji, 452
- SELinux, 52

- separatory, 304
- skalowanie obrazka, 628
- skrót klucza RSA, 104
- sprawdzanie asercji, 512
- sprzedaż urządzeń, 43
- SQLite, 397
- stan
 - obecny systemu, 43
 - początkowy urządzenia, 491
- stare aplikacje, 256
- statystyki, 524
 - urządzeń z Androidem, 340
 - wykorzystania sieci, 602
- storyboards, 464
- stos cofnięć, 269
- stosowanie
 - ADB, 621
 - API, 344
 - dostawców treści, 411, 414
 - dostawcy danych CallLog, 414
 - edytora układów, 622
 - fragmentów, 257, 259, 343
 - jednoczesne układów, 231
 - karty Allocation Tracker, 602
 - klas pojemników, 232
 - klasy PreferenceActivity, 374
 - konsoli, 590
 - kontekstu aplikacji, 111
 - kontrolki ListFragment, 236
 - kontrolki ListView, 236
 - Material Design, 311
 - mechanizmów rejestracji, 482
 - menu, 167
 - okien dialogowych, 288
 - podglądu hierarchii, 622
 - pól wyboru, 194
 - preferencji, 367
 - przełączników, 194
 - stylów, 295
 - trybu Layout View, 624
 - typów fragmentów, 247
 - układu GridLayout, 228
 - uprawnień, 136
 - wbudowanych układów, 217
 - zasobów alternatywnych, 354

- zasobów graficznych, 161
- zasobów XML, 168
- zasobów z wymiarami, 159
- zwyczajnych przycisków, 192
- strategie testowania, 494
- style, 295
 - wspomagające Material Design, 313
- Subversion, 444
- sygnalizowanie aktywności, 202
- symboliczny widok, 94
- symulowanie
 - odbieranych połączeń, 586
 - połączeń przychodzących, 608
 - wiadomości SMS, 587, 608
- synchronizacja projektu, 641
- system
 - Gradle, 635
 - plików, 391
 - zarządzania kodem źródłowym, 443
- system operacyjny
 - Cyanogen OS, 57
 - CyanogenMod, 57
 - Fire OS, 57
 - Linux, 50
- szkice, 462
- szkielet aplikacji, 55, 70
- szuflada nawigacyjna, 123, 274

S

- śledzenie cyklu życia obiektów, 286
- środowisko
 - programistyczne, 46, 61, 81
 - testowe, 489
 - wykonawcze aplikacji, 50

T

- tablice łańcuchów znaków, 155
- TDD, Test-Driven Development, 504
- test jednostkowy, 480
- testowanie, 449, 480
 - aktualizacji, 451
 - aplikacji, 441, 448, 487
 - aplikacji klient-serwer, 441

- aplikacji wielojęzycznych, 498
- instalacji, 498
- kopii zapasowych, 498
- na urządzeniach, 493
- opłat w aplikacji, 499
- pakietu, 531
- punktów integracji aplikacji, 497
- środowiska programistycznego, 81
- uaktualnień aplikacji, 497, 498
- usług, 495
- użyteczności, 496
- w emulatorze, 493
- wersji finalnej, 465
- wizualnej atrakcyjności aplikacji, 496
- wydajności, 499
- zdarzeń nieprzewidzianych, 499
- zgodności, 498
- testy
 - funkcjonalne, 494
 - jednostkowe, 503, 507
 - strukturalne, 494
 - wstępne, 493
- tłumaczenie aplikacji, 542
- tożsamość aplikacji, 130, 460
- treści
 - generowane przez aplikację, 383
 - multimedialne, 383
 - pobrane z internetu, 383
- tryb
 - Layout View, 623, 624
 - MODE_APPEND, 385
 - MODE_PRIVATE, 385
 - Pixel Perfect, 623, 626
- tworzenie
 - aplikacji, 47, 448
 - AVD, 570, 574
 - intencji, 121
 - interfejsów użytkownika, 211
 - kart, 402
 - konfiguracji uruchomieniowej, 507
 - oprogramowania, 427
 - metoda dostosowywania, 430
 - metoda kaskadowa, 428
 - metoda najmniejszego wspólnego mianownika, 429

- tworzenie
 - pakietu aplikacji, 527
 - pakietu instalacyjnego, 525
 - pierwszej aplikacji, 88
 - planów testowania, 442
 - plików, 387, 392
 - prywatnych preferencji, 368
 - przypadków użycia, 432
 - testów jednostkowych, 482
 - wspólnych preferencji, 368
- typografia, 305
- typy
 - ekranów, 344
 - fragmentów, 247
 - okien dialogowych, 284
 - zasobów, 145, 152

U

- układ, layout, 170, 211, 300
 - aktywności edycji, 329
 - DrawerLayout, 238
 - FrameLayout, 223
 - GridLayout, 228
 - LinearLayout, 213, 218
 - RelativeLayout, 220
 - TabLayout, 273
 - TableLayout, 226
- układy wbudowane, 217
- ukrywanie okna dialogowego, 287
- umiejscawianie elementów, 211
- unikanie błędów, 484
- upływ czasu, 204
- uprawnienia, 52, 136
 - do plików, 385
 - wymagane, 139
- uruchamianie
 - aktywności, 120, 121
 - aplikacji, 84, 86, 95
 - emulatora, 575, 577, 580
 - zadań, 269
- urządzenia, 433
 - docelowe, 437
 - wirtualne, AVD, 567

- usługa, 110, 123
 - GitHub, 76
- usługi
 - Google, 56
 - platformy, 55
- ustawienia w manifeście, 127, 140
- usterki, 483
- usuwanie
 - karty, 333, 406
 - operacji, 401
 - preferencji, 369
 - rekordów, 421
 - statycznych danych, 117
- użycie
 - aktywności, 112
 - fragmentów, 243
 - intencji, 120
 - JUnit, 503
 - TextView, 181
 - układów, 211
 - wskazników, 199
 - wzorców, 265
- użyteczność aplikacji, 463

W

- wartości
 - atrybutu quantity, 155
 - logiczne, 156
 - RGB, 157
- wbudowane klasy pojemników, 232
- wbudowani dostawcy treści, 412
- wbudowane układy, 217
- wdrażanie aplikacji, 450
- wersje aplikacji, 526, 644, 648
- weryfikacja uprawnień aplikacji, 527
- wideo, 206
- widok, 179
 - Android, 93
 - Project, 93
- widoki przeciągane, 271
- widżet, 180
 - TextInputLayout, 301
 - Toolbar, 302
- wielkości ekranu, 133

wizualne informacje zwrotne, 463
 własne okna dialogowe, 288
 własność intelektualna, 522
 wskaźniki, 199
 wsparcie
 aplikacji, 450
 dla fragmentów, 256
 dla typów ekranów, 344
 współdziałanie aplikacji, 447
 współrzędne geograficzne, 609
 wybieranie elementów, 235
 wybór modelu dystrybucji, 521
 wydajność emulatora, 576
 wygląd markowy aplikacji, 303
 wymagania
 aplikacji, 429, 440
 projektowe, 429
 systemowe, 131
 użytkowników, 468
 wymiary, 158
 wyniki testów, 513
 wyodrębnianie
 metod, 561
 zmiennych, 560
 wyświetlanie
 czasu, 205
 fragmentów pliku, 553
 oceny, 203
 okien edytora, 553
 okna dialogowego, 287
 tekstów, 181
 wzorce projektowe nawigacji, 270

Z

zakładka Emulator Control, 607
 zależności, 313
 zamykanie kart, 554
 zapisywanie plików, 392
 zarządzanie
 bazą danych urządzeń, 433
 kodem źródłowym, 443
 konfiguracjami, 443
 konfiguracjami urządzeń, 489
 kontaktami, 121

modyfikacjami fragmentów, 246
 plikami, 384, 603
 preferencjami, 372
 stanem aplikacji, 113
 środowiskiem testowym, 489
 tożsamością aplikacji, 130
 zasobami, 113, 143
 zmianami aktywności, 120
 zasoby
 alternatywne, 147, 348, 345
 aplikacji, 111, 112, 143
 animacje poklatkowe, 165
 animacje przejęć, 165
 definiujące układy, 174
 domyślne katalogi, 144
 dostęp programowy, 148
 graficzne, 160
 łańcuchy znaków, 152, 314
 menu, 166
 przechowywanie, 146
 typy, 152
 typy wartości, 144
 XML, 168
 z kolorami, 157, 314
 z liczbami całkowitymi, 157
 z wartościami logicznymi, 156
 z wymiarami, 158, 159
 preferencji, 372
 systemowe, 175
 układów, 315
 zastosowanie filtrów, 186
 zdarzenia, 235
 zgodność
 aplikacji, 339
 interfejsów użytkownika, 342
 wsteczna, 393
 z SafetyNet, 361
 zintegrowane środowisko programistyczne,
 IDE, 62
 zmiana
 konfiguracji, 357
 położenia okien, 552
 stanu telefonii, 607
 wielkości okna edytora, 552
 wielkości okna Tools, 553

znacznik

<activity>, 304

<compatible-screens>, 134

<include>, 300, 301

<merge>, 232, 300

<plurals>, 154

<selector>, 163

<supports-gl-texture>, 134

<uses-configuration>, 132

znak #, 157

zrzut ekranu, 610

zyski z reklam, 524

Ż

żądanie przydzielania uprawnień, 137

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>



Android – ogranicza go tylko Twoja wyobraźnia!

Android jest otwartą, kompletną i darmową platformą mobilną. Pod jego kontrolą pracują setki milionów urządzeń, przy czym nie są to jedynie tablety i smartfony. Mogą to być właściwie wszystkie urządzenia, do których da się włożyć procesor. Rynek aplikacji mobilnych dla Androida to raj dla programistów, jednak sukces zależy od pomysłu, talentu i wysokiej jakości tworzonych aplikacji.

Niniejsza książka to bezcenne kompendium dla każdego programisty piszącego aplikacje na Androida. Będzie niezastąpiona dla początkujących, którzy dzięki niej zyskają solidne podstawy, ale docenią ją również poważni projektanci profesjonalnych aplikacji. Kompleksowo prezentuje środowisko Android Studio oraz Android SDK, opisuje nowy system uprawnień w Android 6.0, przedstawia sposób wykorzystania bazy SQLite, pokazuje sporo sztuczek i porad związanych z programowaniem na Androida – a to wszystko, by możliwie najpełniej i najstarszanniej wyeksponować aktualne i najbardziej interesujące możliwości tej platformy.

Najważniejsze zagadnienia ujęte w książce:

- podstawy platformy Android, jej architektura i zasady funkcjonowania
- architektura aplikacji mobilnej i jej cechy szczególne
- planowanie procesu wytwarzania aplikacji mobilnych i prowadzenie kontroli jakości
- Material Design i jego znaczenie
- pisanie aplikacji, jej testowanie i debugowanie oraz publikacja
- strategię tworzenia oprogramowania dla Androida

Joseph Annuzzi Jr. jest programistą, grafikiem, przedsiębiorcą i autorem książek. Biegłe posługuje się różnymi językami programowania, interesuje się kryptografią, algorytmami biometrycznymi i tworzeniem grafik 3D. Mieszka w Dolinie Krzemowej. **Lauren Darcy** jest profesjonalną programistką i uznanym autorytetem w dziedzinie architektury aplikacji oraz aplikacji mobilnych. **Shane Conder** od ponad dziesięciu lat pisze aplikacje dla środowisk mobilnych i osadzonych. Zaprojektował i napisał wiele komercyjnych aplikacji na Androida, iPhone'a, BREW, BlackBerry, Palm oraz dla środowisk J2ME i Windows Mobile.

Helion	
księgarnia internetowa	
http://helion.pl	
zamówienia telefoniczne	
	0 801 339900
	0 601 339900
Informatyka w najlepszym wydaniu	

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

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

ISBN 978-83-283-2612-5



9 788328 326125

cena: 99,00 zł

sięgnij po WIĘCEJ



KOD KORZYŚCI


Addison
Wesley