

A 3D rendered bicycle seat and its mounting post, shown in a light blue color against a darker blue background.

WYDANIE III

Programowanie aplikacji dla Androida

BIG NERD RANCH GUIDE

A 3D rendered bicycle wheel with a knobby tire and a pedal, shown in a light blue color against a darker blue background.

Bill Phillips, Chris Stewart, Kristin Marsicano

Tytuł oryginału: Android Programming: The Big Nerd Ranch Guide (3rd Edition)

Tłumaczenie: Grzegorz Kowalczyk

ISBN: 978-83-283-3636-0

Authorized translation from the English language edition, entitled: ANDROID PROGRAMMING: THE BIG NERD RANCH GUIDE, Third Edition; ISBN 0134706056; by Bill Phillips; and by Chris Stewart; and by Kristin Marsicano; published by Pearson Education, Inc; publishing as The Big Nerd Ranch Guides.

Copyright © 2017 by Big Nerd Ranch, LLC.

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 © 2017.

The 10-gallon hat with propeller logo is a trademark of Big Nerd Ranch, Inc.

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/prapan>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/prapan.zip>

Printed in Poland.

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

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

Spis treści

Poznanwanie systemu Android	19
Wymagania wstępne	20
Co nowego w trzecim wydaniu?	20
Jak korzystać z tej książki?	20
Jak zorganizowana jest ta książka	21
Wyzwania	22
Czy jesteś dociekliwy?	22
Styl kodowania	22
Konwencje typograficzne	23
Wersje systemu Android	23
Niezbędne narzędzia	25
Pobieranie i instalowanie Android Studio	25
Pobieranie starszych wersji środowiska SDK	25
Urządzenie sprzętowe	26
1 Twoja pierwsza aplikacja dla systemu Android	27
Podstawowe elementy aplikacji	28
Tworzenie projektu aplikacji dla systemu Android	29
Poruszanie się w Android Studio	33
Tworzenie układu interfejsu użytkownika	34
Hierarchia widoków	38
Atrybuty widgetów	39
Tworzenie zasobów tekstowych	40
Podgląd układu	41
Od pliku układu XML do obiektów klasy View	42
Zasoby i identyfikatory zasobów	43
Podłączanie widgetów	45
Pobieranie odwołań do widgetów	46
Tworzenie obiektów nasłuchujących	47
Tworzenie komunikatów toast	49
Zastosowanie mechanizmu dopełniania kodu	50
Uruchamianie aplikacji w emulatorze	51
Dla dociekliwych: proces budowania aplikacji	55
Narzędzia wspomagające budowanie aplikacji	56
Wyzwania	56
Wyzwanie: dostosowywanie komunikatów toast do własnych potrzeb	57

2	System Android i wzorzec MVC	59
	Tworzenie nowej klasy	60
	Generowanie getterów i setterów.....	61
	Wzorzec MVC a system Android.....	63
	Zalety stosowania wzorca MVC.....	64
	Aktualizacje warstwy widoku	65
	Aktualizacja warstwy kontrolera.....	67
	Uruchamianie aplikacji na urządzeniu fizycznym	71
	Podłączanie urządzenia fizycznego.....	71
	Konfigurowanie urządzenia do pracy ze środowiskiem deweloperskim	72
	Dodawanie ikon.....	74
	Dodawanie nowych zasobów do projektu aplikacji	75
	Odwołania do zasobów w kodzie XML.....	77
	Wyzwanie: dodaj listener do widoku TextView	78
	Wyzwanie: dodaj przycisk Poprzednie	78
	Wyzwanie: od przycisku Button do przycisku ImageButton.....	79
3	Cykl życia aktywności.....	81
	Logowanie cyklu życia aktywności	83
	Tworzenie dziennika komunikatów	83
	Zastosowanie panelu Logcat.....	85
	Badanie cyklu życia aktywności w praktyce	87
	Zmiana orientacji urządzenia a cykl życia aktywności	90
	Konfiguracje urządzeń i zasoby alternatywne.....	90
	Zapisywanie danych przy zmianie orientacji urządzenia.....	95
	Nadpisywanie metody onSaveInstanceState(Bundle)	95
	Jeszcze kilka słów o cyklu życia aktywności	97
	Dla dociekliwych: bieżący proces czyszczenia aktywności.....	99
	Dla dociekliwych: poziomy logowania i odpowiadające im metody.....	100
	Wyzwanie: zapobieganie wielokrotnym odpowiedziom na to samo pytanie	101
	Wyzwanie: ocena poziomu poprawności odpowiedzi	101
4	Debugowanie aplikacji dla systemu Android	103
	Wyjątki i ślad stosu	104
	Diagnozowanie problemów	106
	Logowanie śladów stosu	106
	Ustawianie punktów przerwania w kodzie.....	108
	Zastosowanie pułapek z wyjątkami	111
	Mechanizmy debugowania specyficzne dla systemu Android.....	112
	Korzystanie z programu Android Lint.....	112
	Problemy z klasą R	114
	Wyzwanie: zastosowanie narzędzia Layout Inspector	115
	Wyzwanie: śledzenie alokacji zasobów pamięci	116

5	Twoja druga aktywność	119
	Konfigurowanie drugiej aktywności.....	120
	Tworzenie nowej aktywności.....	121
	Podklasa nowej aktywności.....	124
	Deklarowanie aktywności w manifeście aplikacji.....	124
	Dodawanie przycisku podpowiedzi do aktywności QuizActivity.....	125
	Uruchamianie aktywności	127
	Komunikowanie się z intencjami.....	128
	Przekazywanie danych między intencjami.....	129
	Używanie danych typu extras w intencjach	130
	Pobieranie wyników z aktywności podrzędnej.....	134
	Jak system Android widzi Twoje aktywności.....	138
	Wyzwanie: zamykanie luk dla oszustów.....	141
6	Wersje Android SDK i kompatybilność aplikacji	143
	Wersje środowiska Android SDK.....	143
	Kompatybilność aplikacji z różnymi wersjami systemu Android	144
	Rozsądne minimum	145
	Minimalna wersja SDK.....	147
	Docelowa wersja SDK.....	147
	Wersja kompilacji SDK	147
	Bezpieczne dodawanie kodu z nowszych wersji API.....	148
	Korzystanie z dokumentacji dla deweloperów aplikacji systemu Android.....	151
	Wyzwanie: raportowanie wersji SDK.....	153
	Wyzwanie: ograniczenie podpowiadania	154
7	Fragmenty w interfejsie użytkownika i menedżer fragmentów	155
	Potrzeba elastyczności interfejsu użytkownika.....	155
	Wprowadzenie do zastosowania fragmentów	157
	Uruchamianie aplikacji CriminalIntent.....	158
	Tworzenie nowego projektu	160
	Dwa typy fragmentów.....	162
	Dodawanie zależności w Android Studio	162
	Tworzenie klasy Crime.....	165
	Hostowanie fragmentów interfejsu użytkownika.....	166
	Cykl życia fragmentu	167
	Dwa podejścia do hostowania fragmentów	168
	Definiowanie widoku kontenera.....	168
	Tworzenie fragmentu interfejsu użytkownika	170
	Definiowanie układu fragmentu CrimeFragment.....	170
	Tworzenie klasy CrimeFragment.....	172
	Dodawanie fragmentów interfejsu użytkownika do instancji klasy FragmentManager	177
	Transakcje fragmentów	179
	FragmentManager i cykl życia fragmentów	180
	Architektura aplikacji wykorzystującej fragmenty	182
	Powód, dla którego wszystkie nasze aktywności używają fragmentów.....	182
	Dla dociekliwych: fragmenty i biblioteka wsparcia.....	184
	Dla dociekliwych: dlaczego fragmenty z biblioteki wsparcia są bardziej użyteczne?.....	185

8	Wyświetlanie list w widoku RecyclerView.....	187
	Aktualizacja warstwy modelu aplikacji CriminalIntent.....	188
	Wzorzec Singleton i scentralizowane przechowywanie danych.....	189
	Abstrakcyjna aktywność do przechowywania fragmentów.....	191
	Ogólny układ przechowujący fragmenty.....	191
	Abstrakcyjna klasa Activity.....	192
	Klasy RecyclerView, Adapter oraz ViewHolder.....	197
	Klasy ViewHolder i Adapter.....	198
	Adaptery.....	199
	Zastosowanie kontenera RecyclerView.....	201
	Widok do wyświetlenia.....	202
	Implementowanie klas ViewHolder i Adapter.....	203
	Dowiązkiwanie elementów listy.....	206
	Reagowanie na naciśnięcie.....	208
	Dla dociekliwych: kontenery ListView i GridView.....	209
	Dla dociekliwych: singletony.....	210
	Wyzwanie: typy widoków kontenera RecyclerView.....	211
9	Tworzenie interfejsów użytkownika z układami i widgetami.....	213
	Korzystanie z graficznego narzędzia tworzenia układów.....	214
	Wprowadzenie do dynamicznych układów interfejsu — ConstraintLayout.....	215
	Korzystanie z układu ConstraintLayout.....	216
	Edytor graficzny.....	217
	Tworzenie miejsca dla widoków.....	218
	Dodawanie widgetów.....	221
	Wewnętrzne ustawienia układu ConstraintLayout.....	224
	Edytowanie właściwości.....	225
	Tworzenie dynamicznych elementów listy.....	228
	Jeszcze kilka słów o atrybutach układu.....	229
	Gęstość pikseli ekranu i jednostki dp oraz sp.....	229
	Marginesy i odstępy.....	231
	Style, motywy i atrybuty motywów.....	232
	Wytyczne dla deweloperów aplikacji dla systemu Android.....	233
	Graficzne narzędzia do tworzenia układów.....	233
	Wyzwanie: formatowanie daty.....	233
10	Zastosowanie argumentów fragmentów.....	235
	Uruchamianie aktywności przez fragment.....	235
	Dodawanie danych extras.....	237
	Pobieranie danych extras.....	237
	Wypełnianie widoku fragmentu CrimeFragment danymi z obiektu Crime.....	238
	Wady bezpośredniego pobierania danych.....	239
	Argumenty fragmentu.....	240
	Przypisywanie argumentów do fragmentu.....	240
	Pobieranie argumentów.....	241

Przeładowywanie listy.....	242
Pobieranie wyników działania fragmentów	244
Dla dociekliwych: dlaczego używamy argumentów fragmentów?.....	245
Wyzwanie: efektywne przeładowywanie zawartości kontenera RecyclerView.....	246
Wyzwanie: ulepszenie wydajności działania obiektu CrimeLab.....	246
11 Zastosowanie klasy ViewPager	247
Tworzenie klasy CrimePagerActivity.....	248
Obiekty ViewPager i PagerAdapter.....	249
Integrowanie aktywności CrimePagerActivity	251
FragmentManagerAdapter kontra FragmentPagerAdapter	253
Dla dociekliwych: jak naprawdę działa ViewPager	255
Dla dociekliwych: rozmieszczanie widoków z poziomu kodu	256
Wyzwanie: odtwarzanie marginesów fragmentu CrimeFragment.....	257
Wyzwanie: dodawanie przycisków Pierwsza i Ostatnia	257
12 Okna dialogowe	259
Tworzenie instancji klasy DialogFragment	261
Wyświetlanie fragmentu DialogFragment	263
Dodawanie zawartości okna dialogowego.....	265
Przekazywanie danych między fragmentami.....	267
Przekazywanie danych do fragmentu DatePickerFragment.....	268
Zwracanie danych do fragmentu CrimeFragment.....	270
Wyzwanie: więcej okien dialogowych.....	278
Wyzwanie: bardziej responsywny DialogFragment.....	278
13 Pasek narzędzi.....	279
Biblioteka AppCompatActivity.....	280
Używanie biblioteki AppCompatActivity	280
Menu	282
Definiowanie menu w kodzie XML.....	282
Tworzenie menu.....	288
Reagowanie na wybranie elementu menu	291
Włączanie nawigacji hierarchicznej	292
Jak działa nawigacja hierarchiczna	293
Alternatywny przycisk akcji.....	294
Przełączanie tytułu przycisku akcji.....	295
„Jeszcze tylko jedna sprawa...”	297
Dla dociekliwych: pasek akcji a pasek narzędzi	299
Wyzwanie: usuwanie przestępstw z listy.....	300
Wyzwanie: zasoby tekstowe typu plural	300
Wyzwanie: pusty widok dla kontenera RecyclerView	300
14 Bazy danych SQLite.....	301
Definiowanie schematu.....	302
Tworzenie szkieletu bazy danych.....	303
Eksploracja plików przy użyciu Android Device Monitor.....	306
Debugowanie problemów z bazą danych	307

Patroszymy CrimeLab	308
Zapisywanie danych w bazie.....	310
Zastosowanie klasy ContentValues	310
Wstawianie i aktualizowanie wierszy tabeli	311
Odczytywanie danych z bazy.....	313
Zastosowanie klasy CursorWrapper	314
Zamiana na obiekty modelu	316
Dla dociekliwych: więcej baz danych	319
Dla dociekliwych: kontekst aplikacji	320
Wyzwanie: usuwanie przestępstw z listy.....	320
15 Intencje niejawne.....	321
Dodawanie przycisków.....	322
Dodawanie podejrzanego do warstwy modelu	323
Zastosowanie ciągów formatujących.....	325
Zastosowanie niejawnych intencji	326
Elementy składowe niejawnej intencji	327
Wysyłanie raportu o przestępstwie.....	328
Korzystanie z listy kontaktów.....	330
Wyszukiwanie aktywności zdolnych do wykonania danego zadania.....	334
Wyzwanie: klasa ShareCompat	337
Wyzwanie: kolejna intencja niejawna	337
16 Wykonywanie zdjęć przy użyciu intencji	339
Miejsce do przechowywania zdjęć	339
Miejsce na przechowywanie plików	342
Zastosowanie klasy FileProvider	343
Określanie lokalizacji zdjęć	344
Zastosowanie intencji do uruchomienia aparatu fotograficznego.....	345
Uruchamianie intencji	346
Skalowanie i wyświetlanie bitmap	348
Deklarowanie wymagań aplikacji	351
Wyzwanie: wyświetlanie zdjęcia w pełnym rozmiarze	351
Wyzwanie: efektywne ładowanie miniaturki zdjęć	351
17 Dwupanelowy interfejs typu lista-szczegóły	353
Dodawanie elastyczności do układu.....	354
Modyfikowanie klasy SingleFragmentActivity	355
Tworzenie układu z dwoma kontenerami dla fragmentów	356
Używanie aliasów zasobów	358
Tworzenie alternatywy dla tableatów.....	359
Aktywność: nadzorca fragmentów	360
Interfejsy zwrotne układów	361
Dla dociekliwych: jeszcze kilka słów o sprawdzaniu rozmiarów ekranu urządzenia.....	369
Wyzwanie: dodawanie funkcji „przeciągnij, aby usunąć”	370

18	Lokalizacja	371
	Lokalizacja zasobów.....	372
	Zasoby domyślne.....	375
	Sprawdzanie pokrycia zasobów przy użyciu edytora tłumaczeń Translations Editor.....	378
	Dostosowywanie ustawień regionalnych.....	379
	Kwalifikatory konfiguracji.....	382
	Priorytetyzacja zasobów alternatywnych.....	383
	Wiele kwalifikatorów.....	385
	Odszukiwanie zasobów najlepiej dopasowanych do bieżącej konfiguracji.....	386
	Testowanie zasobów alternatywnych.....	387
	Wyzwanie: lokalizacja dat.....	388
19	Ułatwienia dostępu	389
	Usługa TalkBack.....	389
	Eksploracja przez dotyk.....	392
	Nawigacja liniowa poprzez przeciąganie palcem po ekranie.....	393
	Jak spowodować, aby TalkBack odczytywał elementy nietekstowe?.....	395
	Dodawanie opisów zawartości.....	395
	Włączanie możliwości ustawienia fokusu dla wybranego widoku.....	397
	Jak zapewnić zbliżoną funkcjonalność aplikacji z ułatwieniami dostępu.....	398
	Zastosowanie etykiet do udostępniania kontekstu.....	400
	Dla dociekliwych: korzystanie z aplikacji Accessibility Scanner.....	403
	Wyzwanie: poprawianie listy.....	406
	Wyzwanie: dostarczanie kontekstu dla elementu danych.....	406
	Wyzwanie: powiadomienia o zdarzeniach.....	406
20	Wiązanie danych i model MVVM	409
	Różne architektury — dlaczego warto sobie tym zaprzętać głowę.....	409
	Tworzenie aplikacji BeatBox.....	411
	Proste wiązanie danych.....	412
	Importowanie zasobów typu assets.....	416
	Pobieranie zasobów assets.....	418
	Podłączanie zasobów typu asset.....	420
	Wiązanie do danych.....	423
	Tworzenie modelu widoku.....	424
	Wiązanie do modelu widoku.....	425
	Observable data.....	428
	Korzystanie z zasobów typu asset.....	430
	Dla dociekliwych: jeszcze kilka słów o wiązaniu danych.....	431
	Wyrażenia lambda.....	431
	Jeszcze więcej cukru syntaktycznego.....	431
	BindingAdapter.....	432
	Dla dociekliwych: dlaczego zasoby typu asset, a nie zwykłe zasoby aplikacji?.....	432
	Dla dociekliwych: zasoby typu non-asset?.....	433

21	Testy jednostkowe i odtwarzanie plików audio	435
	Tworzenie klasy SoundPool.....	435
	Ładowanie plików dźwiękowych	436
	Odtwarzanie plików dźwiękowych	438
	Zależności testów.....	439
	Tworzenie klasy testowej.....	440
	Konfigurowanie testu	442
	Zastosowanie zależności pozornych.....	443
	Pisanie testów.....	444
	Testowanie interakcji obiektów.....	445
	Wywołania zwrotne wiązania danych.....	448
	Usuwanie plików dźwięków z pamięci.....	449
	Zmiana orientacji urządzenia a ciągłość istnienia obiektów.....	450
	Zachowywanie fragmentów.....	451
	Obracanie urządzenia a zachowywanie fragmentów.....	452
	Dla dociekliwych: czy zachowywać fragmenty?.....	453
	Dla dociekliwych: Espresso i testy integracyjne.....	455
	Dla dociekliwych: objekty pozorne i testowanie.....	456
	Wyzwanie: sterowanie szybkością odtwarzania	457
22	Style i motywy	459
	Zasoby kolorów	460
	Style	460
	Dziedziczenie stylów	462
	Motywy aplikacji	463
	Modyfikowanie motywu	464
	Dodawanie kolorów motywu	466
	Nadpisywanie atrybutów motywu	467
	Eksplorowanie motywu	468
	Modyfikowanie atrybutów przycisków.....	471
	Dla dociekliwych: jeszcze kilka słów o dziedziczeniu stylów.....	474
	Dla dociekliwych: dostęp do atrybutów motywu	475
23	Elementy XML drawable	477
	Tworzenie przycisków o jednolitym wyglądzie	478
	Elementy typu shape drawable.....	479
	Lista stanów.....	481
	Lista warstw.....	482
	Dla bardziej dociekliwych: po co zwracać sobie głowę elementami XML drawable?	483
	Dla bardziej dociekliwych: obrazy Mipmap	484
	Dla dociekliwych: obrazy 9-patch.....	485
	Wyzwanie: motywy przycisków	492
24	Więcej o intencjach i zadaniach	493
	Tworzenie aplikacji NerdLauncher	494
	Rozwiązywanie intencji niejawnych.....	496
	Tworzenie jawnych intencji w czasie działania programu.....	500

Zadania i stos aktywności.....	502
Przełączanie między zadaniami.....	503
Uruchamianie nowego zadania.....	504
Użycie launchera NerdLauncher jako ekranu głównego	507
Wyzwanie: ikony	507
Dla dociekliwych: procesy kontra zadania	508
Dla dociekliwych: dokumenty równoległe	511
25 HTTP i zadania drugoplanowe.....	515
Tworzymy aplikację PhotoGallery.....	517
Podstawowe zagadnienia sieciowe.....	519
Uzyskiwanie uprawnień do korzystania z połączeń sieciowych	521
Uruchamianie zadań asynchronicznych AsyncTask w wątku tła	521
Ty i Twój wątek główny	523
Poza głównym wątkiem.....	524
Pobieranie danych w formacie JSON z serwisu Flickr.....	525
Parsowanie danych zapisanych w formacie JSON	529
Od zadania AsyncTask z powrotem do wątku głównego.....	532
Porządkowanie zadań asynchronicznych AsyncTask.....	535
Dla dociekliwych: jeszcze kilka słów o zadaniach AsyncTask	536
Dla dociekliwych: alternatywy dla zadań AsyncTask	537
Wyzwanie: Gson.....	538
Wyzwanie: podział na strony.....	538
Wyzwanie: dynamiczne modyfikowanie liczby wyświetlanych kolumn	539
26 Obiekty Looper, Handler i HandlerThread.....	541
Przygotowanie kontenera RecyclerView do wyświetlania obrazów	541
Pobieranie wielu małych rzeczy	544
Komunikacja z wątkiem głównym	544
Tworzenie wątku tła.....	546
Komunikaty i handlery komunikatów	548
Anatomia komunikatu	548
Anatomia handlera.....	549
Używanie handlerów	549
Przekazywanie handlerów.....	554
Dla dociekliwych: zadania AsyncTask kontra wątki	559
Dla dociekliwych: rozwiązywanie problemu z pobieraniem zdjęć	560
Dla dociekliwych: tryb StrictMode	561
Wyzwanie: ładowanie wstępne i buforowanie danych	562
27 Wyszukiwanie	563
Wyszukiwanie zdjęć w serwisie Flickr.....	564
Zastosowanie widoku SearchView.....	568
Reagowanie na interakcję widoku SearchView z użytkownikiem	571
Proste zachowywanie danych w plikach preferencji.....	573
Wyglądanie aplikacji.....	577
Wyzwanie: jeszcze bardziej ulepszymy aplikację.....	578

28	Usługi działające w tle.....	579
	Tworzenie usługi IntentService.....	579
	Do czego służą usługi.....	582
	Operacje sieciowe bezpiecznie działające w tle.....	582
	Sprawdzanie, czy pojawiły się nowe wyniki wyszukiwania	584
	Opóźnione wykonywanie z wykorzystaniem usługi AlarmManager	585
	Jak używać alarmów we właściwy sposób.....	588
	Obiekty PendingIntent	589
	Zarządzanie alarmami przy użyciu PendingIntent	590
	Sterowanie alarmami	590
	Powiadomienia	593
	Wyzwanie: powiadomienia na urządzeniach typu Android Wear	595
	Dla dociekliwych: szczegóły usługi.....	596
	Czym zajmują się usługi (a czego nie powinny robić).....	596
	Cykl życia usługi.....	596
	Usługi typu non-sticky	597
	Usługi typu sticky	597
	Wiązanie usług.....	598
	Dla dociekliwych: klasy JobScheduler oraz JobServices	599
	JobScheduler i przyszłość zadań drugoplanowych.....	602
	Wyzwanie: zastosowanie klasy JobService w systemie Lollipop	603
	Dla bardziej dociekliwych: adaptory synchronizacji.....	603
29	Intencje rozgłoszeń	605
	Intencje a intencje rozgłoszeń	605
	Odbieranie rozgłoszeń systemowych — aktywowanie podczas uruchamiania systemu.....	606
	Tworzenie i rejestrowanie samodzielnego odbiornika rozgłoszeń	606
	Używanie odbiorników rozgłoszeń	609
	Filtrowanie powiadomień pierwszoplanowych	611
	Wysyłanie intencji rozgłoszenia.....	611
	Tworzenie i rejestrowanie odbiornika dynamicznego.....	612
	Ograniczanie zasięgu rozgłoszeń tylko do naszej aplikacji przy użyciu prywatnych uprawnień	614
	Przekazywanie i odbieranie danych przy użyciu rozgłoszeń uporządkowanych	617
	Odbiorniki i zadania działające przez długi czas	622
	Dla dociekliwych: zdarzenia lokalne	622
	Używanie biblioteki EventBus.....	623
	Używanie biblioteki RxJava	624
	Dla dociekliwych: sprawdzanie, czy fragment jest widzialny	624
30	Przeglądanie sieci WWW i widoki WebView	627
	Ostatni kawałek danych z serwisu Flickr	628
	Prosty sposób — intencje niejawne	630
	Trudniejszy sposób — widok WebView.....	631
	Zastosowanie interfejsu WebChromeClient do ulepszenia wyświetlania stron internetowych	635

Zmiana orientacji urządzenia a widok WebView	638
Niebezpieczeństwa związane z obsługą zmian konfiguracji	639
Dla dociekliwych: wstrzykiwanie obiektów JavaScript	639
Dla dociekliwych: aktualizacje widoku WebView	640
Wyzwanie: używanie przycisku Wstecz do obsługi historii przeglądania	641
Wyzwanie: obsługa łączy innych niż HTTP	641
31 Widoki niestandardowe i zdarzenia związane z dotykiem	643
Konfigurowanie projektu aplikacji DragAndDraw	644
Tworzenie własnego, niestandardowego widoku	645
Tworzenie widoku BoxDrawingView	646
Obsługa zdarzeń związanych z dotykiem	648
Śledzenie zdarzeń związanych z ruchem palca po ekranie	649
Renderowanie z użyciem metody onDraw(Canvas)	652
Wyzwanie: zapisywanie stanu	653
Wyzwanie: obracanie prostokątów	654
32 Animacja właściwości	657
Budowanie sceny	657
Proste animacje	659
Właściwości transformacji widoków	663
Używanie różnych interpolatorów	665
Modyfikowanie kolorów	665
Uruchamianie kilku animatorów jednocześnie	668
Dla dociekliwych: inne API animacji	669
Starsze narzędzia animacyjne	669
Transformacje	669
Wyzwania	670
33 Lokalizacja i usługi Play	671
Lokalizacja i biblioteki	672
Usługi Google Play	672
Tworzenie aplikacji Locatr	673
Google Play Services a testowanie usług lokalizacji w emulatorze	673
Symulowane dane lokalizacji	675
Budowanie aplikacji	677
Konfigurowanie Google Play Services	680
Uprawnienia do wyznaczania lokalizacji	681
Używanie Google Play Services	682
Wyszukiwanie zdjęć w serwisie Flickr z użyciem geolokalizacji	684
Pobieranie informacji o bieżącej lokalizacji	685
Żądanie udzielenia uprawnień po uruchomieniu aplikacji	688
Sprawdzanie uprawnień	688
Wyszukiwanie i wyświetlanie zdjęć z użyciem geolokalizacji	693
Wyzwanie: uzasadnianie prośby o udzielenie uprawnień	695
Wyzwanie: wskaźnik postępu wyszukiwania	696

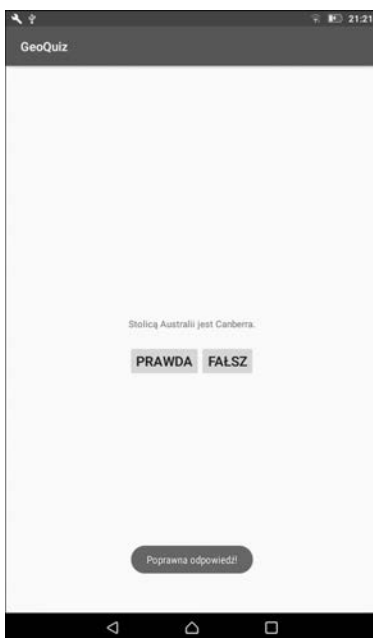
34	Mapy	697
	Importowanie biblioteki Play Services Maps	697
	Korzystanie z map w systemie Android.....	697
	Pobieranie klucza Maps API dla map Google.....	698
	Konfigurowanie map	700
	Pobieranie dodatkowych danych o lokalizacji.....	701
	Praca z mapą	704
	Rysowanie na mapie.....	707
	Dla dociekliwych: zespoły programistów i klucze API.....	708
35	Interfejs Material Design	711
	Warstwy materiałów	712
	Elewacja i wartości osi Z.....	713
	Animatory listy stanów.....	715
	Narzędzia animacji.....	717
	Animacje typu circular reveal.....	717
	Transformacje współdzielonego elementu.....	719
	Komponenty widoków	722
	Karty	722
	Przyciski FAB.....	724
	Powiadomienia Snackbar	725
	Jeszcze kilka słów o interfejsie Material Design.....	727
36	Posłowie	729
	Wyzwanie finałowe	729
	Trochę bezwstydnego prywaty	730
	Dziękujemy!	730
	Skorowidz.....	731

Twoja pierwsza aplikacja dla systemu Android

W tym rozdziale omówimy szereg podstawowych zagadnień, których znajomość jest niezbędna do zbudowania aplikacji dla systemu Android. Nie powinieneś się jednak przejmować, jeżeli po przeczytaniu tego rozdziału nie będziesz czegoś rozumiał, ponieważ w kolejnych rozdziałach tej książki będziemy wielokrotnie powracać do omawianych tutaj tematów i analizować je w znacznie bardziej szczegółowy sposób.

Aplikacja, którą utworzymy na początek, nosi nazwę GeoQuiz. Zadaniem tej aplikacji jest sprawdzanie wiedzy użytkownika na temat geografii. Użytkownik odpowiada na pytania pojawiające się na ekranie, naciskając przyciski *PRAWDA* albo *FALSZ*, a GeoQuiz natychmiast sprawdza wprowadzane odpowiedzi.

Na rysunku 1.1 przedstawiono wygląd ekranu aplikacji po wybraniu przez użytkownika poprawnej odpowiedzi.



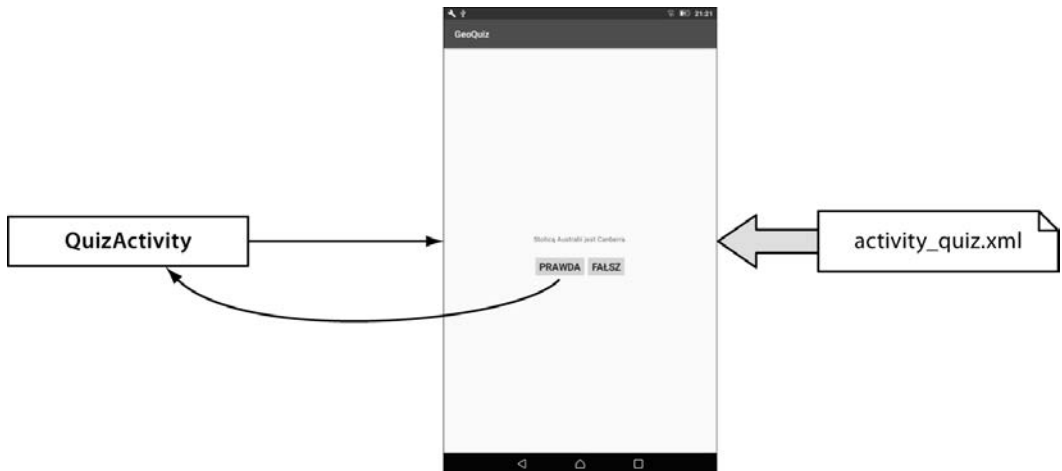
Rysunek 1.1 Czy pochodzisz z antypodów?

Podstawowe elementy aplikacji

Aplikacja GeoQuiz będzie się składała z **aktywności** (ang. *activity*) oraz **układu** (ang. *layout*).

- **Aktywność** jest instancją klasy `Activity`, wchodzącej w skład biblioteki Android SDK. Aktywności odpowiadają za zarządzanie interakcjami z użytkownikiem na ekranie aplikacji. Aby zaimplementować poszczególne funkcjonalności aplikacji, będziesz tworzył odpowiednie podklasy klasy `Activity`. W prostych aplikacjach często wystarczy tylko jedna podklasa, bardziej złożone aplikacje mogą wymagać zaimplementowania bardzo wielu różnych podklas. GeoQuiz jest prostą aplikacją, zatem będzie mieć tylko jedną podklasę klasy `Activity`, której nadamy nazwę `QuizActivity`. Zadaniem podklasy `QuizActivity` będzie zarządzanie interfejsem użytkownika, który został przedstawiony na rysunku 1.1.
- Za pomocą **układu** definiujemy zestaw obiektów interfejsu użytkownika i ich położenie na ekranie aplikacji. Układy mają postać zbiorów definicji obiektów zapisanych w postaci kodu XML. Poszczególne definicje są wykorzystywane do tworzenia obiektów pojawiających się na ekranie, takich jak przyciski czy tekst. Częścią aplikacji GeoQuiz będzie plik układu o nazwie `activity_quiz.xml`. Kod XML zapisany w tym pliku zawiera definicje elementów interfejsu użytkownika przedstawionego na rysunku 1.1.

Relacje pomiędzy klasą `QuizActivity` a plikiem układu `activity_quiz.xml` zostały przedstawione na rysunku 1.2.



Rysunek 1.2. Klasa `QuizActivity` zarządza obiektami zdefiniowanymi w pliku `activity_quiz.xml`

Mając w pamięci wspomniane wyżej dwa podstawowe elementy składowe, spróbujemy teraz utworzyć naszą aplikację.

Tworzenie projektu aplikacji dla systemu Android

Pierwszym etapem tworzenia aplikacji dla systemu Android jest utworzenie jej **projektu**. Taki projekt zawiera wszystkie pliki składowe aplikacji dla systemu Android. Aby utworzyć nowy projekt, musisz najpierw uruchomić program Android Studio.

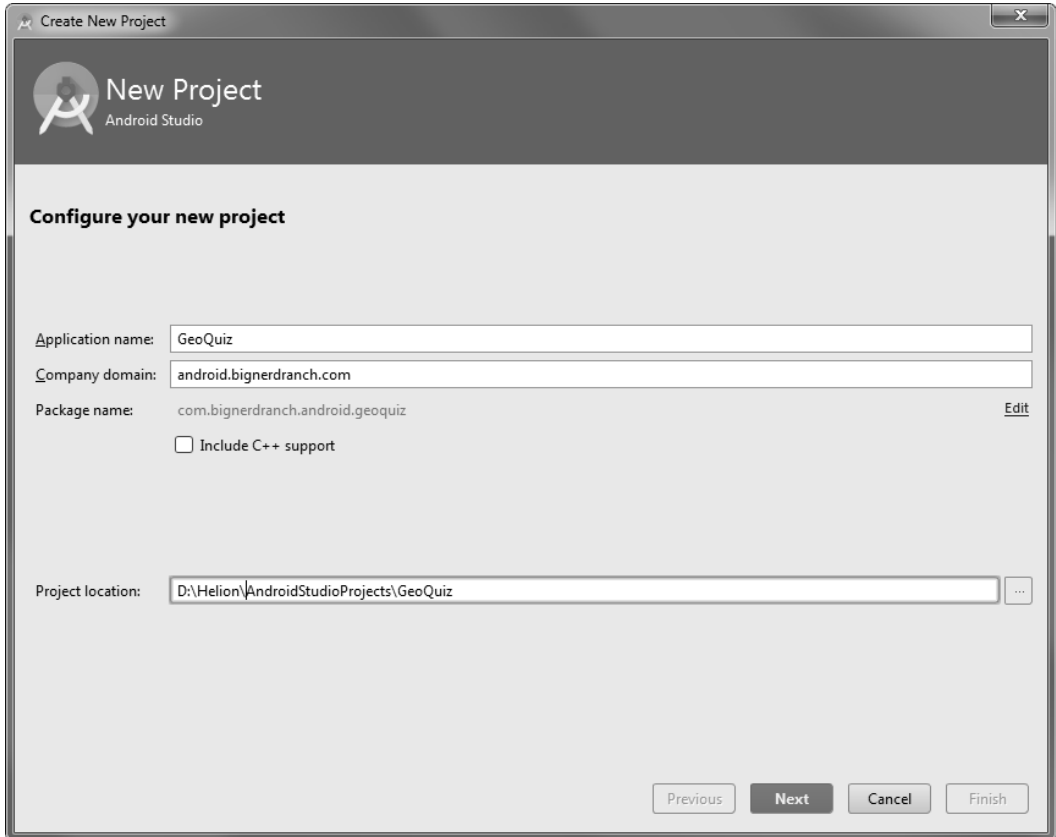
Jeżeli uruchamiasz program Android Studio po raz pierwszy, na ekranie powinieneś zobaczyć okno powitalne, przedstawione na rysunku 1.3.



Rysunek 1.3. Okno powitalne programu Android Studio

Kiedy wspomniane okno powitalne pojawi się na ekranie, wybierz opcję *Start a new Android Studio project* (utwórz nowy projekt Android Studio). Jeżeli po uruchomieniu Android Studio okno powitalne nie pojawia się, możesz utworzyć nowy projekt, wybierając z menu głównego opcję *File/New/New Project...* (plik/nowy/nowy projekt).

Na ekranie pojawi się okno dialogowe kreatora nowego projektu (zobacz rysunek 1.4). W pierwszym oknie kreatora jako nazwę aplikacji w polu *Application name* wpisz **GeoQuiz**. Jako nazwę domeny (pole *Company domain*) wpisz **android.bignerdranch.com**. Gdy to zrobisz, przekonasz się, że automatycznie generowana nazwa pakietu zmieniła się na **com.bignerdranch.android.geoquiz**. W polu *Project location* wpisz ścieżkę do dowolnie wybranego foldera, w którym przechowywany będzie tworzony projekt.



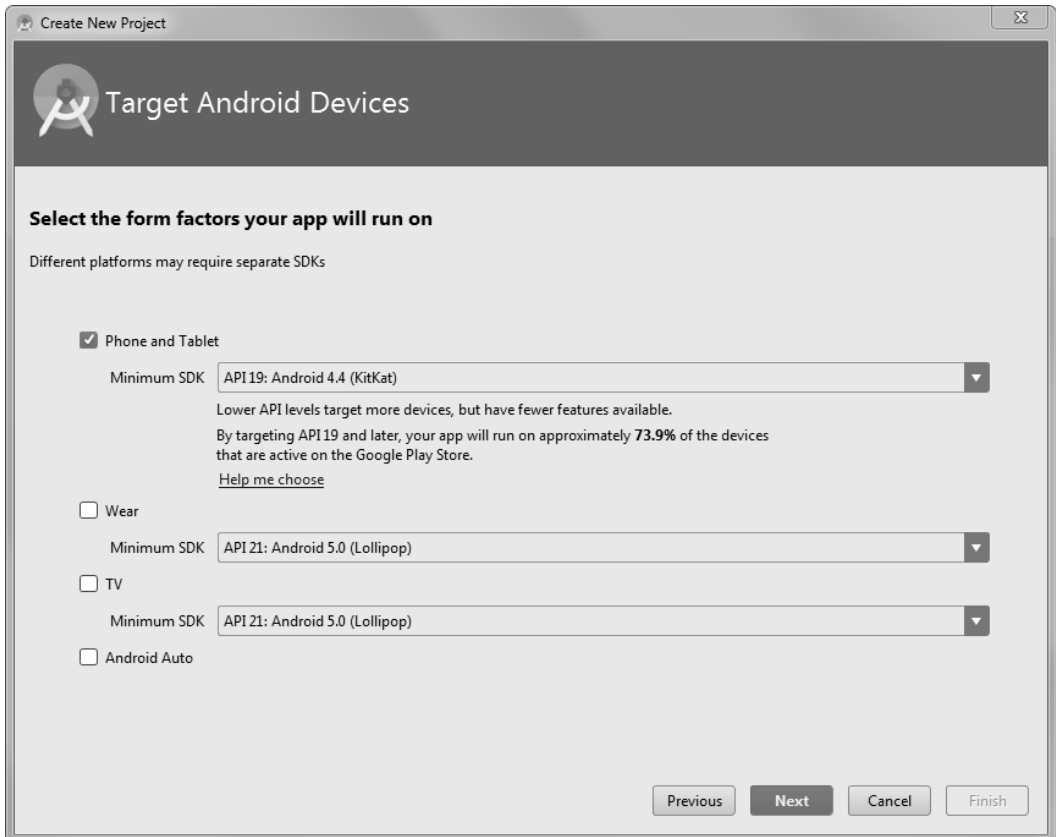
Rysunek 1.4. Tworzenie nowej aplikacji

Zwróć uwagę na fakt, że nazwa pakietu jest automatycznie generowana w „odwrotnej notacji DNS”: nazwa domeny Twojej organizacji jest zapisywana od końca i uzupełniana od prawej dodatkowymi identyfikatorami. Taka konwencja zapisu powoduje, że nazwy pakietów są unikatowe, i pozwala na łatwe rozróżnianie poszczególnych aplikacji zarówno w urządzeniu, jak i w usłudze Google Play.

Naciśnij przycisk *Next* (dalej). Opcje na kolejnym ekranie kreatora pozwalają na wybranie platform, na jakich będzie działała Twoja aplikacja. Aplikacja GeoQuiz będzie obsługiwana jedynie na telefonach, dlatego powinieneś zaznaczyć tylko opcję *Phone and Tablet* (telefony i tablety). Jako minimalną obsługiwaną wersję pakietu SDK wybierz *API 19: Android 4.4 (KitKat)* (zobacz rysunek 1.5). Więcej szczegółowych informacji na temat różnych wersji systemu Android znajdziesz w rozdziale 6.

Naciśnij przycisk *Next*.

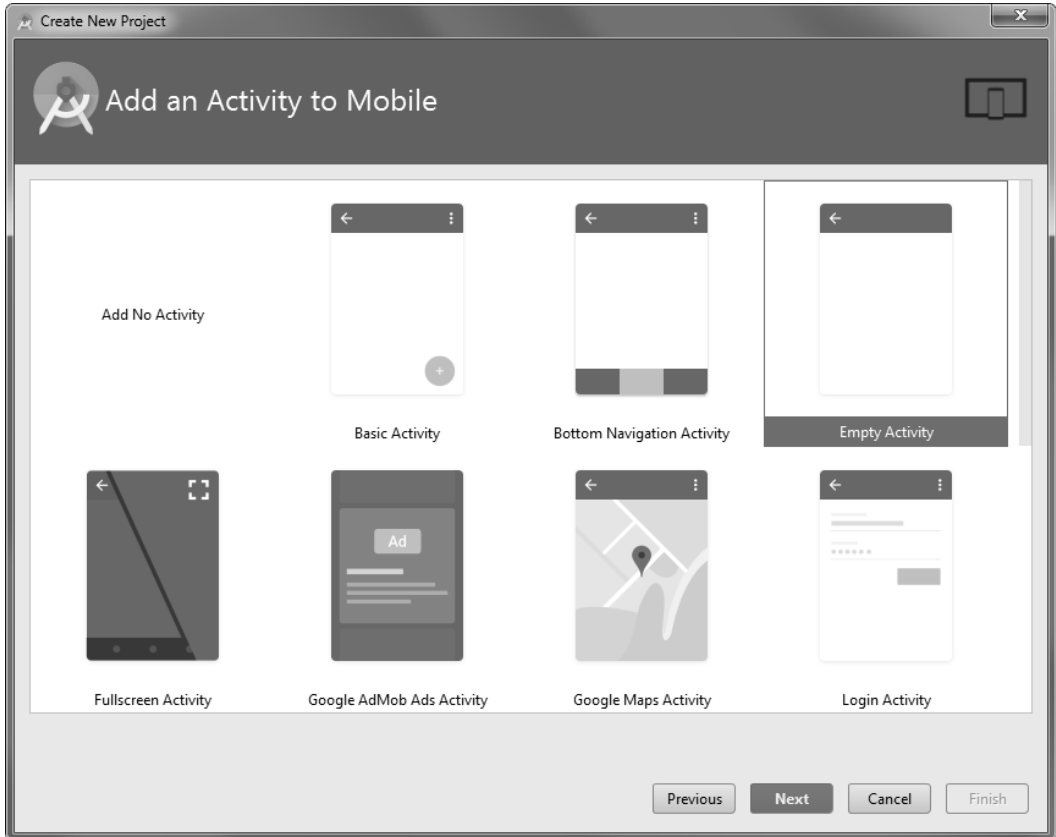
Na kolejnym ekranie kreatora będziesz miał możliwość wybrania szablonu dla pierwszego ekranu aplikacji GeoQuiz (zobacz rysunek 1.6). Wybierz najprostszy, pusty szablon o nazwie *Empty Activity* (pusta aktywność) i naciśnij przycisk *Next*.



Rysunek 1.5. Wybieranie obsługiwanych urządzeń

Pamiętaj, że pakiet Android Studio jest bardzo często aktualizowany, więc wygląd poszczególnych ekranów kreatora i innych elementów interfejsu może się nieco różnić od tych przedstawionych na rysunkach w książce. Zazwyczaj jednak nie powinno to stwarzać żadnych problemów, ponieważ dostępne opcje powinny być podobne. Jeżeli jednak w Twoim systemie okna lub inne narzędzia zdecydowanie różnią się od tych przedstawionych w książce, oznacza to, że w międzyczasie pojawiły się jakieś poważne aktualizacje. W takiej sytuacji nie powinieneś jednak wpadać w panikę — zamiast tego opisz swój problem na forum naszej książki pod adresem <https://forums.bignerdranch.com/>, a my postaramy się Ci pomóc w jego rozwiązaniu.

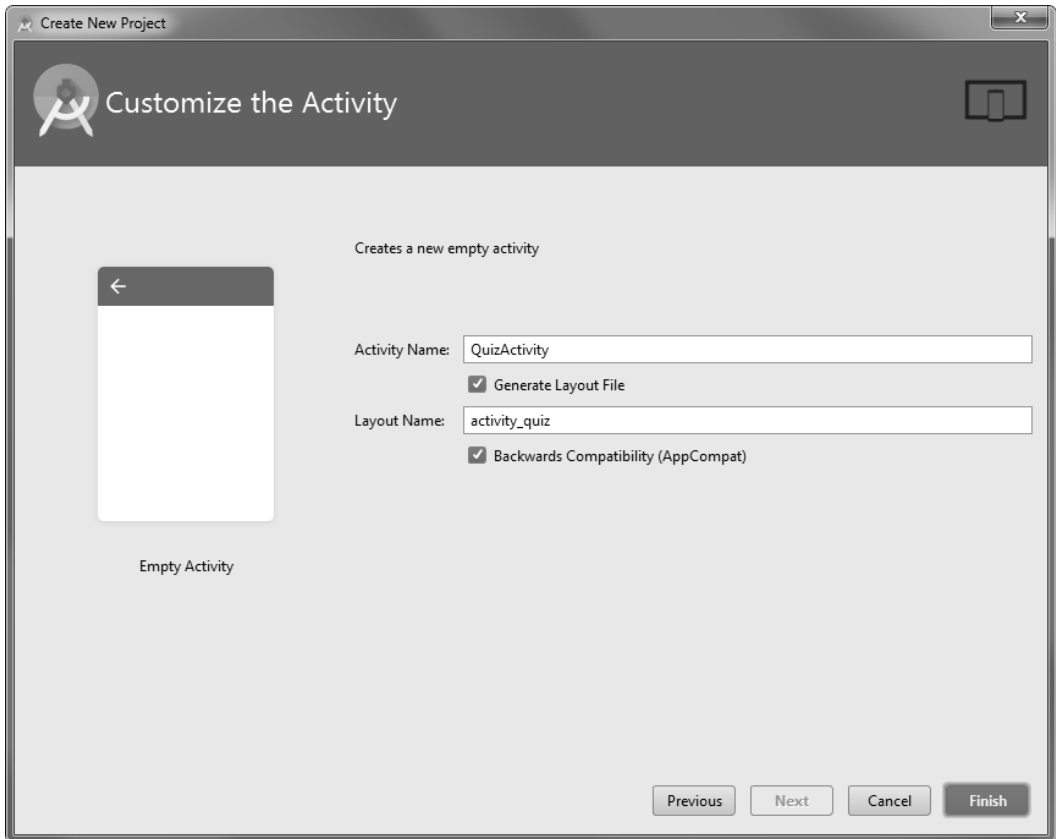
W ostatnim oknie kreatora jako nazwę podklasy aktywności w polu *Activity Name* wpisz **QuizActivity** (zobacz rysunek 1.7). Zwróć uwagę na przyrostek *Activity* w nazwie klasy. Stosowanie takich przyrostków nie jest oczywiście w żaden sposób obowiązkowe, niemniej jednak zachowanie takiej konwencji jest zdecydowanie polecaną praktyką.



Rysunek 1.6. Wybieranie typu aktywności

Pozostaw zaznaczoną opcję *Generate Layout File* (generuj plik układu). Nazwa pliku układu zostanie automatycznie zamieniona na *activity_quiz*, tak aby odzwierciedlała nową nazwę aktywności. Nazwa układu jest odwrotnością nazwy aktywności, jest zapisana małymi literami i poszczególne słowa składowe są oddzielone od siebie znakami podkreślenia zamiast spacji. Taka konwencja nazewnictwa jest zalecana zarówno dla wszystkich układów, jak i plików zasobów, o których będziemy pisać nieco później.

Jeżeli w Twojej wersji Android Studio na tym ekranie kreatora znajdują się jeszcze jakieś inne opcje, pozostaw je na wartościach domyślnych. Naciśnij przycisk *Finish* (zakończ). Android Studio wygeneruje nowy projekt i otworzy go w edytorze.



Rysunek 1.7. Konfigurowanie nowej aktywności

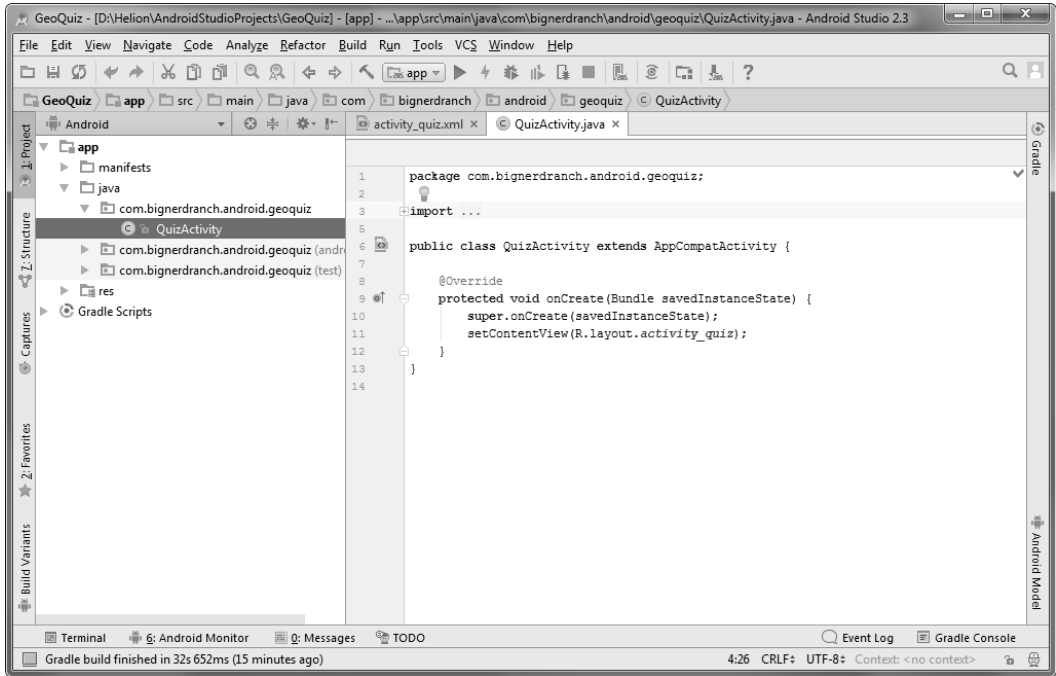
Poruszanie się w Android Studio

Android Studio otwiera projekt w oknie programu, tak jak to zostało pokazane na rysunku 1.8.

Okno programu Android Studio składa się z wielu paneli nazywanych **oknami narzędziowymi** (ang. *tool windows*).

Panel po lewej stronie okna nazywany jest **oknem projektu**. Za jego pomocą możesz przeglądać wszystkie pliki powiązane z projektem i zarządzać nimi.

Głównym panelem okna Android Studio jest **edytor**, w którym na początek otwarty został plik *QuizActivity.java*.



Rysunek 1.8. Okno nowo utworzonego projektu

W razie potrzeby możesz przełączać się między poszczególnymi oknami, klikając ich nazwy znajdujące się na paskach narzędziowych mieszczących się z lewej i z prawej strony oraz na dole ekranu. Dla wielu okien dostępne są również odpowiednie skróty klawiszowe. Jeżeli paski z nazwami okien nie są widoczne, kliknij szary, kwadratowy przycisk znajdujący się w lewym, dolnym rogu okna lub po prostu wybierz z menu głównego polecenie *View/Tool Buttons* (widok/przyciski narzędzi).

Tworzenie układu interfejsu użytkownika

Otwórz w edytorze plik *app/res/layout/activity_quiz.xml*. Jeżeli zamiast kodu XML widzisz graficzny podgląd układu, kliknij kartę *Text* (tekst) znajdującą się w dolnej części okna edytora.

Obecnie plik układu *activity_quiz.xml* zawiera definicję domyślnego układu aktywności. W kolejnych wersjach Android Studio definicje domyślne zmieniają się dosyć często, ale mimo to kod XML powinien wyglądać mniej więcej tak, jak to zostało przedstawione na listingu 1.1.

Listing 1.1. Domyślny układ aktywności (plik *activity_quiz.xml*)

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_quiz"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

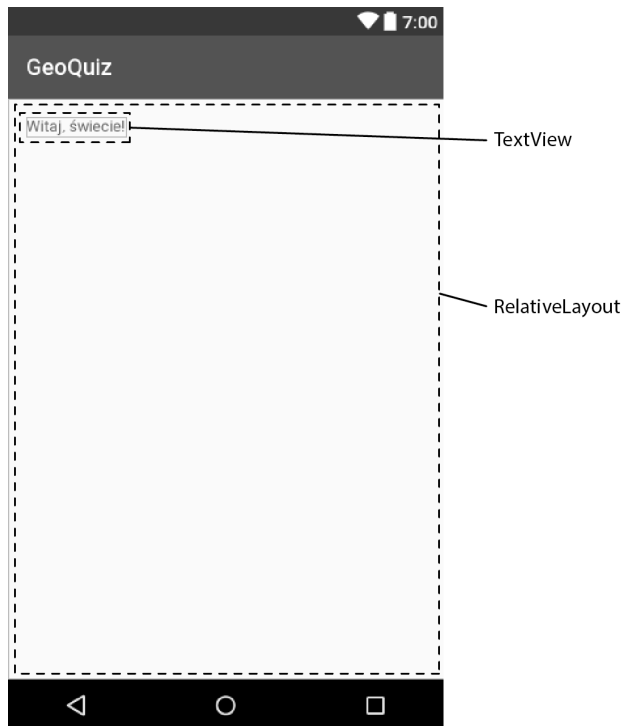
```
android:paddingBottom="16dp"  
android:paddingLeft="16dp"  
android:paddingRight="16dp"  
android:paddingTop="16dp"  
tools:context="com.bignerdranch.android.geoquiz.QuizActivity">  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Witaj, świecie!"/>  
</RelativeLayout>
```

W domyślnym układzie aktywności znajdują się definicje dwóch widgetów: `RelativeLayout` oraz `TextView`.

Widgety to elementy składowe, z których budowane są interfejsy użytkownika. Widgety mogą wyświetlać tekst lub grafikę, komunikować się z użytkownikiem czy rozmieszczać inne widgety na ekranie. Przykładami widgetów są przyciski, kontrolki pozwalające na wprowadzanie tekstu czy pola opcji.

W środowisku Android SDK znajdziesz wiele różnych widgetów, które możesz konfigurować tak, aby dostosować ich wygląd i zachowanie do własnych potrzeb. Każdy widget jest instancją klasy `View` lub jednej z jej podklas (takich jak `TextView` czy `Button`).

Na rysunku 1.9 pokazano, jak widgety `RelativeLayout` oraz `TextView`, zdefiniowane na listingu 1.1, będą wyglądać na ekranie.

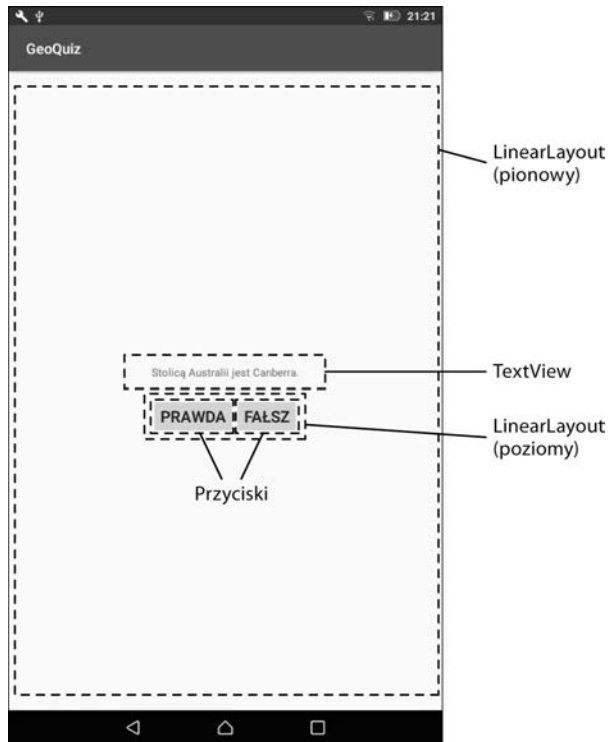


Rysunek 1.9. Wygląd domyślnych widgetów na ekranie

Nie są to jednak widżety, których szukamy. Dla interfejsu aktywności `QuizActivity` będzie nam potrzebnych pięć widżetów:

- pionowy `LinearLayout`,
- `TextView`,
- poziomy `LinearLayout`,
- dwa przyciski `Button`.

Na rysunku 1.10 pokazano, w jaki sposób wymienione widżety będą tworzyć interfejs aktywności `QuizActivity`.



Rysunek 1.10. Planowany wygląd widżetów na ekranie

Teraz musimy zdefiniować nasze widżety w pliku układu `activity_quiz.xml`.

W oknie projektu aplikacji odszukaj katalog `app/res/layout`, wyświetl jego zawartość i otwórz plik `activity_quiz.xml`. Wprowadź zmiany przedstawione na listingu 1.2. Kod XML, który musisz usunąć, został na listingu oznaczony przekreśloną czcionką, a kod XML, który powinieneś dodać, został na tym listingu wyróżniony pogrubioną czcionką. Takiej konwencji zapisu kodu będziemy używać w całej książce.

Listing 1.2. Definiowanie widgetów w kodzie XML (plik activity_quiz.xml)

```

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_quiz"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.bignerdranch.android.geoquiz.QuizActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Witaj, świecie!"/>
</RelativeLayout>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button" />
    </LinearLayout>
</LinearLayout>

```

Nie przejmuj się, jeżeli wpisywany kod będzie dla Ciebie niezrozumiały — już za chwilę dowiesz się, jak to działa. Przepisując poszczególne wiersze, powinieneś jednak zachować ostrożność. Poprawność kodu XML w plikach układu nie jest w żaden sposób weryfikowana, stąd każda literówka wcześniej czy później może być przyczyną potencjalnych kłopotów.

Po wpisaniu kodu Android Studio wyświetli błędy w trzech wierszach rozpoczynających się od polecenia `android:text` — możesz je spokojnie zignorować, naprawimy je już za chwilę.

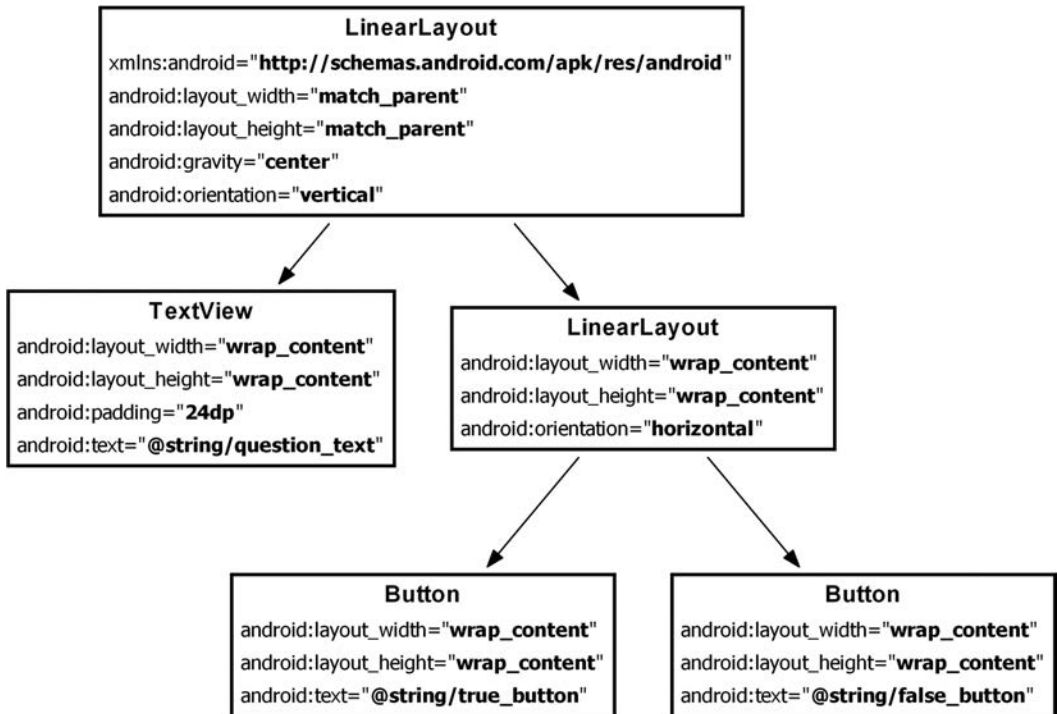
Porównaj kod XML z wyglądem interfejsu użytkownika przedstawionego na rysunku 1.10. Każdemu widgetowi przypisany jest odpowiedni element kodu XML, którego nazwa determinuje typ definiowanego widgetu.

Każdy element posiada szereg przypisanych *atrybutów* XML. Każdy *atrybut* jest swego rodzaju instrukcją określającą sposób skonfigurowania określonej właściwości widgetu.

Aby lepiej zrozumieć, jak działają elementy i ich atrybuty, spróbujemy spojrzeć na układ z perspektywy hierarchicznej.

Hierarchia widoków

Twoje widgety funkcjonują w pewnej hierarchii obiektów `View`, nazywanej **hierarchią widoków**. Na rysunku 1.11 pokazano hierarchię widoków odpowiadającą kodowi XML z listingu 1.2.



Rysunek 1.11. Hierarchiczny układ widжетów i atrybutów

Głównym elementem tej hierarchii widoku jest układ `LinearLayout`, dlatego w jego kodzie musi znajdować się definicja przestrzeni nazw zasobów XML (<http://schemas.android.com/apk/res/android>).

`LinearLayout` dziedziczy swoje właściwości z klasy `ViewGroup`, będącej podklasą klasy `View`. Obiekt `ViewGroup` to widget spełniający rolę kontenera dla innych widgetów. Układu `LinearLayout` używamy, kiedy chcemy ułożyć widgety w pojedynczym wierszu lub jednej kolumnie. Innymi podklasami kontenera `ViewGroup` są `FrameLayout`, `TableLayout` i `RelativeLayout`.

Kiedy dany widget znajduje się w kontenerze `ViewGroup`, mówimy, że jest obiektem podrzędnym tej klasy. W naszym przypadku główny układ `LinearLayout` posiada dwa obiekty podrzędne: kontrolkę `TextView` oraz kolejny układ `LinearLayout`. Z kolei w układzie `LinearLayout` znajdują się dwa przyciski będące jego obiektami podrzędnymi.

Atrybuty widgetów

Omówimy teraz wybrane atrybuty, których używaliśmy do skonfigurowania naszych widgetów.

Atrybuty `android:layout_width` oraz `android:layout_height`

Atrybuty `android:layout_width` i `android:layout_height` są wymagane niemal dla wszystkich widgetów. Zazwyczaj atrybuty te są ustawiane na wartość `match_parent` lub `wrap_content`:

- wartość `match_parent` oznacza, że widok będzie miał takie rozmiary jak jego obiekt nadrzędny;
- wartość `wrap_content` oznacza, że widok będzie miał rozmiary dopasowane do rozmiarów jego zawartości.

(Od czasu do czasu możesz się również spotkać z wartością `fill_parent`; ta przestarzała wartość jest odpowiednikiem wartości `match_parent`).

W przypadku głównego układu `LinearLayout` wartością atrybutów określających jego szerokość i wysokość jest `match_parent`. Układ `LinearLayout` jest elementem głównym, ale mimo to posiadającym swój obiekt nadrzędny — jest nim widok, jaki system Android udostępnia do wyświetlenia hierarchii widoków naszej aplikacji.

Pozostałe widgety w naszej aplikacji posiadają atrybuty szerokości i wysokości ustawione na wartość `wrap_content`. Na rysunku 1.10 możesz zobaczyć, jaki ma to wpływ na ich rozmiary.

Rozmiary widgetu `TextView` są nieco większe niż rozmiary umieszczonego w nim tekstu, ponieważ widget ten posiada atrybut `android:padding="24dp"`, który powoduje, że efektywne rozmiary widgetu są większe o podaną wartość od rozmiarów swojej zawartości. Tego atrybutu będziemy używać do utworzenia pewnego odstępu pomiędzy pytaniem quizowym a przyciskami odpowiedzi. Zastanawiasz się, co oznacza `dp`? To tak zwana jednostka miary niezależna od gęstości pikseli na ekranie urządzenia (ang. *density-independent pixels*).

Atrybut `android:orientation`

Atrybut `android:orientation` naszych układów `LinearLayout` określa, czy ich obiekty podrzędne będą wyświetlane w pionie, czy w poziomie. Główny układ `LinearLayout` posiada orientację pionową, a jego podrzędny układ `LinearLayout` — orientację poziomą.

Kolejność definiowania poszczególnych obiektów podrzędnych determinuje jednocześnie kolejność, w jakiej będą się one pojawiały na ekranie. W pionowym układzie `LinearLayout` pierwszy zdefiniowany obiekt podrzędny zostanie wyświetlony jako pierwszy od góry. W poziomym układzie `LinearLayout` pierwszy zdefiniowany obiekt będzie wyświetlany jako pierwszy od lewej (o ile oczywiście dane urządzenie

nie jest skonfigurowane do używania wersji językowej, w której słowa zapisujemy w kierunku od prawej do lewej strony, jak na przykład w językach arabskim czy hebrajskim — w takiej sytuacji pierwszy zdefiniowany obiekt podrzędny będzie wyświetlany jako pierwszy od prawej strony).

Atrybut `android:text`

Widgety `TextView` i `Button` posiadają atrybut `android:text`, który odpowiada za wyświetlany tekst.

Zwróć uwagę, że wartościami tego atrybutu nie są literały — zamiast tego używamy tutaj odwołań do odpowiednich ciągów znaków w plikach zasobów tekstowych aplikacji.

Ciąg zasobów (ang. *string resource*) to ciąg znaków przechowywanych w osobnym pliku XML **zasobów tekstowych** (ang. *strings file*). Teoretycznie można również przypisywać do widgetów zakodowane na „sztywno” ciągi znaków (na przykład `android:text="Prawda"`), ale zazwyczaj nie jest to najlepsze rozwiązanie. Umieszczanie wszystkich ciągów znaków w osobnym pliku zasobów i następnie odwoływanie się do odpowiednich ciągów w kodzie programu znakomicie ułatwia tłumaczenie aplikacji na inne języki.

Plik zasobów tekstowych, do którego odwołujemy się w pliku `activity_quiz.xml`, jeszcze nie istnieje. Musimy zatem to naprawić.

Tworzenie zasobów tekstowych

Każdy projekt posiada domyślny plik zasobów tekstowych o nazwie `strings.xml`.

Otwórz plik `res/values/strings.xml`. W szablonie pliku domyślnie zdefiniowany jest jeden ciąg znaków. Dodamy teraz do tego pliku trzy nowe ciągi znaków, które będą wykorzystywane przez naszą aplikację.

Listing 1.3. Dodawanie zasobów tekstowych (plik `strings.xml`)

```
<resources>
  <string name="app_name">GeoQuiz</string>
  <string name="question_text">Canberra jest stolicą Australii.</string>
  <string name="true_button">Prawda</string>
  <string name="false_button">Fałsz</string>
</resources>
```

(W zależności od tego, której wersji Android Studio używasz, w pliku zasobów tekstowych mogą znajdować się również inne, domyślne ciągi znaków. Nie kasuj ich, ponieważ ich usunięcie może spowodować pojawienie się szeregu błędów w innych plikach projektu).

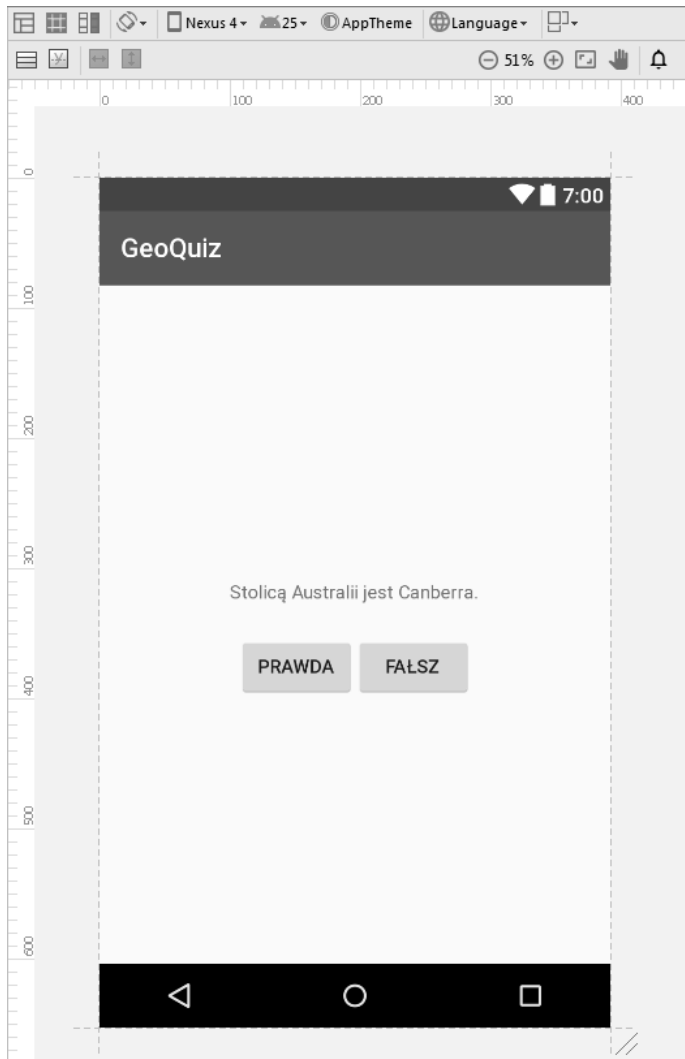
Od tej chwili za każdym razem, kiedy będziesz używał odwołań do ciągu `@string/false_button` w dowolnym pliku XML projektu `GeoQuiz`, po uruchomieniu aplikacji zostanie ono zastąpione ciągiem znaków `Fałsz`.

Komunikaty błędów o braku odpowiednich zasobów tekstowych, które wcześniej pojawiały się w pliku `activity_quiz.xml`, powinny teraz zniknąć (jeżeli nadal są tam jakieś błędy, sprawdź w obu plikach, czy nie zrobiłeś jakichś literówek).

Choć domyślny plik zasobów tekstowych nosi nazwę `strings.xml`, możesz ją zmienić na niemal dowolną inną. Co więcej, w danym projekcie możesz mieć wiele plików zasobów tekstowych. Jeżeli każdy z tych plików znajduje się w katalogu `res/values` i każdy posiada główny element o nazwie `resources` oraz elementy podrzędne `string`, to zdefiniowane w nich zasoby tekstowe będą poprawnie interpretowane i dostępne dla aplikacji.

Podgląd układu

Układ naszej aplikacji jest już kompletny, zatem możesz wyświetlić jego podgląd i sprawdzić, czy wszystko jest tak, jak być powinno (zobacz rysunek 1.12). Zanim to jednak zrobisz, warto się upewnić, czy w edytowanych do tej pory plikach nie ma błędów. Jeżeli wszystko jest w porządku, przejdź do okna pliku *activity_quiz.xml* i otwórz okno podglądu, naciskając przycisk *Preview* (podgląd) znajdujący się na pionowym pasku narzędzi po prawej stronie okna edytora.



Rysunek 1.12. Wyświetlanie pliku *activity_quiz.xml* w oknie podglądu graficznego

Od pliku układu XML do obiektów klasy View

W jaki sposób elementy układu zdefiniowane w plikach XML stają się obiektami klasy View? Poszukiwania odpowiedzi możemy rozpocząć w definicji klasy QuizActivity.

Kiedy tworzyłeś projekt aplikacji GeoQuiz, utworzona została klasa QuizActivity, będąca podklasą klasy Activity. Plik klasy QuizActivity znajduje się w katalogu `app/java` Twojego projektu. Katalog `java` jest miejscem, w którym przechowywany jest kod Java aplikacji.

W oknie projektu odszukaj katalog `app/java/ com.bignerdranch.android.geoquiz`, a następnie znajdź w nim plik o nazwie `QuizActivity.java`, otwórz go i przyjrzyj się jego zawartości (zobacz listing 1.4).

Listing 1.4. Domyślny plik klasy QuizActivity (plik QuizActivity.java)

```
package com.bignerdranch.android.geoquiz;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class QuizActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
}
```

(Zastanawiasz się, czym jest AppCompatActivity? To podklasa klasy Activity, która zapewnia kompatybilność ze starszymi wersjami systemu Android. Więcej szczegółowych informacji na temat klasy AppCompatActivity znajdziesz w rozdziale 13.).

Jeżeli wiersze definicji importowanych klas nie są widoczne, kliknij symbol + znajdujący się po lewej stronie wyrażenia `import`, co spowoduje rozwinięcie i wyświetlenie całej grupy wyrażeń.

Klasa QuizActivity posiada na razie tylko jedną metodę: `onCreate(Bundle)`.

(Jeżeli w Twoim pliku klasy QuizActivity znajdują się również definicje metod `onOptionsItemSelected` (Menu) oraz `onOptionsItemSelected` (MenuItem), powinieneś je po prostu zignorować. Menu aplikacji będziemy szczegółowo omawiać w rozdziale 13.).

Metoda `onCreate(Bundle)` jest wywoływana w sytuacji, kiedy tworzona jest instancja podklasy aktywności. Kiedy tworzona jest aktywność, do jej działania niezbędny jest interfejs użytkownika. Aby udostępnić aktywności taki interfejs, musimy wywołać następującą metodę:

```
public void setContentView(int layoutResID)
```

Metoda ta *rozwija* układ i umieszcza go na ekranie. Po rozwinięciu układu tworzone są poszczególne widgety zgodnie z definicjami określonymi przez ich atrybuty. Wyboru układu do rozwinięcia dokonujemy poprzez przekazanie do metody odpowiedniego identyfikatora układu (`layoutResID`).

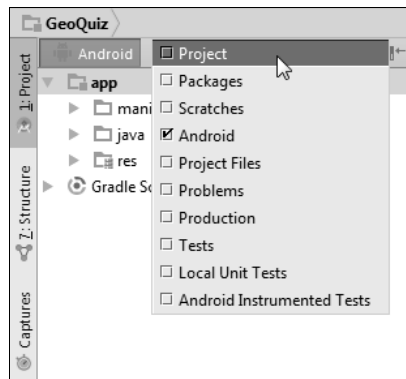
Zasoby i identyfikatory zasobów

Każdy układ jest *zasobem*. **Zasoby** są elementami aplikacji, które nie są kodem — zasobami aplikacji są na przykład pliki graficzne, pliki audio czy pliki XML.

Zasoby Twojego projektu przechowywane są w katalogu *app/res*. W oknie projektu możesz na przykład zobaczyć, że plik *activity_quiz.xml* zlokalizowany jest w katalogu *res/layout*. Plik zasobów tekstowych, w którym przechowywane są ciągi znaków wykorzystywane w aplikacji, znajduje się w katalogu *res/values*.

Aby skorzystać z określonego zasobu, musisz użyć odpowiedniego **identyfikatora zasobu** (ang. *resource ID*). Na przykład identyfikator naszego układu w aplikacji wygląda tak: `R.layout.activity_quiz`.

Aby wyświetlić bieżące identyfikatory zasobów aplikacji GeoQuiz, musisz najpierw zmienić sposób wyświetlania projektu. Domyślnie Android Studio wykorzystuje widok o nazwie *Android* (zobacz rysunek 1.13). Widok ten ukrywa rzeczywistą strukturę katalogów projektu, dzięki czemu możesz bardziej skoncentrować się na plikach i katalogach, które są Ci potrzebne najczęściej.



Rysunek 1.13. Zmiana sposobu wyświetlania projektu

Odszukaj listę rozwijaną widoków, znajdującą się w górnej części okna projektu, i zmień widok *Android* na widok *Project*. W widoku *Project* wyświetlana jest rzeczywista struktura plików i katalogów Twojej aplikacji.

Aby wyświetlić pliki zasobów aplikacji GeoQuiz, przejdź do katalogu *app/build/generated/source/r/debug*. Teraz odszukaj katalog o nazwie odpowiadającej nazwie pakietu projektu i otwórz znajdujący się w nim plik *R.java*. Ponieważ plik ten jest automatycznie generowany podczas procesu budowania aplikacji, nie powinieneś go w żaden sposób modyfikować, o czym zresztą informuje komunikat zamieszczony w pierwszych kilku wierszach pliku.

Pamiętaj, że zmiany dokonywane w zasobach nie są natychmiast odzwierciedlane w tym pliku. Android Studio posiada ukryty plik *R.java*, na podstawie którego budowany jest kod aplikacji. Plik *R.java*, którego zawartość przedstawiono na listingu 1.5, został wygenerowany dla Twojej aplikacji, zanim została zainstalowana w urządzeniu lub w emulatorze. Zawartość tego pliku będzie zaktualizowana po uruchomieniu aplikacji.

Listing 1.5. Bieżące identyfikatory zasobów aplikacji GeoQuiz (plik R.java)

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */
package com.bignerdranch.android.geoquiz;
public final class R {
    public static final class anim {
        ...
    }
    ...
    public static final class id {
        ...
    }
    public static final class layout {
        ...
        public static final int activity_quiz=0x7f030017;
    }
    public static final class mipmap {
        public static final int ic_launcher=0x7f030000;
    }
    public static final class string {
        ...
        public static final int app_name=0x7f0a0010;
        public static final int false_button=0x7f0a0012;
        public static final int question_text=0x7f0a0014;
        public static final int true_button=0x7f0a0015;
    }
}
}
```

Plik *R.java* może mieć duże rozmiary, dlatego na listingu 1.5 większość jego zawartości została celowo pominięta.

Właśnie stąd pochodzi identyfikator `R.layout.activity_quiz` — jest to stała wartość całkowita o nazwie `activity_quiz`, znajdująca się w wewnętrznej klasie `R.layout`.

Ciągi znaków, których używasz w aplikacji, również posiadają swoje identyfikatory zasobów. W kodzie aplikacji jeszcze nie używałeś odwołań do takich ciągów znaków, ale jeżeli byś to zrobił, to takie odwołanie mogłoby mieć następującą postać:

```
setTitle(R.string.app_name);
```

System Android wygenerował identyfikatory zasobów dla całego układu i dla poszczególnych ciągów znaków, ale nie utworzył identyfikatorów dla poszczególnych widgetów zdefiniowanych w pliku *activity_quiz.xml*. Nie każdy widget potrzebuje przydzielonego identyfikatora zasobu. W tym rozdziale cała interakcja użytkownika jest ograniczona do pracy z dwoma przyciskami, więc tylko one będą potrzebowały identyfikatorów zasobów.

Przed wygenerowaniem identyfikatorów zasobów przełącz okno projektu z powrotem do widoku *Android*. We wszystkich przykładach omawianych w tej książce będziemy korzystali z widoku *Android*, choć jeżeli wolisz, to możesz spokojnie używać widoku *Project*.

Aby wygenerować identyfikator zasobu dla wybranego widgetu, powinieneś w kodzie definicji widgetu użyć atrybutu `android:id`. W pliku `activity_quiz.xml` dodamy teraz atrybuty `android:id` dla obu przycisków (zobacz listing 1.6).

Listing 1.6. Dodawanie identyfikatorów dla przycisków Button (plik `activity_quiz.xml`)

```
<LinearLayout ... >
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="@string/question_text" />
  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <Button
      android:id="@+id/true_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/true_button" />
    <Button
      android:id="@+id/false_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/false_button" />
  </LinearLayout>
</LinearLayout>
```

Zwróć uwagę, że przy wartościach atrybutów `android:id` znajduje się znak `+`, a przy wartościach atrybutów `android:text` takiego znaku nie ma. Dzieje się tak, ponieważ w naszym kodzie *tworzymy* identyfikatory zasobu dla przycisków i *odwołujemy się* do ciągów znaków.

Podłączanie widgetów

Kiedy oba przyciski posiadają już swoje identyfikatory, możesz się do nich odwoływać z poziomu kodu aplikacji `QuizActivity`. Na początek musimy dla przycisków utworzyć odpowiednie **zmienne składowe** (ang. *member variables*).

Dodaj kod zamieszczony na listingu 1.7 do pliku `QuizActivity.java` (nie korzystaj z opcji dopełniania słów kluczowych; przepisz całość ręcznie). Po zapisaniu pliku na ekranie powinny się pojawić dwa komunikaty o wystąpieniu błędów.

Listing 1.7. Dodawanie parametrów klasy (plik QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    private Button mTrueButton;
    private Button mFalseButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
}
```

Wyświetlone błędy naprawimy już za chwilę. Zwróć uwagę na prefiks `m` w nazwach obu zmiennych składowych. Zastosowanie tego prefiksu wynika z przyjętej w systemie Android konwencji nazywania zmiennych, z której będziemy konsekwentnie korzystać w przykładach omawianych w tej książce.

Ustaw teraz wskaźnik myszy nad wyróżnionymi na czerwono fragmentami kodu, wskazującymi miejsca wystąpienia błędów. W obu przypadkach komunikat o błędzie jest taki sam: `Cannot resolve symbol 'Button'` (nie można rozwiązać symbolu `'Button'`).

Takie komunikaty o błędzie sugerują, że do kodu naszej aplikacji powinienś zaimportować klasę `android.widget.Button`. Aby to zrobić, powinienś w sekcji `import` kodu klasy umieścić następujące polecenie:

```
import android.widget.Button;
```

Zamiast tego możesz jednak pójść na skróty i pozwolić, aby Android Studio wykonało całą pracę za Ciebie. Aby to zrobić, ustaw kursor na podświetlonym fragmencie kodu i naciśnij kombinację klawiszy *lewy Alt+Enter*, a wspaniały mechanizm o nazwie Intellij dokona odpowiednich uzupełnień i modyfikacji w kodzie. Nowe polecenie importu automatycznie pojawi się w sekcji `import` na początku pliku. Opisana kombinacja klawiszy wywołująca mechanizm Intellij jest bardzo użyteczna w wielu sytuacjach, kiedy w kodzie aplikacji pojawiają się jakieś problemy. Staraj się korzystać z niej jak najczęściej!

Wykonanie opisanych powyżej czynności powinno załatwić problem z błędami (jeżeli jednak jakieś błędy nadal pojawiają się w kodzie, sprawdź, czy nie popełniłeś literówek w kodzie Java oraz XML).

Jeżeli wszystko jest w porządku, możemy przystąpić do podłączania widgetów reprezentujących nasze przyciski. Cały proces jest dwuetapowy:

- Pobierz odwołania do rozwiniętych obiektów widoku.
- Ustaw dla tych obiektów listenery (obiekty nasłuchujące), które będą reagowały na akcje podejmowane przez użytkownika.

Pobieranie odwołań do widgetów

Aby w danej aktywności pobrać odwołanie do określonego widgetu w rozwiniętym widoku, powinienś wywołać następującą metodę:

```
public View findViewById(int id)
```

Parametrem wywołania metody jest identyfikator zasobu, a wynikiem działania odpowiadający mu obiekt widoku.

W kodzie *QuizActivity.java* będziemy używać identyfikatorów zasobu reprezentujących przyciski do pobrania rozwiniętych obiektów i przypisania ich do zmiennych składowych. Zwróć uwagę, że przed dokonaniem przypisania musisz dokonać rzutowania zwracanego obiektu `View` na typ `Button` (zobacz listing 1.8).

Listing 1.8. Pobieranie odwołań do widgetów (plik *QuizActivity.java*)

```
public class QuizActivity extends AppCompatActivity {
    private Button mTrueButton;
    private Button mFalseButton;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mTrueButton = (Button) findViewById(R.id.true_button);
        mFalseButton = (Button) findViewById(R.id.false_button);
    }
}
```

Tworzenie obiektów nasłuchujących

Aplikacje dla systemu Android są zazwyczaj *sterowane zdarzeniami*. W przeciwieństwie do programów działających z poziomu konsoli czy skryptów aplikacje sterowane zdarzeniami po uruchomieniu oczekują na zdarzenia, takie jak na przykład naciśnięcie przycisku przez użytkownika (zdarzenia mogą być również inicjowane przez system operacyjny bądź inne aplikacje, niemniej najbardziej oczywiste są zdarzenia inicjowane przez użytkownika).

Kiedy aplikacja oczekuje na pojawienie się określonego zdarzenia, mówimy, że taka aplikacja „nasłuchuje” zdarzeń. Obiekty, których zadaniem jest reagowanie na dane zdarzenie, nazywamy **obiektami nasłuchującymi** (ang. *listener*). Obiekt nasłuchujący implementuje dla danego zdarzenia odpowiedni interfejs (ang. *listener interface*).

Środowisko Android SDK posiada odpowiednie interfejsy obiektów nasłuchujących do obsługi różnych zdarzeń, dzięki czemu nie musisz tworzyć własnych listenerów. W tym przypadku zdarzeniem, którego będziemy nasłuchiwać, jest naciśnięcie (bądź jak kto woli „kliknięcie”) przycisku, zatem nasz listener będzie implementował interfejs `View.OnClickListener`.

Rozpocniemy od przycisku *PRAWDA*. W pliku *QuizActivity.java* do procedury `onCreate(Bundle)` dopisz kod przedstawiony poniżej i umieść go zaraz za deklaracją zmiennych (zobacz listing 1.9).

Listing 1.9. Tworzenie listenera dla przycisku *PRAWDA* (plik *QuizActivity.java*)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_quiz);

    mTrueButton = (Button) findViewById(R.id.true_button);
    mTrueButton.setOnClickListener(new View.OnClickListener() {
        @Override
```

```
        public void onClick(View v) {
            // Nic tu się nie dzieje, ale wkrótce to zmienimy!
        }
    });
    mFalseButton = (Button) findViewById(R.id.false_button);
}
}
```

Jeżeli po dodaniu kodu pojawia się błąd *View cannot be resolved to a type*, spróbuj zaimportować klasę `View`, naciskając kombinację klawiszy *lewy Alt+Enter*.

W kodzie przedstawionym na listingu 1.9 ustawiamy obiekt nasłuchujący, którego zadaniem będzie informowanie o tym, że przycisk `Button` o nazwie `mTrueButton` został naciśnięty. Argumentem wywołania metody `setOnClickListener(OnClickListener)` jest obiekt nasłuchujący. W naszym przypadku argumentem wywołania tej metody jest obiekt implementujący listenera `OnClickListener`.

Zastosowanie anonimowych klas wewnętrznych

Nasz listener został zaimplementowany jako **anonimowa klasa wewnętrzna** (ang. *anonymous inner class*). Taka nazwa jest nieco złożona, ale pomaga zapamiętać, że wszystko, co znajduje się w obrębie zewnętrznych nawiasów okrągłych, jest przekazywane do `setOnClickListener(OnClickListener)`. Wewnątrz nawiasów tworzymy nową, nieposiadającą swojej nazwy klasę i przekazujemy jej całą implementację do tej metody.

```
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Nic tu się nie dzieje, ale wkrótce to zmienimy!
    }
})
```

Wszystkie listenery omawiane w tej książce zostały zaimplementowane jako anonimowe klasy wewnętrzne. Takie rozwiązanie przenosi implementację metod obiektów nasłuchujących dokładnie tam, gdzie chcemy je widzieć. Co więcej, nie ma tutaj potrzeby stosowania pełnowymiarowej klasy nazwanej, ponieważ taka klasa będzie wykorzystywana tylko w jednym miejscu.

Ponieważ nasza anonimowa klasa wewnętrzna implementuje obiekt nasłuchujący `OnClickListener`, musi również implementować jedyną metodę tego interfejsu, `onClick(View)`. Na razie w ciele tej metody nie umieściliśmy jeszcze żadnego kodu, ale z punktu widzenia kompilatora nie stanowi to żadnego problemu. Interfejs listenera wymaga co prawda zaimplementowania metody `onClick(View)`, ale nie nakłada żadnych wymagań co do tego, *jak* powinieneś to zrobić.

(Jeżeli Twoja znajomość zagadnień związanych z anonimowymi klasami wewnętrznymi, listenerami czy ich interfejsami nie jest najlepsza, powinieneś sobie nieco odświeżyć wybrane tematy z dziedziny programowania w języku Java, a przynajmniej powinieneś mieć pod ręką jakąś dobrą książkę na ten temat).

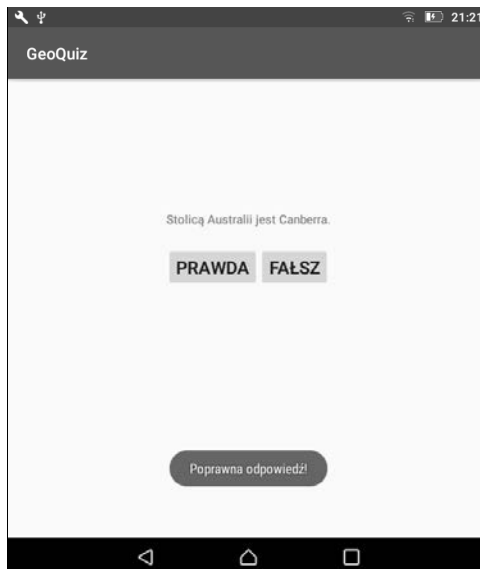
Podobny do omawianego wyżej obiekt nasłuchujący utworzymy teraz dla przycisku *FAŁSZ* (zobacz listing 1.10).

Listing 1.10. Tworzenie listenera dla przycisku FAŁSZ (plik QuizActivity.java)

```
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Nic tu się nie dzieje, ale wkrótce to zmienimy!
    }
});
mFalseButton = (Button) findViewById(R.id.false_button);
mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Nic tu się nie dzieje, ale wkrótce to zmienimy!
    }
});
}
```

Tworzenie komunikatów toast

Teraz musimy oprogramować nasze przyciski tak, aby były w pełni działające i funkcjonalne. Zgodnie z założeniami aplikacji naciśnięcie każdego z tych przycisków powinno powodować wyświetlenie na ekranie specjalnego komunikatu, który w terminologii systemu Android nosi nazwę *toast*. Komunikat *toast* to krótka wiadomość pojawiająca się na ekranie, której zadaniem jest informowanie użytkownika o czymś, co nie wymaga od niego podawania danych ani podejmowania żadnej akcji. W naszym przypadku utworzymy komunikaty *toast*, które będą informowały użytkownika, czy poprawnie odpowiedział na zadane pytanie quizowe (zobacz rysunek 1.14).



Rysunek 1.14. Komunikat toast informujący, że odpowiedź użytkownika była poprawna

Na początek powrócimy do pliku *strings.xml* i dodamy do niego zasoby tekstowe, które będą wyświetlane za pośrednictwem komunikatów *toast* (zobacz listing 1.11).

Listing 1.11. Dodawanie zasobów tekstowych dla komunikatów toast (plik *strings.xml*)

```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra jest stolicą Australii.</string>
    <string name="true_button">Prawda</string>
    <string name="false_button">Fałsz</string>
    <string name="correct_toast">Poprawna odpowiedź!</string>
    <string name="incorrect_toast">Niepoprawna odpowiedź!</string>
</resources>
```

Aby utworzyć komunikat *toast*, powinieneś wywołać następującą metodę z klasy *Toast*:

```
public static Toast makeText(Context context, int resId, int duration)
```

Parametr *Context* zazwyczaj ma postać instancji *Activity* (*Activity* jest podklasą klasy *Context*). Drugi parametr to identyfikator zasobu tekstowego, który będzie wyświetlany za pośrednictwem komunikatu *toast*. Parametr *Context* jest potrzebny, aby klasa *Toast* mogła odnaleźć identyfikator zasobu tekstowego i go użyć. Trzeci parametr to jedna z dwóch stałych klasy *Toast* odpowiadających za to, jak długo komunikat *toast* będzie wyświetlany na ekranie.

Po utworzeniu komunikatu możesz wyświetlić go na ekranie za pomocą wywołania metody *Toast.show()*.

W klasie *QuizActivity* utworzymy wywołania metody *makeText(...)* w kodzie listenera każdego z przycisków. Zamiast jednak pracować wpisywać wszystko ręcznie, spróbujemy dodać wywołania tej metody za pomocą mechanizmu dopełniania kodu, w jaki wyposażony jest pakiet *Android Studio*.

Zastosowanie mechanizmu dopełniania kodu

Mechanizm automatycznego dopełniania wpisywanego kodu może zaoszczędzić Ci mnóstwa czasu, zatem z pewnością warto zapoznać się z nim jak najwcześniej.

Rozpocznij wpisywanie dodatkowego kodu pokazanego na listingu 1.12. Kiedy dojdiesz do kropki znajdującej się po nazwie klasy *Toast*, na ekranie pojawi się okno dialogowe z listą sugerowanych metod i stałych z klasy *Toast*.

Listing 1.12. Tworzenie komunikatów toast (plik *QuizActivity.java*)

```
mTrueButton = (Button) findViewById(R.id.true_button);
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
            R.string.correct_toast,
            Toast.LENGTH_SHORT).show();
        // Nic tu się nie dzieje, ale wkrótce to zmienimy!
    }
});
```

```

mFalseButton = (Button) findViewById(R.id.false_button);
mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
            R.string.incorrect_toast,
            Toast.LENGTH_SHORT).show();
        //Nie tu się nie dzieje, ale wkrótce to zmienimy!
    }
});

```

Aby wybrać jedną z proponowanych sugestii, użyj klawiszy kursora. (Jeżeli chcesz zignorować podpowiedzi, po prostu pisz dalej — mechanizm dopełniania nie wstawi żadnego kodu, jeżeli nie użyjesz klawisza *Tab*, klawisza *Enter* ani nie klikniesz czegoś w oknie dialogowym podpowiedzi).

Z listy podpowiedzi wybierz metodę `makeText(Context context, int resID, int duration)`, a mechanizm dopełniania kodu automatycznie wstawi kompletne wywołanie metody w kodzie programu.

Teraz pozostaje Ci jedynie uzupełnić parametry wywołania metody `makeText`, o ile nie zrobiłeś tego już wcześniej podczas przepisywania kodu przedstawionego na listingu 1.12.

W wywołaniu metody `makeText(...)` jako argument `Context` przekazujemy instancję `QuizActivity`. Pamiętaj jednak, że nie możemy po prostu przekazać zmiennej `this`, tak jak mógłbyś się tego spodziewać. Dzieje się tak, ponieważ w tym miejscu kodu definiujemy klasę anonimową, gdzie `this` odnosi się do obiektu `View.OnClickListener`.

Ponieważ korzystamy z mechanizmu dopełniania kodu, nie musimy robić niczego dodatkowego, aby zaimportować klasę `Toast`. Jeżeli zaakceptujesz sugestie proponowane przez mechanizm dopełniania, wszystkie niezbędne klasy zostaną zaimportowane automatycznie.

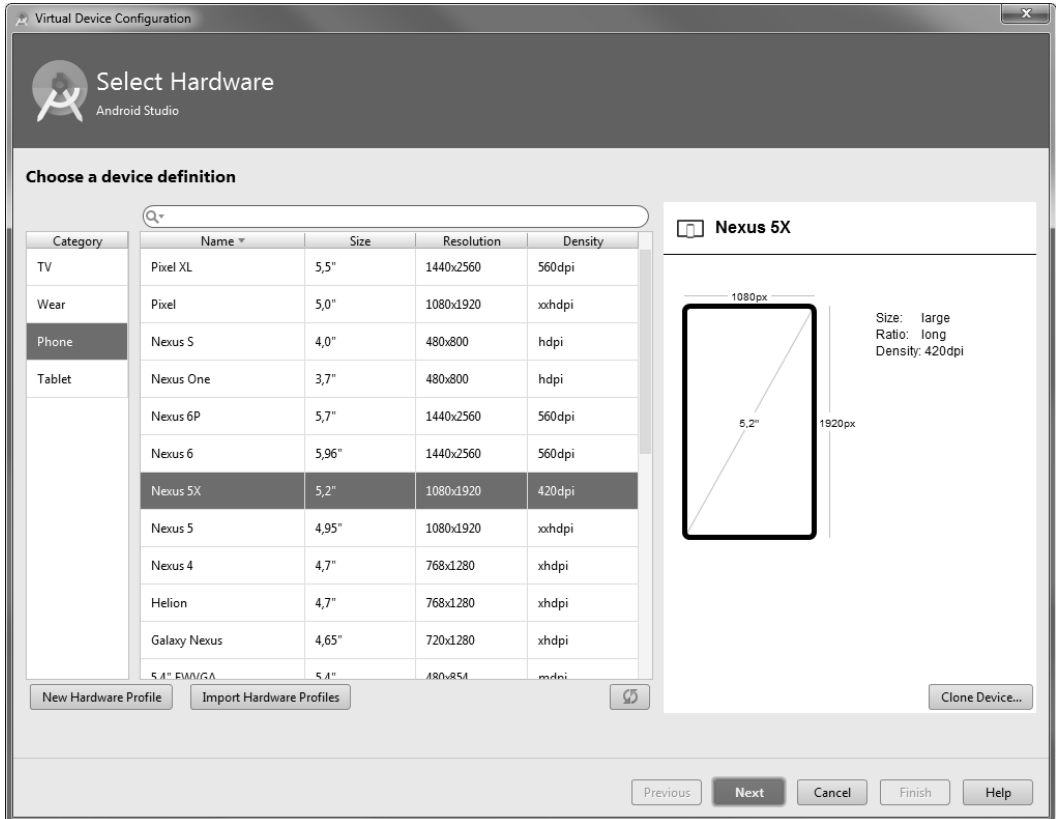
Zobaczmy teraz naszą nową aplikację w działaniu.

Uruchamianie aplikacji w emulatorze

Aby uruchomić aplikację dla systemu Android, musisz dysponować odpowiednim urządzeniem — może to być urządzenie sprzętowe lub *urządzenie wirtualne*. Urządzenia wirtualne są uruchamiane za pośrednictwem specjalnego emulatora, który jest częścią pakietu Android Studio.

Aby utworzyć urządzenie wirtualne dla systemu Android (ang. *Android Virtual Device* — AVD), z menu głównego Android Studio wybierz polecenie *Tools/Android/AVD Manager* (narzędzia/Android/menedżer urządzeń wirtualnych). Kiedy na ekranie pojawi się okno menedżera urządzeń wirtualnych, naciśnij przycisk *+Create Virtual Device...* (utwórz urządzenie wirtualne), znajdujący się w lewym dolnym rogu okna.

Na ekranie pojawi się okno dialogowe tworzenia nowego urządzenia wirtualnego, w którym znajdziesz wiele opcji konfiguracyjnych dla urządzeń wirtualnych. Dla naszego pierwszego urządzenia wirtualnego wybierz opcję emulacji telefonu *Nexus 5X*, tak jak to zostało pokazane na rysunku 1.15. Naciśnij przycisk *Next* (dalej).

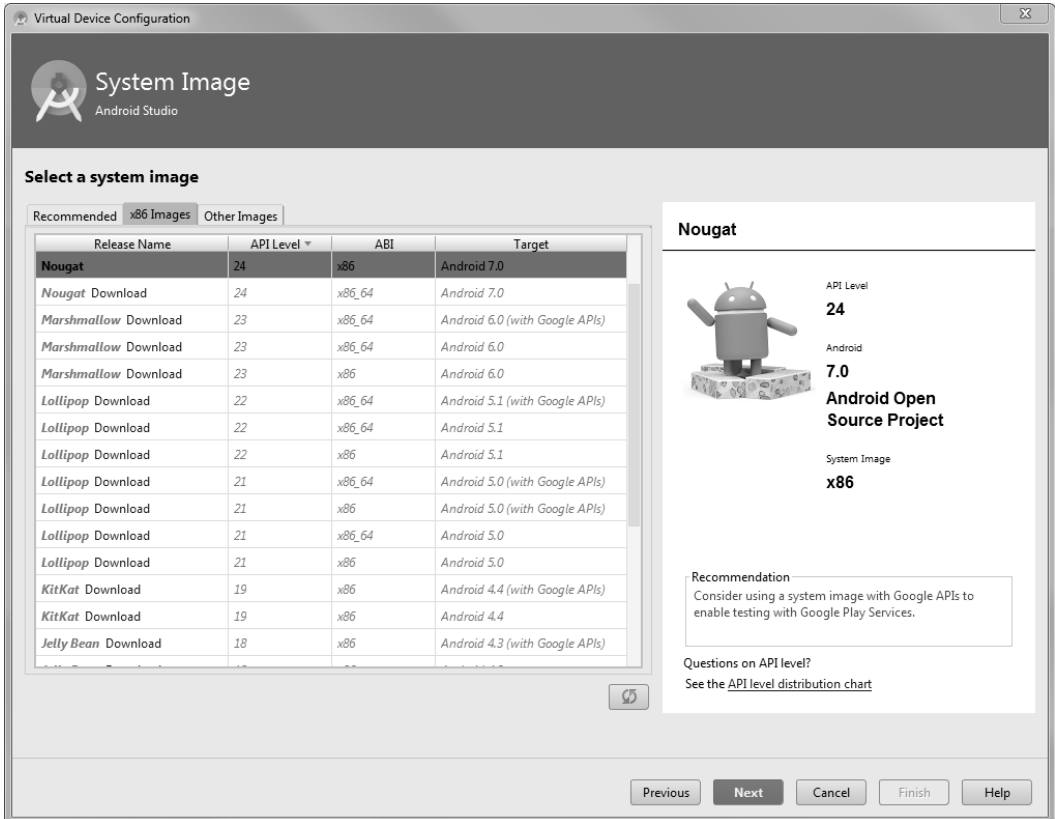


Rysunek 1.15. Wybieranie urządzenia wirtualnego

Na ekranie pojawi się kolejne okno kreatora urządzeń wirtualnych, pozwalające na wybranie obrazu systemu, który będzie wykorzystywany przez Twój emulator. W naszym przypadku wybierz dla emulatora obraz systemu *x86 Nougat* i naciśnij przycisk *Next* (zobacz rysunek 1.16). Pamiętaj, że w przypadku niektórych obrazów przed naciśnięciem przycisku *Next* konieczne będzie pobranie odpowiednich komponentów emulatora z sieci.

Wreszcie na koniec możesz sprawdzić i ewentualnie odpowiednio zmodyfikować właściwości emulatora (w razie potrzeby możesz to oczywiście zrobić również później). Teraz zmień tylko nazwę emulatora na coś, co ułatwi Ci później jego identyfikację, i naciśnij przycisk *Finish* (zakończ), tak jak to zostało pokazane na rysunku 1.17.

Po utworzeniu urządzenia wirtualnego możesz spróbować uruchomić na nim naszą aplikację *GeoQuiz*. Aby to zrobić, na pasku narzędzi Android Studio odszukaj i naciśnij przycisk *Run 'app'* (uruchom aplikację) lub po prostu naciśnij kombinację klawiszy *Ctrl+R*. Android Studio odszuka utworzone wcześniej urządzenie wirtualne, uruchomi je, a następnie zainstaluje w nim pakiet aplikacji i uruchomi samą aplikację.



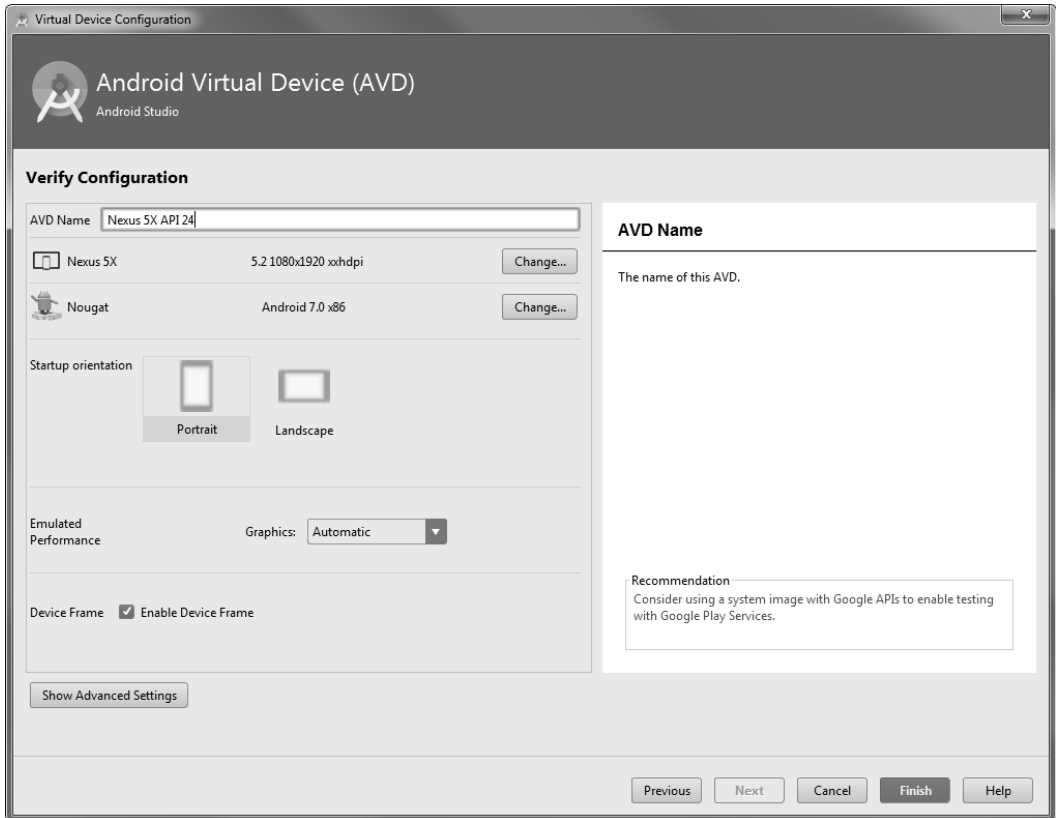
Rysunek 1.16. Wybieranie obrazu systemu dla emulatora

Uruchomienie emulatora może zająć kilka minut, ale po pewnym czasie nasza aplikacja GeoQuiz zostanie uruchomiona w urządzeniu wirtualnym. Spróbuj naciskać przyciski aplikacji i podziwiaj działanie komunikatów *toast*.

Jeżeli aplikacja GeoQuiz ulegnie awarii podczas uruchamiania lub kiedy naciśniesz przycisk, na karcie *Logcat* okna *Android Monitor* pojawi się szereg bardzo użytecznych informacji (jeżeli karta *Logcat* nie pojawia się automatycznie, możesz ją otworzyć, naciskając przycisk *Android Monitor* znajdujący się na pasku narzędzi w dolnej części okna programu Android Studio). W logu powinieneś poszukać informacji o wyjątkach (ang. *exceptions*), które są wyróżniane czerwonym kolorem czcionki, tak jak to zostało pokazane na rysunku 1.18.

Jeżeli wystąpi jakiś błąd, porównaj kod swojej aplikacji z kodem zaprezentowanym w książce i spróbuj znaleźć przyczynę problemów, a następnie ponownie spróbuj uruchomić aplikację (więcej szczegółowych informacji na temat karty *Logcat* i debugowania kodu znajdziesz w kolejnych dwóch rozdziałach).



Po zakończeniu testowania aplikacji nie wyłączaj emulatora, dzięki czemu za każdym razem przy kolejnych uruchomieniach aplikacji nie będziesz musiał cierpliwie czekać na uruchomienie emulatora.



Rysunek 1.17. Ustawianie właściwości emulatora

```
Text
at dalvik.system.NativeStart.main(Native Method)
Caused by: java.lang.NullPointerException
at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:21)
at android.app.Activity.performCreate(Activity.java:5008)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1079)
```

Rysunek 1.18. Przykładowy wyjątek NullPointerException w wierszu 21

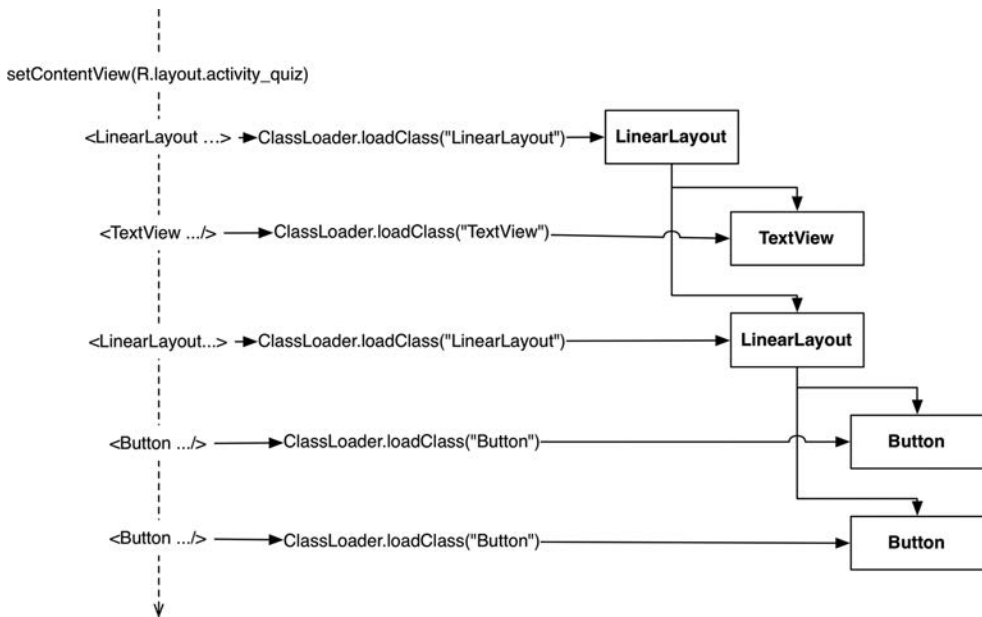
Aplikację możesz zatrzymać, naciskając przycisk *Back* (wstecz) emulatora. Przycisk *Back* ma kształt trójkąta skierowanego w lewo , a w starszych wersjach systemu Android kształt zawierającej w lewo strzałki . Po zatrzymaniu aplikacji wprowadź niezbędne modyfikacje, ponownie spróbuj uruchomić aplikację i przetestuj wprowadzone zmiany.

Emulator jest bardzo użyteczny, ale mimo to testowanie aplikacji na rzeczywistym, fizycznym urządzeniu daje znacznie bardziej miarodajne rezultaty. W rozdziale 2. pokażemy, w jaki sposób możesz uruchomić aplikację GeoQuiz na urządzeniu fizycznym. Oprócz tego w tym rozdziale wprowadzimy również do naszej aplikacji kilka nowych pytań, dzięki czemu będziesz mógł znacznie gruntowniej przetestować wiedzę użytkownika.

Dla dociekliwych: proces budowania aplikacji

Prawdopodobnie trapi Cię teraz wiele palących pytań na temat tego, jak naprawdę przebiega proces budowania aplikacji dla systemu Android. Zapewne zauważyłeś już, że Android Studio buduje cały projekt automatycznie w miarę wprowadzania do niego zmian. Podczas procesu budowania aplikacji Android Studio bierze zdefiniowane zasoby, kod aplikacji oraz plik manifestu *AndroidManifest.xml* (który zawiera metadane opisujące aplikację) i przetwarza je do postaci pakietu *.apk*. Plik pakietu jest podpisany kluczem debugera, który pozwala na uruchomienie aplikacji w emulatorze. Aby udostępnić pakiet *.apk* szerokim rzeszom użytkowników, musisz podpisać go docelowym kluczem wydania. Więcej szczegółowych informacji na temat tego procesu znajdziesz w dokumentacji dla deweloperów systemu Android dostępnej pod adresem <https://developer.android.com/studio/publish/preparing.html>.

W jaki sposób zawartość pliku *activity_quiz.xml* zamienia się w obiekty *View* w aplikacji? W trakcie procesu budowania aplikacji program *aapt* (Android Asset Packaging Tool) kompiluje pliki zasobów układu do bardziej kompaktowego formatu. Takie skompilowane zasoby są pakowane do pliku *.apk*. Następnie kiedy *setContentView(...)* jest wywoływany z poziomu metody *onCreate(Bundle)*, aktywność *QuizActivity* wykorzystuje klasę *LayoutInflater* do utworzenia instancji wszystkich obiektów *View* zdefiniowanych w pliku układu (zobacz rysunek 1.19).



Rysunek 1.19. Rozwijanie układu *activity_quiz.xml*

Zamiast definiować widok w pliku XML, możesz również utworzyć odpowiednie klasy programowo, niemniej odseparowanie warstwy prezentacji od logiki aplikacji przynosi wiele dobrego. Jedną z najważniejszych zalet jest możliwość korzystania ze zmian konfiguracji wbudowanych w SDK, o czym będziemy szerzej pisać w rozdziale 3.

Więcej szczegółowych informacji na temat tego, jak działają różne atrybuty XML oraz jak widoki są wyświetlane na ekranie, znajdziesz w rozdziale 9.

Narzędzia wspomagające budowanie aplikacji

Wszystkie aplikacje, o których pisaliśmy do tej pory, były uruchamiane z poziomu pakietu Android Studio. Narzędzia wspomagające budowanie aplikacji są zintegrowane ze środowiskiem IDE, dzięki czemu w tym procesie wykorzystywane są standardowe narzędzia systemu Android, takie jak aapt, a cały proces budowania aplikacji jest zarządzany przez Android Studio.

Jeżeli chcesz, możesz oczywiście budować aplikacje na zewnątrz, poza Android Studio. Najłatwiejszym sposobem wykonania takiego zadania jest użycie narzędzi do budowania aplikacji działających z poziomu wiersza poleceń konsoli. Wszystkie nowe systemy budowania aplikacji dla systemu Android wykorzystują narzędzie o nazwie Gradle.

Sam możesz ocenić, czy zagadnienia omawiane w tym podrozdziale będą dla Ciebie przydatne. Jeżeli nie, możesz spokojnie pominąć tę sekcję, choć zachęcamy do jej przeczytania. Nie powinieneś się jednak przejmować, jeżeli okaże się, że nie do końca rozumiesz składnię czy przeznaczenie poszczególnych poleceń, bądź jeżeli niektóre z tych poleceń nie będą działać tak, jak się tego spodziewałeś. Niestety szczegółowe omawianie narzędzi i przebiegu procesu budowania aplikacji z poziomu wiersza poleceń konsoli wykracza daleko poza ramy tej książki.

Aby skorzystać z narzędzi Gradle z poziomu wiersza poleceń konsoli, przejdź do katalogu, w którym przechowywane są pliki projektu Twojej aplikacji, i wykonaj polecenie przedstawione poniżej:

```
$ ./gradlew tasks
```

Jeżeli używasz systemu Windows, takie polecenie będzie wyglądało nieco inaczej:

```
> gradlew.bat tasks
```

Wykonanie tego polecenia spowoduje wyświetlenie listy dostępnych zadań, które możesz wykonać. Zadanie, którego poszukujemy, nosi nazwę `installDebug`. Aby je uruchomić, powinieneś wykonać polecenie przedstawione poniżej:

```
$ ./gradlew installDebug
```

W systemie Windows takie polecenie będzie wyglądało następująco:

```
> gradlew.bat installDebug
```

Wykonanie tego polecenia spowoduje zainstalowanie Twojej aplikacji na aktualnie podłączonym urządzeniu. Pamiętaj jednak, że nie spowoduje to uruchomienia aplikacji. Aby to zrobić, będziesz musiał skorzystać z programu uruchomieniowego (tzw. *launcher*) i uruchomić aplikację ręcznie.

Wyzwania

Wyzwania to seria ćwiczeń zamieszczanych na końcu każdego rozdziału, przeznaczonych do samodzielnego wykonania. Niektóre z nich są relatywnie proste i pozwalają na praktyczne wykorzystanie i pogłębienie wiadomości omawianych w danym rozdziale. Inne ćwiczenia są znacznie trudniejsze i często wymagają umiejętności rozwiązywania różnych problemów.

Serdecznie zachęcamy Cię do wykonywania tych ćwiczeń. Ich realizacja z całą pewnością spowoduje utrwalenie wiedzy nabytej w danym rozdziale, pozwoli Ci lepiej poznać swoje umiejętności i nabrać pewności siebie w ich używaniu oraz wypełni lukę pomiędzy tym, czego próbujemy Cię nauczyć, a tym, co będziesz w stanie samodzielnie zrobić, tworząc swoje własne aplikacje dla systemu Android.

Jeżeli utkniesz podczas realizacji któregoś z wyzwań, zrób sobie przerwę, przemyśl problem i powróć do niego po jakimś czasie z zupełnie świeżym spojrzeniem na to zagadnienie. Jeżeli to nie pomoże, zajrzyj na forum dyskusyjne tej książki, dostępne pod adresem <https://forums.bignerdranch.com/>. Na forum możesz zapoznać się z pytaniami i odpowiedziami innych czytelników, jak również możesz tam zamieszczać swoje pytania i rozwiązania.

Aby uchronić swój bieżący projekt przed przypadkowymi zmianami i błędami, przed rozpoczęciem realizacji ćwiczeń powinieneś wykonać kopię projektu i pracować nad wyzwaniami na nowej kopii, dzięki czemu oryginalny projekt aplikacji pozostanie nienaruszony.

Uruchom menedżera plików i przejdź do katalogu, w którym przechowujesz swoje projekty. Skopiuj katalog o nazwie *GeoQuiz* i utwórz jego kopię (na komputerach z systemem macOS możesz skorzystać z opcji *Duplicate*). Zmień nazwę nowo utworzonego katalogu na *GeoQuiz Challenge*. Powróć do Android Studio i z menu głównego wybierz polecenie *File/Import Project* (plik/importuj projekt). W oknie importu odszukaj i zaznacz katalog *GeoQuiz Challenge*, a następnie naciśnij przycisk *OK*. Skopiowany projekt pojawi się w nowym oknie Android Studio i będziesz mógł rozpocząć pracę.

Wyzwanie: dostosowywanie komunikatów toast do własnych potrzeb

W tym ćwiczeniu będziesz musiał zmodyfikować sposób wyświetlania komunikatów *toast* tak, aby wyświetlały się w górnej części okna zamiast w dolnej. Aby to zrobić, powinieneś użyć metody `setGravity` klasy `Toast`. Jako wartości parametru `gravity` powinieneś użyć stałej `Gravity.TOP`.

Więcej szczegółowych informacji na temat metody `setGravity` znajdziesz w dokumentacji na stronie <https://developer.android.com/reference/android/widget/Toast.html> - `setGravity(int, int, int)`.

Skorowidz

A

- abstrakcyjna
 - aktywność, 191
 - klasa Activity, 192
- Accessibility Scanner, 403
- adapter, 199
 - FragmentPagerAdapter, 253
 - FragmentStatePagerAdapter, 253
 - RecyclerView.Adapter, 498, 532
 - SoundAdapter, 415
- adaptery synchronizacji, sync
 - adapters, 603
- adres URL, 675
- aktualizacja
 - identyfikatora widoku, 228
 - informacji, 311
 - obiektu, 296
 - motywów graficznych, 280
 - pliku układu, 192, 203
 - stanu przycisku, 683
 - warstwy kontrolera, 67
 - warstwy modelu, 188
 - warstwy widoku, 65
 - widoku WebView, 640
 - wyświetlania obiektów, 238
- aktywności
 - abstrakcyjne, 191
 - bazowe, 502
 - deklarowanie, 124
 - dodawanie przycisku
 - podpowiedzi, 125
 - konfigurowanie, 120
 - komunikacja, 276
 - pobieranie wyników, 134
 - podklasa, 124
 - podrzędne, 134
 - sekwencja interakcji, 136
 - tworzenie, 121
 - uruchamianie, 127
- aktywność, activity, 28
 - CheatActivity, 119, 159
 - CrimeActivity, 161
 - CrimePagerActivity, 248, 251
 - główna, 138
 - MapsActivity, 699
 - QuizActivity, 120
- alarmy, 587
 - okresowe, 588
- aliasy zasobów, 358
- Allocation Tracker, 116
- alokacja zasobów pamięci, 116
- alternatywne zasoby tekstowe, 384
- alternatywny przycisk akcji, 294
- analizator statyczny, 112
- anatomia
 - handlera, 549
 - komunikatu, 548
- Android Asset Studio, 285
- Android Device Monitor, 306
- Android Lint, 112
- Android SDK, 143
- Android Studio, 25
 - okno powitalne, 29
 - okno projektu, 34
- animacja, 659
 - nocnego nieba, 668
 - słońca, 661
- animacje
 - typu circular reveal, 717
 - właściwości, 657
- animatory, 668
 - listy stanów, 715
- animowane
 - elementy listy stanów, 716
 - klasy wewnętrzne, 48
- ANR, Application Not Responding, 524, 561
- API animacji, 669
- aplikacja
 - Accessibility Scanner, 403
- aplikacja
 - BeatBox, 410
 - CriminalIntent, 155
 - DragAndDraw, 644, 650
 - Locatr, 673, 678
 - MockWalker, 676
 - NerdLauncher, 494
 - PhotoGallery, 515, 516, 517
- aplikacje typu lista-szczegóły, 156
- architektura
 - MVC, 409
 - MVVM, 409
- argumenty fragmentów, 235, 240
- Asset Studio, 285
- atrybut
 - android:layout_height, 39
 - android:layout_width, 39
 - android:orientation, 39
 - android:text, 40
 - owner, 629
 - protectionLevel, 617

atrybuty

- motywów, 232, 467
- przycisków, 471
- układu, 229
- widgetów, 39

B

badanie cyklu życia aktywności, 87

- baza danych SQLite, 301, 304
- debugowanie, 307
- odczytywanie danych, 313
- tworzenie, 303
- zapisywanie danych, 310

BeatBox, 410

bezpieczne dodawanie kodu, 148

biblioteka

- AppCompat, 162, 184, 280
- EventBus, 623
- Gson, 538
- Hamcrest, 439
- Mockito, 439
- Play Services, 699
- Play Services Maps, 697
- RxJava, 624

bitmapy

- skalowanie, 348
- wyświetlanie, 348

błąd ANR, 524, 561

buforowanie danych, 562

C

ciągi formatujące, 322, 325

ciągłość istnienia obiektów, 450

CriminalIntent, 155

- aktualizacja warstwy modelu, 188
- baza danych, 301
- diagram obiektowy, 160, 244, 247
- dodanie widgetu ViewPager, 247
- intencje niejawne, 321
- interfejs typu lista-szczegóły, 156

lista elementów, 235

- lokalizacja, 371
- okna dialogowe, 259
- osobne zadanie, 505
- pasek narzędzi, 279
- plan rozbudowy, 188
- reagowanie na naciśnięcie, 208
- stos aktywności, 243
- tworzenie aplikacji, 161
- układy, 213
- ułatwienia dostępu, 389
- uruchamianie aplikacji, 158
- widgety, 213
- własne zadanie, 506
- wyświetlanie informacji o elemencie, 235
- wyświetlanie listy, 188
- zamiana fragmentu, 158

cykl życia

- aktywności, 81, 97
- fragmentu, 167, 454, 180
- usługi, 596

czas, 589

czyszczenie aktywności, 99

D

dane

- extras, 237
 - JSON, 525
 - lokalizacji, 675
 - modelu, 318
 - o lokalizacji, 701
 - pobieranie bezpośrednie, 239
 - typu extras, 130
- debuger, 109
- debugowanie, 103, 112, 307, 710
- definicje typów kolumn, 319
- definiowanie
- fragmentu docelowego, 270
 - menu, 282
 - obszaru zawartości, 491
 - schematu, 302
 - widoku kontenera, 168

deklarowanie

- aktywności, 124
 - klasy CrimeListActivity, 195
 - wymagań aplikacji, 351
- Device Monitor, 306
- diagnozowanie problemów, 106
- diagram
- cyklu życia fragmentu, 167
 - dziedziczenia dla przycisków, 79
 - obiektowy aplikacji, 63, 244, 528
 - obiektowy fragmentów, 261
 - obiektów, 160
 - stanów aktywności, 82
- diamentowa notacja, 190
- dodawanie
- danych extras, 237
 - definicji kolorów, 657
 - elastyczności, 354
 - flagi, 505
 - ikon, 74
 - kolorów motywu, 466
 - kolumny, 323, 324
 - kontenera RecyclerView, 201
 - kwalifikatora, 359
 - kwalifikowanego pliku zasobów, 372
 - listenera do widoku, 78
 - marginisu, 705
 - menu, 679
 - metod wiążących, 425
 - opisów ścieżek, 343
 - opisów zawartości, 395
 - paska postępu, 636
 - pola, 323, 331
 - pola identyfikatora, 436
 - powiadomienia, 594
 - prywatnych uprawnień, 615
 - przycisków, 78, 294, 322
 - przycisku podpowiedzi, 125
 - przyspieszenia, 665
 - sprawdzenia uprawnień, 690
 - stałych, 526
 - stylu, 460
 - uprawnień, 521

- usługi do manifestu, 581
- widgetów, 221, 247
- zależności, 162
- zasobów do projektu, 75
- zasobów tekstowych, 50, 383
- zawartości okna dialogowego, 265
- zmiennych obiektowych, 341
- znaczników do mapy, 708
- dokumentacja, 151
 - klasy ViewAnimationUtils, 152
- dokumenty równoległe, 511
- domyślny
 - manifest aplikacji, 280
 - układ aktywności, 34
 - zasób aplikacji, 377
- dopełnianie kodu, 50
- dostarczanie kontekstu, 406
- dostęp do atrybutów motywu, 475
- dostosowywanie
 - komunikatów toast, 57
 - ustawień regionalnych, 379
- dotyk, 648
- dowiązkiwanie elementów listy, 206
- drzewo komponentów, 215
- dynamiczne
 - elementy listy, 228
 - układy interfejsu, 215
 - wyświetlanie kolumn, 539
 - odbiorniki rozgłoszeń, 612
- działanie
 - nawigacji hierarchicznej, 293
 - skanera dostępności, 404
 - ViewPager, 255
- dziedziczenie stylów, 462, 474
- dziennik komunikatów, 83

E

- edytor, 33
 - graficzny, 217, 387
 - Translations Editor, 378

- edytowanie
 - konfiguracji uruchamiania, 73
 - właściwości widoku, 225
- ekran główny, 507
- ekrany, 504
- eksploracja
 - motywu, 468
 - plików, 306
 - przez dotyk, 392
- elastyczność, 354
- elementy
 - aplikacji, 28
 - dynamiczne, 228
 - składowe niejawnej intencji, 327
 - typu shape drawable, 479
 - współdzielone, 719
 - XML drawable, 477, 483
- elewacja, 713
- emulator, 51, 673
- etykiety, 400

F

- filtr, 86
 - IntentFilter, 613
- filtrowanie powiadomień
 - pierwszoplanowych, 611
- fokus, 397
- folder zasobów asset, 417
- formatowanie daty, 233
- fragment
 - CrimeFragment, 159, 170, 238, 273
 - CrimeListFragment, 236
 - DatePickerFragment, 268
 - DialogFragment, 262
 - SunsetFragment, 659
 - SupportMapFragment, 704
- fragmenty, 182
 - adapter
 - FragmentPagerAdapter, 254
 - adapter
 - FragmentManagerAdapter, 253

- argumenty, 235, 240
- biblioteka wsparcia, 184
- cykl życia, 180, 454
- dane extras, 237
- diagram cyklu życia, 167
- docelowe, 270
- elastyczność prezentowania, 275
- hostowanie, 166
- implementowanie metod
 - cyklu życia, 173
- interfejsu użytkownika, 157
- komunikacja, 277
- narzędziowe, 162
- natywne, 162
- podłączanie widgetów, 175
- przechowywanie, 191
- przekazywanie danych, 267
- przesyłanie danych, 271
- responsywne, 278
- stan zachowania, 453
- transakcje, 179
- układu, 168
- uruchamianie aktywności, 235
- używanie, 182
- wyniki działania, 244
- wyświetlanie podglądu
 - układu, 171
- zachowywanie, 451
- zwracanie danych, 270
- framework Mockito, 439

G

- generowanie
 - getterów i setterów, 61
 - listy, 191
- geolokalizacja, 684, 693
- getter, 61
- gęstość pikseli ekranu, 229, 377
- Google Play Services, 673, 680, 682
- graficzne narzędzia, 214, 233
- graficzny edytor układów, 387
- grupy uprawnień, 689

H

handlery, 549
 komunikatów, 548
 hierarchia
 danych JSON, 529
 widoków, 38
 hierarchiczny układ widgetów
 i atrybutów, 38
 historia przeglądania, 641
 hostowanie fragmentów, 166
 HTTP, 515

I

identyfikator
 widoku, 228
 zasobu, 43
 ikona systemowa, 285
 ikony, 507
 opcje wyboru, 286
 implementowanie
 adaptera
 RecyclerView.Adapter, 498
 elementu ViewHolder, 498
 interfejsu
 CrimeFragment.Callbacks,
 366
 klasy Adapter, 203
 klasy ViewHolder, 203
 metody newInstance(), 240
 importowanie
 biblioteki Mockito, 439
 tabeli, 305
 zasobów, 416
 indeksy, 319
 informacje
 o elemencie, 235
 o lokalizacji, 685
 instalowanie Android Studio, 25
 integrowanie aktywności
 CrimePagerActivity, 251
 intencje, 128, 493
 dane typu extras, 130
 jawne, explicit intents, 129,
 326, 500

niejawne, implicit intent, 129,
 321, 496, 630
 przekazywanie danych, 129
 rozgłoszeń, 605
 usługi, 580
 zwracanie, 135
 interakcja
 obiektów, 445
 widoku z użytkownikiem, 571
 interfejs
 Callbacks, 361
 List<E>, 190
 Material Design, 711, 727
 Observable, 428
 SearchView.OnQueryText
 ↪Listener, 571
 WebChromeClient, 635, 636
 interfejsy
 typu lista-szczegóły, 156, 353
 użytkownika, 34, 155
 fragmenty, 157
 hostowanie fragmentów,
 166
 tworzenie fragmentu, 170
 układy, 213
 widgety, 213
 zwrotne układów, 361
 interpolator, 665

J

jawne intencje, explicit intents,
 129, 326, 500
 jednostka
 dp, 229
 in, 230
 mm, 230
 pt, 230
 px, 230
 sp, 229
 JSON, JavaScript Object
 Notation, 525

K

kalendarz, 266
 karty, 722
 katalog drawable, 75
 katalogi zasobów, 386
 klasa
 JobServices, 599
 Activity, 192
 Adapter, 197, 203
 AlertDialog, 260
 AppCompatActivity, 281
 ArgbEvaluator, 667
 AsyncTaskLoader, 538
 BeatBox, 418
 BeatBoxActivity, 412
 BeatBoxFragment, 411
 ContentValues, 310
 Context, 342
 Crime, 165
 CrimeActivity, 192
 CrimeBaseHelper, 303
 CrimeCursorWrapper, 316
 CrimeFragment, 172
 CrimeLab, 189, 308
 CrimeListActivity, 195
 CrimePagerActivity, 248
 CursorWrapper, 314
 DateFormat, 388
 Dialog, 260
 DialogFragment, 261
 FileProvider, 343
 FragmentManager, 177, 180
 JobScheduler, 599
 JobService, 603
 LocatrFragment, 678
 PhotoHolder, 542
 QuizActivity, 28
 R, 114
 RecyclerView, 197, 201
 ShareCompat, 337
 SingleFragmentActivity, 355
 SoundHolder, 415
 SoundPool, 435
 SoundViewModel, 424
 SQLiteOpenHelper, 303

- UUID, 165
 - View, 42
 - ViewHolder, 197–203, 532
 - ViewPager, 247
 - klasy
 - testowe, 440
 - typu singleton, 189
 - wiążące, 413
 - klucz API, 698, 708
 - klucze obce, 319
 - kod
 - powrotu, 135
 - żądania, 134
 - kolejka
 - komunikatów, 545
 - żądań, 558
 - kolor tła, 471
 - kolory, 657
 - modyfikowanie, 665
 - kolumna
 - dodawanie, 323
 - odczytywanie danych, 324
 - zapisywanie danych, 324
 - kompatybilność aplikacji, 144
 - komponenty widoków, 722
 - komunikacja
 - między aktywnościami, 276
 - między fragmentami, 277
 - z intencjami, 128
 - z wątkiem głównym, 544
 - komunikat, 548
 - Message, 550
 - toast, 49
 - konfigurowanie
 - aktywności, 120
 - Google Play Services, 680
 - klucza debugowania, 710
 - kontenera, 414
 - map, 700
 - menu, 678
 - nowej aktywności, 33
 - testu, 442
 - urządzeń, 72, 90, 386
 - widoku, 202
 - konstruktor JSONObject, 530
 - kontekst, 406
 - aplikacji, 320
 - kontener
 - definiowanie widoku, 168
 - FrameLayout, 169
 - GridView, 209
 - ListView, 209
 - RecyclerView, 197–201, 414, 518
 - konwersacja, 200, 267
 - konwersja układu, 216
 - kwalifikatory, 359, 385
 - konfiguracji, 382
 - rozmiarów ekranu, 370
- ## L
- lambda, 431
 - launcher NerdLauncher, 507
 - Lint, 112
 - lista
 - kontaktów, 330
 - obiektów, 190, 421
 - obiektów Sound, 421
 - stanów, 481
 - warstw, 482
 - zależności, 163
 - listener, 47
 - OnClickListener, 208
 - SearchView.OnQueryText
 - ↳ Listener, 571
 - listy
 - dowiązkiwanie elementów, 206
 - elementy dynamiczne, 228
 - przeładowywanie, 242
 - tworzenie, 190
 - wyświetlanie, 187
 - wyświetlanie elementów, 247
 - logowanie, 101
 - cyklu życia aktywności, 83
 - śladów stosu, 106
 - wyjątku, 107
 - lokalizacja, 671, 701
 - dat, 388
 - interfejsu aplikacji, 371
 - interfejsu użytkownika, 371
 - zasobów, 372
 - zdjęć, 344
 - looper, 545
 - LRU, least recently used, 562
 - luki w zabezpieczeniach, 141
- ## Ł
- ładowanie
 - plików dźwiękowych, 436, 437
 - wstępne, 562
 - łańcuchy ilościowe, quantity strings, 300
 - łącza HTTP, 641
- ## M
- magistrala zdarzeń, event bus, 622
 - manifest aplikacji, 521
 - deklarowanie aktywności, 124
 - mapy, 697
 - dodawanie znaczników, 708
 - Google, 698
 - konfigurowanie, 700
 - w systemie, 697
 - ze znacznikami, 709
 - marginesy, 231, 705
 - maszyna wirtualna AVD, 354
 - matchery, 439
 - Material Design, 711
 - mechanizm
 - dopełniania kodu, 50
 - wiązania danych, 409–433
 - menedżer ActivityManager, 128
 - menu, 282
 - kreatora aktywności, 121
 - przepełnienia, overflow menu, 283
 - rozwijanie zasobu, 288
 - tworzenie, 288
 - wybranie elementu, 291

metoda
 bind(), 427
 buildUrl(), 566, 684
 fetchItems(), 526, 527
 fileList(), 342
 Fragment.setArguments(), 240
 getCacheDir(), 342
 getCrime(), 315, 317
 getDir(), 342
 getFilesDir(), 342
 getUrlBytes(), 519
 getUrlString(), 519
 isServiceAlarmOn(), 590
 newInstance(), 240
 newIntent(), 593
 onClicked(), 446
 onCreate(), 173, 305
 onCreateOptionsMenu(), 570
 onCreateView(), 174
 onDraw(), 652
 onResume(), 244
 onSaveInstanceState, 95
 onStart(), 244
 openFileInput(), 342
 openFileOutput(), 342
 requestPermissions(), 691
 setCrimes(), 318
 startWithTransition(), 722
 updateDate(), 275
 updateSubtitle(), 294
 metody cyklu życia fragmentu, 173
 miniatury, 544
 mipmap, 485
 model widoku, 424
 modyfikowanie
 aktywności, 644
 atrybutów przycisków, 471
 modyfikowanie
 kolorów, 665
 motywu, 464
 motyw graficzny, 232, 280, 459
 aplikacji, 463
 przycisków, 492

MVC, Model-View-Controller, 63, 409
 MVVM, Model-View-ViewModel, 409

N

nadpisywanie
 atrybutów motywu, 467
 metody, 173
 nadzorca, authority, 343
 fragmentów, 360
 naprawianie błędu, 114
 narzędzia, 25
 animacji, 669, 717
 graficzne, 233
 ORM, 302
 wspomagające, 56
 narzędzie
 Allocation Tracker, 116
 Android Lint, 112
 Custom Locale, 381
 Layout Inspector, 115
 nawiązywanie połączenia, 683
 nawigacja
 hierarchiczna, 292, 293
 liniowa, 393
 rodowa, 292
 temporalna, 292
 niejawne intencje niejawne, implicit intent, 129, 321, 496, 630
 elementy składowe, 327
 niszczenie aktywności, 87

O

obiekt
 AnimatorSet, 668
 AssetManager, 419
 GoogleMap, 704
 Handler, 549
 HandlerThread, 549
 LocationRequest, 685
 Looper, 545, 549
 MenuItem, 296
 Message, 549

PagerAdapter, 249
 PendingIntent, 589
 Point, 652
 SharedPreferences, 573
 Sound, 420
 SoundPool, 435
 ThumbnailDownloader, 554
 ViewHolder, 199
 ViewPager, 249
 obiekty
 kontrolera, 64
 nasłuchujące, listenery, 47
 modelu, 63, 316
 opcji, options objects, 707
 pozorne, mock objects, 439, 456
 widoku, 38, 63
 obracanie prostokątów, 654
 obrazy
 9-patch, 485, 488
 Mipmap, 484
 obrót
 urządzenia, 452
 urządzenia z fragmentem, 452
 obsługa
 dotknięcia ekranu, 208
 historii przeglądania, 641
 komunikatów, 552, 555
 łączy, 641
 wyników, 137
 zmian konfiguracja, 639
 obszar zawartości, 491
 odbieranie
 intencji rozgłoszenia, 608
 rozgłoszeń systemowych, 606
 wywołań zwrotnych, 289
 odbiorniki
 dynamiczne, 606
 odpowiedzi, result receiver, 618
 rozgłoszeń, 606, 609
 odczytywanie danych
 z bazy, 313
 z kolumny, 324
 odmowa dostępu, 687

- odstępy, 231
 - odszukiwanie zasobów, 386
 - odświeżanie danych modelu, 318
 - odtworzenie
 - marginesów, 257
 - plików dźwiękowych, 435, 438
 - odwołania
 - do atrybutów motywu, 232
 - do zasobów, 77
 - do zasobu lokalnego, 287
 - do zasobu tekstowego, 376
 - ograniczanie zasięgu rozgłoszeń, 614
 - okna
 - dialogowe, 259
 - dodawanie zawartości, 265
 - uprawnień, 692
 - narzędziowe, 33
 - okno
 - AlertDialog, 266
 - Asset Studio, 286
 - debugera, 109
 - graficznego narzędzia, 214
 - kreatora aktywności, 122
 - Logcat, 609
 - projektu, 33
 - opcje wyboru ikon, 286
 - operacje sieciowe, 582
 - opis zawartości, content
 - description, 395
 - orientacja urządzenia, 91, 450
 - ORM, Object-Relational Mappers, 302, 319
 - os Z, 713
- P**
- pakiet
 - APK, 484, 676
 - Espresso, 455
 - paleta narzędzi, 214
 - panel
 - Logcat, 85
 - właściwości, 215
 - parametry układu, layout
 - parameters, 224
 - parsowanie danych, 529
 - pasek
 - akcji, action bar, 280, 299
 - narzędzi, 279, 281, 299
 - postępu, 636
 - pętla komunikatów, 545, 554
 - piaskownica, sandbox, 301
 - pierwsza aplikacja, 27
 - piksel, 229, 377
 - pisanie testów, 444
 - plik
 - activity_quiz.xml, 45
 - AndroidManifest.xml, 139, 521, 581, 615
 - app/build.gradle, 146, 162, 440, 699
 - BeatBox.java, 418, 435
 - BeatBoxFragment.java, 414, 419, 427
 - BoxDrawingView.java, 648, 650, 652, 653
 - CheatActivity.java, 132, 135, 150
 - Crime.java, 165
 - CrimeBaseHelper.java, 303
 - CrimeCursorWrapper.java, 315
 - CrimeDbSchema.java, 302
 - CrimeFragment.java, 175, 273
 - CrimeLab.java, 189, 309
 - CrimeListActivity.java, 364
 - CrimeListFragment.java, 202–207, 228, 236, 288
 - CrimePagerActivity.java, 250, 251
 - DatePickerFragment.java, 262, 265, 268, 272
 - DragAndDrawActivity.java, 644
 - FlickrFetchr.java, 519, 526, 564, 566, 629, 684
 - fragment_crime.xml, 170
 - fragment_crime_list.xml, 201
 - GalleryItem.java, 628
 - layout/dialog_date.xml, 265
 - layout/fragment_photo_gallery.xml, 518
 - layout/list_item_crime.xml, 224
 - layout-land/activity_quiz.xml, 126
 - list_item_crime.xml, 203, 214
 - list_item_sound.xml, 426, 448
 - LocatrFragment.java, 683, 685, 690
 - NerdLauncherFragment.java, 495–498, 501
 - PhotoGalleryActivity.java, 518
 - PhotoGalleryFragment.java, 522, 527, 532, 542, 575, 587
 - PictureUtils.java, 348
 - PollService.java, 579, 582, 586, 611
 - QueryPreferences.java, 574, 584, 609
 - Question.java, 62
 - QuizActivity.java, 42, 46, 47
 - R.java, 44
 - res/drawable/button_beat_box_normal.xml, 479
 - res/layout.fragment_crime.xml, 340
 - res/layout/fragment_beat_box.xml, 411, 413
 - res/layout/list_item_sound.xml, 478
 - res/menu/fragment_crime_list.xml, 283, 287
 - res/values/dimens.xml, 705
 - res/values/strings.xml, 282, 396, 569
 - res/values/styles.xml, 281, 460, 465, 473
 - res/xml/files.xml, 343
 - SingleFragmentActivity.java, 193, 355
 - Sound.java, 420
 - SoundViewModel.java, 424, 425, 429, 446

- plik
 - SoundViewModelTest.java, 442, 445
 - StartupReceiver.java, 607
 - strings.xml, 50, 325, 373
 - SunsetFragment.java, 661, 665, 667, 668
 - ThumbnailDownloader.java, 546, 551, 555
 - values/strings.xml, 262
 - VisibleFragment.java, 612, 619
- pliki
 - dźwiękowe, 435
 - ładowanie, 436
 - odtworzenie, 438
 - usuwanie, 449
 - pamięci podręcznej, 342
 - preferencji, 573
- plaszczyszna Z, 714
- pobieranie
 - argumentów, 241
 - bezpośrednie danych, 239
 - danych extras, 237
 - danych JSON, 525
 - danych z listy kontaktów, 333
 - informacji o lokalizacji, 685, 701
 - listy zdjęć, 566
 - nazwy kontaktu, 333
 - odwołań do widgetów, 46
 - wyników działania fragmentów, 244
 - zasobów assets, 418
 - zdjęć, 560
- podgląd układu, 41
- podklasa nowej aktywności, 124
- podłączanie
 - listy obiektów, 421
 - modelu widoku, 426
 - przycisku, 448, 686
 - urządzenia fizycznego, 71
 - widgetu, 45, 175
 - zasobu typu asset, 420
- podpinanie opisów ścieżek, 344
- podpisywanie aplikacji, 710
- podział na strony, 538
- pole
 - EditText, 401
 - identyfikatora pliku, 436
 - TextView, 225
- polecenia, commands, 580
- połączenia
 - bezpośrednie, 423
 - sieciowe, 521
- poprawianie listy, 406
- porównanie wątków regularnych, 523
- porządkowanie zadań
 - asynchronicznych, 535
- powiadomienie, notification, 593
 - o zdarzeniach, 406
 - pierwszoplanowe, 611
 - Snackbar, 725
- powiązanie, constraint, 215
 - dolne, 223
 - górne, 222
 - przeciwnie, 215
- poziomy
 - logowania, 100
 - ochrony, protection levels, 615
- pozorny obiekt, 443
- preferencje stylu kodu, 61
- priorytetyzacja zasobów
 - alternatywnych, 383
- proces budowania aplikacji, 55
- procesy, 508
- przechowywanie
 - fragmentów, 191
 - plików, 342
 - zdjęć, 339
 - wyjątków, 111
- przeglądanie
 - podłączonych urządzeń, 72
 - sieci WWW, 627
- przekazywanie
 - aktualizacji, 312
 - danych, 267
 - danych do fragmentu, 268
 - handlerów, 554
 - obiektów, 237, 422
- przekształcanie, translating, 661
- przeładowywanie
 - listy, 242
 - zawartości kontenera, 246
- przełączanie
 - między zadaniami, 503
 - tytułu, 295
- przeźródło nazw app, 284
- przesyłanie danych, 271
- przycisk
 - akcji, action item, 279, 295
 - alternatywny akcji, 294
 - Button, 79
 - ImageButton, 79
 - FAB, 724
 - narzędzi powiązań, 217
 - negatywny, negative button, 263
 - neutralny, neutral button, 263
 - plywający, 724
 - podpowiedzi, 125
 - pozytywny, positive button, 263
 - wyszukiwania, 686
 - zablokowany, 336
- przyciski
 - aktualizacja stanu, 683
 - dodawanie, 322
 - modyfikowanie atrybutów, 471
 - motywy, 492
 - o jednolitym wyglądzie, 478
 - tworzenie układu, 415
 - zaokrąglanie, 479
 - zmiana rozstawu, 478
- przyspieszenie, 665
- pułapka, 108
 - na wyjątki, 111
- punkt przzerwania, 108
- pusty widok, 300

R

- raport, 328
 - wyjątku, 105
- raportowanie wersji SDK, 153
- reagowanie na naciśnięcie, 208, 295

reguła jednej odpowiedzialności,
SRP, 424

rejestrowanie odbiornika
dynamicznego, 612

rekomendacje skanera
dostępności, 405

renderowanie, 652

responsywny DialogFragment,
278

rotacja widoku, 664

rozgłoszenia
systemowe, 606
uporządkowane, 617

rozmiar
ekranu, 369
widoku, 219

rozmieszczanie widoków, 256

rozwiązywanie intencji
niejawnych, 496

rysowanie na mapie, 707

S

samodzielny odbiornik
rozgłoszeń, 606

scena, 657

scentralizowane przechowywanie
danych, 189

schemat bazy danych, 302

sekcja Develop, 151

sekwencja
interakcji aktywności, 136
zdarzeń, 268

setter, 61

siatka przycisków, 422

sieć, 519

singletony, 210

skalowanie
bitmap, 348
widoku, 664

skaner dostępności, 404

słowo kluczowe private, 565

Snackbar, 726

sprawdzanie
pokrycia zasobów, 378
tłumaczeń, 378

uprawnień, 688, 690
wyników wyszukiwania, 584

SQLite, 301

SRP, Single Responsibility
Principle, 424

stała
ACITION_CANCEL, 648
ACTION_DOWN, 648
ACTION_MOVE, 648
ACTION_UP, 648

statyczny analizator kodu, 112

sterowanie
alarmami, 590
obrotem, 77
szybkością odtwarzania, 457

stos aktywności, 139, 243, 502

strategia LRU, 562

styl przycisku, 473

style, 232, 459, 460

symulowane dane lokalizacji, 675

systemy ORM, 302, 319

szkielet bazy danych, 303

szybkość odtwarzania dźwięku, 457

Ś

ścieżki, 343

śląd stosu, 104

śledzenie
alokacji zasobów pamięci, 116
zdarzeń, 649

środowisko SDK, 25, 143

T

tabela starszeństwa, 386

tabele
aktualizowanie wierszy, 311
import, 305
tworzenie, 305
wstawianie wierszy, 311

tablet, 359

tekst powiadomienia, 593

testowanie
interakcji obiektów, 445
usług lokalizacji, 673

ustawień regionalnych, 381

zasobów alternatywnych, 387

testy
integracyjne, 441, 455
jednostkowe, 435

tło
przycisków, 486, 487, 490
zmienianie koloru, 471

tłumaczenia, 378

transakcje fragmentów, 179

transformacja, 669
bohatera, hero transition, 719
współdzielonego elementu,
719

tryb StrictMode, 561

tworzenie
adaptera, 205, 415
aktywności, 121, 161
alternatywnego zasobu
tekstowego, 385
aplikacji
BeatBox, 411
Locatr, 673
NerdLauncher, 494
PhotoGallery, 517
bazy danych, 303
dynamicznych elementów
listy, 228
dziennika komunikatów, 83
filtra, 86
foldera zasobów, 417
fragmentu, 644
fragmentu interfejsu
użytkownika, 170
górnego powiązania, 222
instancji BeatBox, 419
instancji DialogFragment,
261
jawnych intencji, 500
katalogu zasobów, 91
klasy, 60
Crime, 165
CrimeBaseHelper, 303
CrimeCursorWrapper, 314
CrimeFragment, 172
CrimePagerActivity, 248

tworzenie

- LocatrFragment, 678
- SoundHolder, 415
- SoundPool, 435
- SoundViewModel, 424
- testowej, 440
- typu singleton, 189
- komunikatów toast, 49
- komunikatu Message, 550
- kontrolerów, 194
- listenera dla przycisku, 47, 49
- listy obiektów, 190, 421
- menu, 288
- modelu widoku, 424
- obiektów nasłuchujących, 47
- obiektu Sound, 420
- obrazu 9-patch, 489
- odbiornika dynamicznego, 612
- opisów zawartości, 396
- pliku menu, 283
- podkatalogu, 418
- pozornego obiektu, 443
- projektu aplikacji, 29, 160
- stylu przycisku, 473
- tabeli, 305
- układu, 214, 233, 658
 - dla przycisków, 415
 - interfejsu użytkownika, 34
- usługi IntentService, 579
- wątku tła, 546
- widoku BoxDrawingView, 646
- zadania AsyncTask, 522
- zaokrąglonych przycisków, 479
- zasobów tekstowych, 40
- zasobu alternatywnego, 359
- typ zasobu, Resource type, 343
- typy
 - fragmentów, 162
 - widoków kontenera RecyclerView, 211
- tytuł
 - paska narzędzi, 294
 - przycisku akcji, 295

U

- uchwyt powiązania, constraint handle, 222
- udostępnianie kontekstu, 400
- układ, layout, 28
 - activity_cheat.xml, 124
 - activity_quiz.xml, 55
 - ConstraintLayout, 215, 224
 - dla orientacji poziomej, 91, 93
 - dla przycisków, 415
 - dwupanelowy, 357
 - dynamiczny, 215
 - przechowujący fragmenty, 191
 - widoków, 341
 - fragmentu CrimeFragment, 170
 - z dwoma kontenerami fragmentów, 356
- układy
 - atrybuty, 229
 - dodawanie elastyczności, 354
 - dodawanie zależności, 217
 - graficzne narzędzia, 214, 233
 - konwersja, 216
 - marginesy, 231
 - odstęp, 231
 - ostrzeżenia, 218
 - parametry, 224
 - tworzenie, 214
 - ustawienia wewnętrzne, 224
- ukrywanie
 - obiektu, 237
 - podtytułu, 296
- ułatwienia dostępu, 389, 398
- uporządkowane intencje rozgłoszeń, 618
- uprawnienia
 - dla odbiornika rozgłoszeń, 616
 - do korzystania z kontaktów, 334
 - do wyznaczania lokalizacji, 681
 - grupy, 689
 - prywatne, 614
 - sprawdzanie, 688
 - usługi TalkBack, 391
- uruchamianie
 - aktywności, 127, 501
 - aktywności przez fragment, 235
 - alarmu, 587
 - animatorów, 668
 - aparatu fotograficznego, 345
 - aplikacji, 140, 158
 - na urządzeniu fizycznym, 71
 - w emulatorze, 51
 - intencji, 346
 - testu, 447
 - zadania, 504
 - zadań asynchronicznych, 521
- urządzenia
 - typu Android Wear, 595
 - wirtualne, 52
- urządzenie
 - konfiguracje, 90
 - obracanie, 452
 - pozioma orientacja, 91
 - zapisywanie danych, 95
 - zmiana orientacji, 90, 95, 450, 638
- usługa, 596
 - AlarmManager, 585
 - IntentService, 579
 - Google Play, 671
 - TalkBack, 389, 390
- usługi
 - działające w tle, 579
 - typu non-sticky, 597
 - typu sticky, 597
 - ułatwień dostępu, 389
- ustawianie
 - etykiety, 402
 - podtytułu paska narzędzi, 294
- ustawienia
 - kont, 604
 - regionalne, 379
 - niestandardowe, 381
 - rozmiaru widoków, 219

- usługi TalkBack, 390
 - wewnętrzne układu, 224
 - właściwości pola, 226
 - usuwanie
 - aktywności MapsActivity, 699
 - aplikacji, 308
 - biblioteki Play Services, 699
 - plików dźwięków, 449
 - zapytania, 575
 - zadań, 558
 - używanie
 - alarmów, 588
 - aliasów zasobów, 358
 - Android Asset Studio, 285
 - argumentów fragmentów, 245
 - biblioteki AppCompat, 280
 - biblioteki EventBus, 623
 - biblioteki RxJava, 624
 - geolokalizacji, 684, 693
 - Google Play Services, 682
 - handlerów, 549
 - klasy AppCompatActivity, 281
 - launchera NerdLauncher, 507
 - odbiorników rozgłoszeń, 609
 - przycisków FAB, 725
 - wiązania danych, 412, 424
 - zasobów typu asset, 430
- ## W
- warstwa modelu, 188
 - warstwy materiałów, 712
 - wartości atrybutu
 - protectionLevel, 617
 - wątek, 559
 - główny, 523, 532
 - interfejsu użytkownika, 524
 - tła, 525, 546
 - wersja
 - docelowa SDK, 145
 - kompilacji SDK, 147
 - minimalna, 147
 - raportowanie, 153
 - wersje
 - Android SDK, 143
 - API, 143
 - systemu Android, 144
 - wiązanie
 - danych, data binding, 409–433
 - do modelu widoku, 425
 - usług, 598
 - lokalne, 598
 - zdalne, 599
 - widget, 35
 - DatePicker, 265
 - FrameLayout, 168
 - ImageView, 215, 221, 222
 - TextView, 220
 - ViewPager, 247
 - widżety
 - dodawanie, 221
 - podłączanie, 45
 - uchwyty powiązania, 222, 223
 - widok, 38, 218
 - BoxDrawingView, 646, 649
 - crime_title, 225
 - DatePicker, 265, 266
 - RecyclerView, 187, 197
 - SearchView, 568, 571, 573
 - TextView, 219, 226
 - WebView, 631, 638
 - widoki
 - fragmentu, 157
 - kontenera, 168
 - niestandardowe, 643
 - proste, 645
 - rozmieszczanie, 256
 - ustawienia rozmiaru, 219
 - złożone, 645
 - właściwości
 - pola, 226
 - transformacji widoków, 663
 - widoku TextView, 219
 - włączanie
 - transformacji aktywności, 721
 - usługi TalkBack, 392
 - wskaźnik postępu wyszukiwania, 696
 - wstrzykiwanie obiektów JavaScript, 639
 - wybieranie
 - elementu menu, 291
 - ikon, 286
 - obrazu systemu, 53
 - obsługiwanych urządzeń, 31
 - typu aktywności, 32
 - urządzenia wirtualnego, 52
 - widoków, 206
 - wygląd domyślnych widgetów, 35
 - wyglądanie aplikacji, 577
 - wyjątki, 104
 - wykonywanie zdjęć, 339
 - wymagania aplikacji, 351
 - wyodrębnianie metody, 274
 - wypełnianie widoku fragmentu, 238
 - wrażenia lambda, 431
 - wysyłanie
 - intencji rozgłoszenia, 611
 - komunikatu Message, 550
 - raportu, 328, 331
 - zadania LocationRequest, 686
 - wyszukiwanie, 584
 - aktywności, 334
 - zasobów asset, 419
 - zdjęć, 564, 567, 684, 693
 - wyświetlanie
 - bitmap, 348
 - elementów listy, 247
 - fragmentu DialogFragment, 263
 - informacji o elemencie, 235
 - list, 187
 - obiektów, 238
 - obrazów, 541
 - obszaru zawartości, 490
 - plików, 373
 - podtytułu, 296
 - projektu, 43
 - stron internetowych, 635
 - zdjęć, 693
 - wywołania zwrotne wiązania danych, 448

wzorzec

- MVC, 63, 409
 - aktualizacja warstwy kontrolera, 67
 - aktualizacja warstwy widoku, 65
 - zalety stosowania, 64
- MVVM, 409, 424
- Singleton, 189, 210

X

XML

- odwołania do zasobów, 77

Z

- zachowywanie danych, 573
 - fragmentów, 451, 453
- zadania, 493, 502
 - asynchroniczne, 521
 - długotrwałe, 622
 - drugoplanowe, 515, 602
- zadanie
 - AsyncTask, 522, 532, 536, 559
 - CriminalIntent, 502
 - FetchItemsTask, 576
 - NerdLauncher, 504
 - SearchTask, 693
- zależności, 162, 163
 - pozorne, 443
 - testów, 439
- zamiana
 - fragmentu, 158
 - głównego widoku, 216
 - pliku układu, 413
- zamykanie luk, 141
- zapisywanie
 - danych w bazie, 310
 - danych w kolumnie, 324
 - stanu, 653
 - stanu wyświetlania, 298
 - zapytania, 575
- zapytania, 574

zarządzanie

- alarmami, 590
- fragmentami, 253

zasięg

- rozgłoszeń, 614
- zależności Mockito, 440

zasoby, 43

- alternatywne, 90, 359, 387
- assets, 418
- domyślne, 375
- kolorów, 460
- tekstowe, 383
 - alternatywne, 384
- typu asset, 416, 430, 432
- typu non-asset, 433
- typu plural, 300

zastosowanie

- ciągów formatujących, 325
- etykiet, 400
- fragmentów, 157
- klasy abstrakcyjnej, 194
- kontenera RecyclerView, 201
- niejawnych intencji, 326
- pułapek, 111
- usług, 582

zatrzymywanie aktywności, 88

zdarzenia

- lokalne, 622
- związane z dotykiem, 648

zdjęcia, 339

- miniaturki, 351
- określanie lokalizacji, 344
- w pełnym rozmiarze, 351

zgoda na użycie uprawnień, 690

zmiana

- koloru tła, 471
- orientacji urządzenia, 90, 95, 450, 638
- rozstawu przycisków, 478

znacznik <layout>, 413

zwracanie

- danych do fragmentu, 270
- intencji, 135
- listy, 316

Ż

żądanie

- LocationRequest, 686
- udostępnienia lokalizacji, 685
- udzielenia uprawnień, 688

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



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

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

GRUPA WYDAWNICZA

 **Helion SA**

Android — zaprogramuj przyszłość!

Android jest systemem stworzonym na urządzenia mobilne. Pod jego kontrolą pracują smartfony, tablety, lodówki, telewizory i wszystko wskazuje na to, że internet rzeczy będzie również w dużej części androidowy. Wielu programistów widzi w tym niespotykane dotąd możliwości, jednak pisanie aplikacji na Androida wcale nie jest łatwym zadaniem. Biegłe poruszanie się po tym środowisku wymaga opanowania licznych nowych kompetencji i technik.

Jeśli umiesz pisać zorientowany obiektowo kod w Javie i postanowiłeś zacząć tworzyć aplikacje dla Androida, wzięteś do ręki odpowiednią książkę. Ten przystępnie napisany przewodnik przeprowadzi Cię przez trudności, jakie napotyka właściwie każdy początkujący programista aplikacji na Androida. Opisano tu zagadnienia związane z tworzeniem projektu i używaniem aktywności, układów oraz intencji jawnych i niejawnych. Pokazano, jak korzystać z fragmentów, tworzyć menu i jak pracować na plikach multimedialnych. Przedstawiono również szczegóły projektowania i rozwijania aplikacji mobilnej.

W tej książce:

- środowisko pracy i zasady projektowania aplikacji
- obsługa wbudowanego aparatu fotograficznego i dotykowości
- architektura MVVM i mechanizmy wiązania danych
- testowanie aplikacji
- animacje w Androidzie
- usługa lokalizacji i korzystanie z map

Autorzy są ekspertami i instruktorami z firmy **Big Nerd Ranch**, która specjalizuje się w rozwijaniu innowacyjnych aplikacji mobilnych.



Bill Phillips

Jest współtwórcą i instruktorem serii szkoleń Big Nerd Ranch Android Bootcamp. Jest również wykładowcą, prowadzi bloga, lubi dobrą literaturę, komponuje muzykę i pisze książki.



Chris Stewart

Jest kierownikiem zespołu do spraw Androida. Zawsze dąży do perfekcji w tym, co robi. W wolnym czasie chodzi po górach i podróżuje.



Kristin Marsicano

Jest deweloperem aplikacji dla systemu Android. Pasjonuje się nauką i rozwijaniem oprogramowania. W wolnych chwilach gotuje, zajmuje się jogą lub po prostu uczy się czegoś nowego.

Helion

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

informatyka w najlepszym wydaniu

Helion SA
ul. Kościuszki 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>



**Big Nerd
Ranch**

ISBN 978-83-283-3636-0



cena: 99,00 zł

sięgnij po **WIĘCEJ**



KOD KORZYŚCI