

Rusz głową!

Android

Programowanie aplikacji



Zobacz
jak wyglądają
fragmenty
pod mikroskopem



Unikaj
kłopotliwych
aktywności

Przekonaj się
jak układy
z ograniczeniami
mogą zmienić
Twoje życie



Twórz
usługi
nie z tego świata



Znajdź
własną drogę
dzięki usługom
lokalizacyjnym
Androida



Zabaw się
z biblioteką
wsparcia
wzornictwa



Tytuł oryginału: Head First Android Development: A Brain-Friendly Guide, 2nd Edition

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-4079-4

© 2018 Helion SA

Authorized Polish translation of the English edition of Head First Android Development 2e
ISBN 9781491974056 © 2017 David Griffiths and Dawn Griffiths

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

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

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

Autor oraz Helion SA 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 Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion SA

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/andrr2>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści (skrótowy)

	Wprowadzenie	xxix
1	Zaczynamy: <i>Skok na głęboką wodę</i>	1
2	Tworzenie interaktywnych aplikacji: <i>Aplikacje, które coś robią</i>	37
3	Wiele aktywności i intencji: <i>Jakie są Twoje intencje?</i>	77
4	Cykl życia aktywności: <i>Była sobie aktywność</i>	119
5	Widoki i grupy widoków: <i>Podziwiał widoki</i>	169
6	Układy z ograniczeniami: <i>Rozmieszczaj rzeczy w odpowiednich miejscach</i>	221
7	Widoki list i adaptery: <i>Zorganizuj się</i>	247
8	Biblioteki wsparcia i paski aplikacji: <i>Na skrót</i>	289
9	Fragmenty: <i>Zadbaj o modularyzację</i>	339
10	Fragmenty dla większych interfejsów: <i>Różne wielkości, różne interfejsy</i>	393
11	Fragmenty dynamiczne: <i>Zagnieżdżanie fragmentów</i>	433
12	Biblioteka wsparcia wzornictwa: <i>Przeciwnięcie w prawo</i>	481
13	Widoki RecyclerView i CardView: <i>Stosuj recykling</i>	537
14	Szuflady nawigacyjne: <i>Z miejsca na miejsce</i>	579
15	Bazy danych SQLite: <i>Odpal bazę danych</i>	621
16	Proste kursory: <i>Pobieranie danych</i>	657
17	Kursory i zadania asynchroniczne: <i>Pozostając w tle</i>	693
18	Usługi uruchomione: <i>Do usług</i>	739
19	Usługi powiązane i uprawnienia: <i>Powiązane ze sobą</i>	767
A	Układy względne i układy siatki: <i>Poznaj krewnych</i>	817
B	Gradle: <i>Program do budowy Gradle</i>	833
C	ART: <i>Środowisko uruchomieniowe Androida</i>	841
D	ADB: <i>Android Debug Bridge</i>	849
E	Emulator: <i>Przyspieszanie emulatora</i>	857
F	Pozostałości: <i>Dziesięć najważniejszych zagadnień (których nie opisaliśmy)</i>	861

Spis treści (z prawdziwego zdarzenia)

W

Wprowadzenie

Twój mózg jest nastawiony na Androida.

Jesteś *tu* po to, by się czegoś *nauczyć*, a Twój *mózg* robi Ci przysługę, upewniając się, że to, czego się nauczyłeś, szybko *wyleci* z pamięci. Twój mózg myśli sobie: „Lepiej zostawić miejsce na coś ważnego, na przykład na zastanowienie się nad tym, których dzikich zwierząt lepiej unikać albo czy jeżdżenie nago na snowboardzie to dobry pomysł”. A zatem w jaki sposób możesz skłonić swój mózg, by myślał, że Twoje życie zależy od umiejętności pisania aplikacji na Androida?

Autorzy książki <i>Head First Android Development</i>	iv
Dla kogo jest ta książka?	xxx
Wiemy, co sobie myślisz	xxxi
Wiemy, co sobie myśli Twój mózg	xxxi
Metapoznanie — myślenie o myśleniu	xxxiv
Oto co MY zrobiliśmy	xxxvi
Przeczytaj to	xxxviii
Zespół recenzentów technicznych	xxxix
Podziękowania	xl

Zastanawiam się,
jak zmusić mózg
do zapamiętania
tych informacji...



Zaczynamy

1

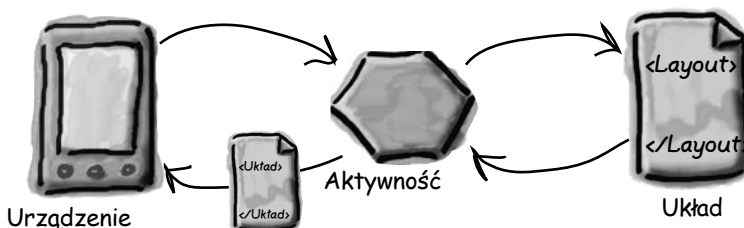
Skok na głęboką wodę

Android błyskawicznie podbił świat.

Każdy chce mieć smartfon lub tablet, a urządzenia z Androidem są niezwykle popularne. W tej książce nauczymy Cię, jak **pisać własne aplikacje**, a zaczniemy od pokazania procesu przygotowania bardzo prostej aplikacji i uruchomienia jej na wirtualnym urządzeniu z Androidem. W trakcie tych prac poznasz także kilka podstawowych komponentów wszystkich aplikacji na Androida, takich jak **aktywności** i **układy**. **Jedyną rzeczą, której będziesz do tego potrzebować, jest znajomość Javy, choć wcale nie musisz być w niej mistrzem...**



Witamy w Androidowie	2
Platforma Android w szczegółach	3
Oto co mamy zamiar zrobić	4
Środowisko programistyczne	5
Zainstaluj Android Studio	6
Stwórzmy prostą aplikację	7
Jak stworzyć aplikację?	8
Aktywności i układy z wysokości 15 tysięcy metrów	12
Jak stworzyć aplikację? (ciąg dalszy)	13
Właśnie utworzyłeś swoją pierwszą aplikację na Androida	15
Android Studio utworzy pełną strukturę katalogów aplikacji	16
Przydatne pliki projektu	17
Edycja kodu z użyciem edytorów Android Studio	18
Uruchamianie aplikacji w emulatorze Androida	23
Tworzenie wirtualnego urządzenia z Androidem	24
Uruchamianie aplikacji w emulatorze	27
Postępy możesz obserwować w konsoli	28
Ale co się właściwie stało?	30
Usprawnienie aplikacji	31
Czym jest układ?	32
Plik activity_main.xml zawiera dwa elementy	33
Aktualizacja tekstu wyświetlanego w układzie	34
Weź aplikację na jazdę próbną	35
Twój przyborek do Androida	36



Tworzenie interaktywnych aplikacji

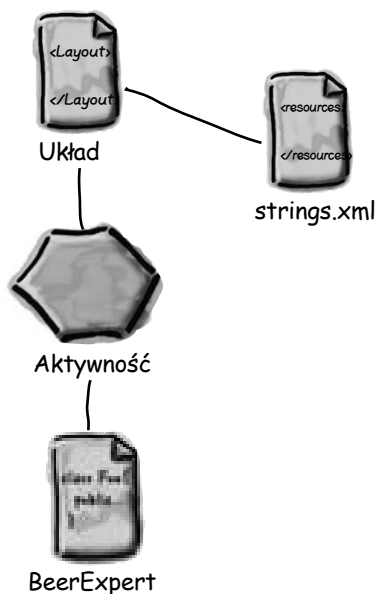
2

Aplikacje, które coś robią

Większość aplikacji musi w jakiś sposób reagować na poczynania użytkowników.

Z tego rozdziału dowiesz się, co zrobić, aby Twoje aplikacje były nieco bardziej interaktywne.

Przekonasz się, jak zmusić aplikację, by coś **zrobiła** w odpowiedzi na działania użytkownika, oraz jak sprawić, by **aktywności i układy porozumiewały się ze sobą** jak starzy kumple. Przy okazji pokażemy Ci **nieco dokładniej, jak naprawdę działa Android** — poznasz plik **R**, czyli ukryty klejnot, który spaja pozostałe elementy aplikacji.



W tym rozdziale napiszemy aplikację Doradca piwny	38
Utworzenie projektu	40
Utworzyliśmy domyślną aktywność i układ	41
Dokładniejsza prezentacja edytora projektu	42
Dodawanie przycisku w edytorze projektu	43
Plik <code>activity_find_beer.xml</code> zawiera nowy przycisk	44
Dokładniejszy przegląd kodu układu	45
Weź swoją aplikację na jazdę próbną	49
Podawanie tekstów na stałe utrudnia lokalizację	50
Utworzenie zasobu łańcuchowego	51
Zastosowanie zasobu łańcuchowego w układzie	52
Kod pliku <code>activity_find_beer.xml</code>	53
Dodawanie wartości do komponentu Spinner	56
Dodanie elementu <code>string-array</code> do pliku <code>strings.xml</code>	57
Jazda próbna komponentu Spinner	58
Musimy zadbać o to, by przycisk coś robił	59
Niech przycisk wywołuje metodę	60
Jak wygląda kod aktywności	61
Dodaj do aktywności metodę <code>onClickFindBeer()</code>	62
Metoda <code>onClickFindBeer()</code> musi coś robić	63
Dysponując obiektem <code>View</code> , można odwoływać się do jego metod	64
Aktualizacja kodu aktywności	65
Pierwsza wersja aktywności	67
Co ten kod robi?	68
Tworzenie własnej klasy Javy	70
Co się dzieje podczas wykonywania tego kodu?	74
Jazda próbna — test aplikacji	75
Twój przyborek do Androida	76

Wiele aktywności i intencji

3

Jakie są Twoje intencje?

Większość aplikacji potrzebuje więcej niż jednej aktywności.

Dotychczas mieliśmy do czynienia z aplikacjami składającymi się tylko z jednej aktywności. Kiedy jednak sprawy się komplikują, jedna aktywność zwyczajnie nie wystarczy. Dlatego w tym rozdziale pokażemy Ci, jak **tworzyć aplikacje składające się z wielu aktywności** i jak nasze aplikacje mogą porozumiewać się z innymi, wykorzystując **intencje**. Pokażemy także, jak można używać intencji, by **wykraczać poza granice naszych aplikacji**, i jak wykorzystywać **aktywności należące do innych aplikacji dostępnych w urządzeniu do wykonywania akcji**. To wszystko zapewni nam znacznie większe możliwości.

Intencja



Do: InnaAktywnosc

Aplikacja może zawierać więcej niż jedną aktywność	78
Oto struktura naszej aplikacji	79
Zaczynamy: utworzenie projektu	79
Aktualizacja układu	80
Utworzenie drugiej aktywności i układu	82
Przedstawiamy plik manifestu aplikacji na Androida	84
Intencja jest rodzajem komunikatu	86
Co się dzieje po uruchomieniu aplikacji?	88
Przekazanie tekstu do drugiej aktywności	90
Aktualizacja właściwości widoku tekstowego	91
Metoda putExtra() zapisuje w intencji dodatkowe informacje	92
Aktualizacja kodu aktywności CreateMessageActivity	95
Zastosowanie informacji przekazanych w intencji w klasie ReceiveMessageActivity	96
Co się dzieje, gdy użytkownik kliknie przycisk Wyślij wiadomość	97
Możemy zmienić aplikację tak, by wiadomość była wysyłana do innych osób	98
Jak działają aplikacje na Androida	99
Utworzenie intencji określającej akcję	101
Zmiana intencji w celu użycia akcji	102
Jak Android korzysta z filtrów intencji?	106
A co, jeśli chcemy, by użytkownik ZAWSZE wybierał aktywność?	112
Co się dzieje w momencie wywołania metody createChooser()?	113
Zmień kod, by wyświetlać okno dialogowe	115
Twój przybornik do Androida	118

Hej, użytkowniku!
Powiedz mi, proszę,
której aktywności chciałbyś
użyć tym razem.



CreateMessageActivity



Android



Użytkownik

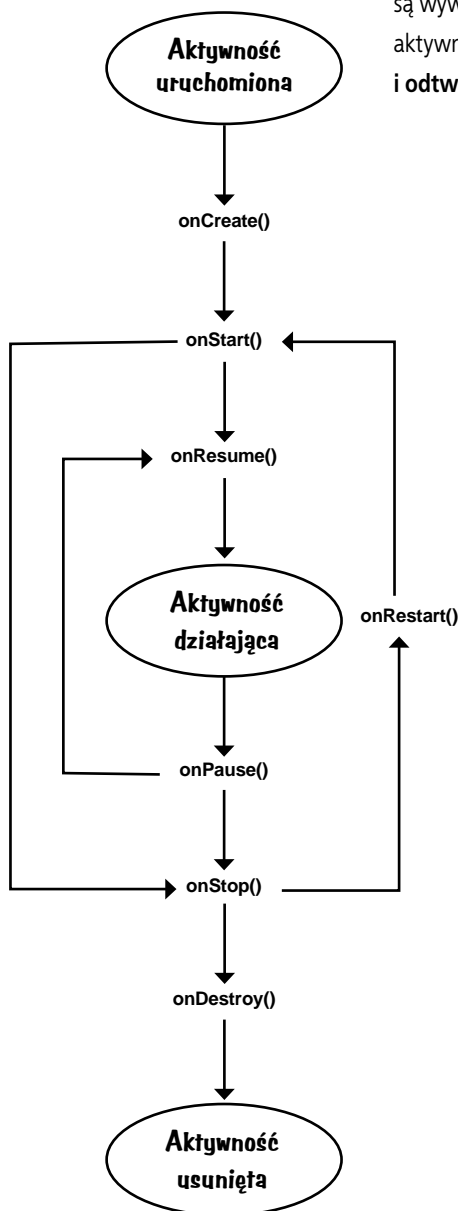
4

Cykl życia aktywności

Była sobie aktywność

Aktywności stanowią podstawę wszystkich aplikacji na Androida.

Wiesz już, jak tworzyć aktywności i jak sprawić, by jedna aktywność uruchomiła drugą, używając intencji. Ale *co tak naprawdę dzieje się za kulisami?* W tym rozdziale nieco dokładniej opiszemy **cykl życia aktywności**. Co się dzieje, kiedy aktywność **jest tworzona i usuwana**? Jakie metody są wywoływane, gdy aktywność jest **wyświetlana i pojawia się na ekranie**, a jakie, gdy aktywność **traci miejsce wprowadzania i jest ukrywana**? W jaki sposób można **zapisywać i odtwarzać stan aktywności**? Przeczytaj, by się dowiedzieć.



Jak właściwie działają aktywności?	120
Aplikacja stopera	122
Dodanie zasobów łańcuchowych	123
Jak będzie działał kod aktywności?	125
Działanie kodu obsługującego przyciski	126
Metoda runTimer()	127
Pełny kod metody runTimer()	129
Kompletny kod aktywności StopwatchActivity	130
Obrót ekranu zmienia konfigurację urządzenia	136
Stany aktywności	137
Cykl życia aktywności: od utworzenia do usunięcia	138
Zaktualizowany kod aktywności StopwatchActivity	142
Co się stanie po uruchomieniu aplikacji?	143
Tworzenie i usuwanie to nie cały cykl życia aktywności	146
Zaktualizowany kod aktywności StopwatchActivity	151
Co się dzieje podczas działania aplikacji?	152
A co się dzieje, jeśli aplikacja jest tylko częściowo widoczna?	154
Cykl życia aktywności: życie na pierwszym planie	155
Zatrzymanie stopera w razie wstrzymania aktywności	158
Implementacja metod onPause() oraz onResume()	159
Kompletny kod aktywności	160
Co się stanie po uruchomieniu aplikacji?	163
Wygodny przewodnik po metodach cyklu życia aktywności	167
Twój przybornik do Androida	168

Widoki i grupy widoków

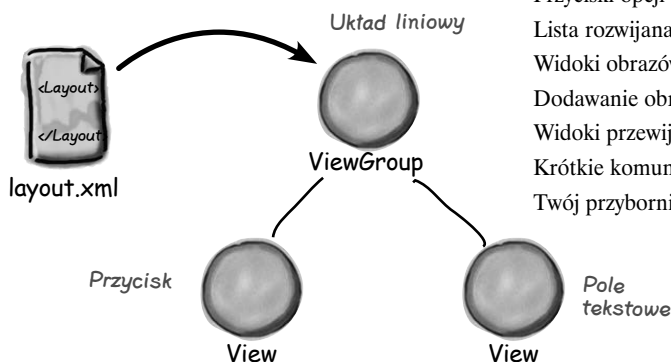
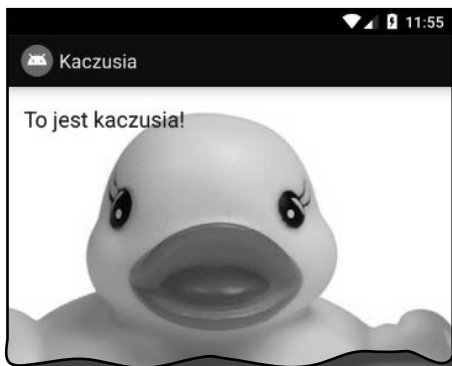
5

Podziwiał widoki

Zobaczyłeś już, jak można rozmieszczać elementy GUI, używając układu `LinearLayout`, ale to jedynie wierzchołek góry logowej.

W tym rozdziale **nico dokładniej** opiszemy *rzeczywisty sposób działania* układu liniowego. Przedstawimy także `FrameLayout`, nazywany również **układem ramki**, będący prostym układem używanym do rozmieszczania widoków jeden na drugim, i zabierzemy Cię na wycieczkę po **najważniejszych komponentach GUI** i **sposobach ich stosowania**. Pod koniec tego rozdziału przekonasz się, że choć wszystkie te układy i komponenty wyglądają nieco inaczej, to jednak **mają ze sobą więcej wspólnego, niż można by przypuszczać**.

Układy `FrameLayout` pozwalają nakładać jedno widoki na inne. To bardzo przydatne w przypadkach, gdy chcemy stworzyć interfejs użytkownika, w którym na przykład tekst jest wyświetlany na tle obrazu.



Interfejs użytkownika aplikacji składa się z układów i komponentów GUI	170
Układ <code>LinearLayout</code> wyświetla widoki w jednym wierszu lub w jednej kolumnie	171
Dodawanie pliku zasobów wymiaru w celu zapewnienia spójnych wypełnień w układach	174
Stosowanie marginesów do oddalania widoków od siebie	176
Zmieńmy nieco prosty układ liniowy	177
Rozciągaaaaamy widok, zwiększając jego wagę	179
Wartości atrybutu <code>android:gravity</code>	183
Kompletny układ liniowy	186
Układy <code>FrameLayout</code> rozmieszczają widoki jeden na drugim	188
Dodanie obrazka do projektu	189
Kompletny kod układu	192
Układy <code>FrameLayout</code> : podsumownie	193
Zabawy z widokami	201
Pola tekstowe	202
Przycisk	203
Przycisk przełącznika	204
Przełącznik	205
Pola wyboru	206
Przyciski opcji	208
Lista rozwijana	210
Widoki obrazów	211
Dodawanie obrazów do przycisków	213
Widoki przewijane	215
Krótkie komunikaty	216
Twój przybornik do Androida	220

Układy z ograniczeniami

6 Rozmieszczaj rzeczy w odpowiednich miejscach

Spójrzmy prawdzie w oczy: musisz wiedzieć, jak tworzyć piękne układy.

Jeśli piszesz aplikacje i chcesz, by inni ich używali, to musisz mieć pewność, że będą one **wyglądać dokładnie tak, jak sobie tego życzysz**. Jak na razie dowiedziałeś się jedynie, jak używać układów `LinearLayout` oraz `RelativeLayout` oraz `FrameLayout`, ale co zrobić, jeśli *projekt interfejsu aplikacji będzie bardziej złożony*? Aby pokazać Ci jak należy sobie radzić w takich sytuacjach, przedstawimy nowy układ dodany do Androida — układ z ograniczeniami (`ConstraintLayout`) — używany do **wizualnego konstruowania układów na podstawie szkicu graficznego**. Pokażemy Ci także, jak **ograniczenia** pozwalają na określenie położenia widoków, w *sposób niezależny od wielkości i orientacji ekranu*. I w końcu dowiesz się też, jak oszczędzać czas, korzystając z możliwości **automatycznego przewidywania i dodawania ograniczeń**, jaką dysponuje Android Studio.



Zagnieżdżone układy mogą być nieefektywne	222
Przedstawiamy układy z ograniczeniami	223
Nie zapomnij dołączyć do projektu biblioteki Constrained Layout Library	224
Dodanie zasobów do strings.xml	225
Zastosowanie narzędzia do tworzenia szkicu	226
Rozmieszczanie widoków przy wykorzystaniu ograniczeń	227
Dodawanie ograniczenia w pionie	228
Zmiany szkicu są uwzględniane w kodzie XML	229
Jak wyśrodkowywać widoki	230
Zmiana położenia widoku poprzez określenie przesunięcia	231
Jak zmieniać wielkość widoku?	232
Jak wyrównywać widoki?	238
Stwórzmy prawdziwy układ	239
Zacznij od dodania widoków do górnego wiersza	240
Mechanizm wnioskowania odgaduje, jakie ograniczenia należy dodać	241
Dodaj do szkicu kolejny wiersz...	242
I w końcu dodaj widok na treść wiadomości	243
Jazda próbna aplikacji	244
Twój przyborek do Androida	245

Widoki list i adaptery

7

Zorganizuj się

Chcesz wiedzieć, jaki jest najlepszy sposób na określenie struktury aplikacji?

Znasz już podstawowe elementy konstrukcyjne używane do tworzenia aplikacji, więc teraz nadszedł czas, **żebyś się lepiej zorganizował**. W tym rozdziale pokażemy Ci, jak możesz **przekształcić** zbiór pomysłów w **niesamowitą aplikację**. Zobaczysz, że **listy danych** mogą stać się kluczowym elementem projektu aplikacji i że **łączenie ich** może prowadzić do powstania aplikacji **łatwej w użyciu i zapewniającej ogromne możliwości**. Przy okazji zapoznasz się z **obiektami nasłuchującymi** i **adapterami**, dzięki którym Twoja aplikacja stanie się bardziej dynamiczna.

Ekran początkowy z listą opcji

Lista oferowanych napojów

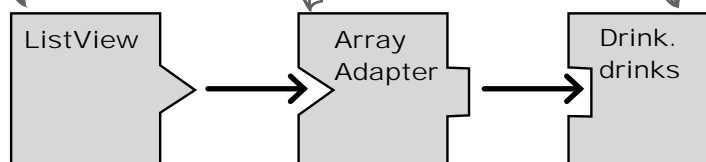
Szczegółowe informacje o każdym napoju

Każda aplikacja zaczyna się od pomysłu	248
Użyj widoku listy do nawigowania po danych	251
Aktywność szczegółów napoju	253
Struktura aplikacji dla kafeтерии Coffeina	254
Klasa Drink	256
Układ aktywności głównego poziomu składa się z obrazka i listy	258
Kompletny kod układu aktywności głównego poziomu	260
Zapewnianie reakcji ListView na kliknięcia za pomocą obiektu nasłuchującego	261
Dodanie obiektu nasłuchującego do widoku listy	262
Aktywność kategorii wyświetla dane jednej kategorii	267
Aktualizacja układu activity_drink_category.xml	268
W przypadku danych statycznych należy użyć adaptera	269
Łączenie widoków ListView z tablicami przy użyciu adaptera	270
Dodanie adaptera ArrayAdapter do aktywności DrinkCategoryActivity	271
Przegląd aplikacji, czyli dokąd dotarliśmy	274
Jak obsługiwaliśmy kliknięcia w aktywności TopLevelActivity	276
Kompletny kod aktywności DrinkCategoryActivity	278
Wypełnienie widoków danymi	281
Kod aktywności DrinkActivity	283
Co się stanie po uruchomieniu aplikacji	284
Twój przybornik do Androida	288

To jest nasz widok listy.

Utworzymy adapter ArrayAdapter, aby powiązać widok listy z naszą tablicą.

To jest nasza tablica.



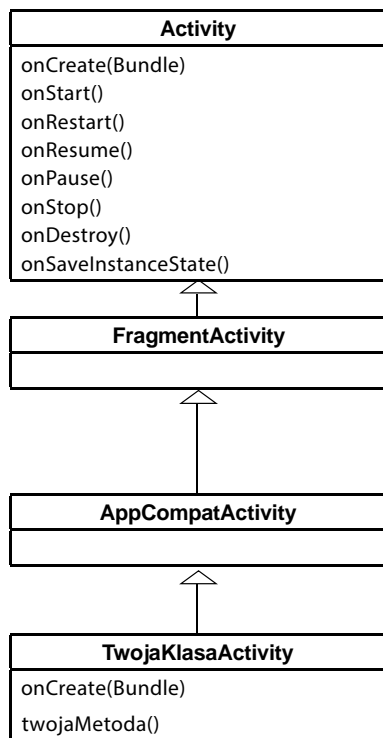
Biblioteki wsparcia i paski aplikacji

Na skróty

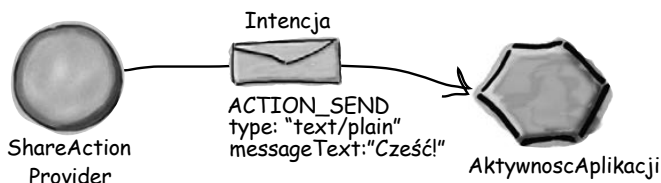
8

Każdy lubi chodzić na skróty.

Z tego rozdziału dowiesz się, jak korzystając z **pasków aplikacji**, wzbogacić aplikację o możliwość chodzenia na skróty. Pokażemy Ci, jak uruchamiać inne aktywności za pomocą *elementów akcji* dodawanych do pasków aplikacji, jak udostępniać treści innym aplikacjom, używając *dostawcy akcji współdzielenia*, oraz jak poruszać się w górę hierarchii aplikacji za pomocą *przycisku W górę* umieszczonego na pasku aplikacji. Jednocześnie przedstawimy Ci potężne **biblioteki wsparcia Androida**, mające kluczowe znaczenie dla zapewniania nowoczesnego wyglądu aplikacji na starszych wersjach systemu.



Świetne aplikacje mają przejrzystą strukturę	290
Różne typy nawigacji	291
Zacznijmy od paska akcji	293
Utwórz aplikację Włoskie Co Nieco	295
Dodaj bibliotekę wsparcia AppCompatActivity v7	296
Plik AndroidManifest.xml może zmieniać postać paska aplikacji	299
Jak zastosować motyw?	300
Zdefiniuj styl w pliku zasobów	301
Dostosuj wygląd aplikacji	303
Zdefiniuj kolory w pliku zasobów kolorów	304
Kod pliku activity_main.xml	305
Pasek aplikacji a pasek narzędzi	306
Dołącz pasek narzędzi do układu aktywności	312
Dodawanie akcji do paska aplikacji	315
Zmień pasek aplikacji, dodając do niego etykietę	318
Kod pliku AndroidManifest.xml	319
Określ wygląd akcji	322
Kompletny kod pliku MainActivity.java	325
Włączanie nawigacji w górę	327
Dzielenie się treściami z poziomu paska aplikacji	331
Dodawanie dostawcy akcji udostępniania do menu_main.xml	332
Określanie treści za pomocą intencji	333
Kompletny kod aktywności MainActivity	334
Twój przyborek do Androida	337



Fragmenty

9

Zadbaj o modularyzację

Wiesz już, jak tworzyć aplikację, które działają tak samo niezależnie od tego, na jakim urządzeniu zostały uruchomione...

...ale co zrobić w przypadku, kiedy akurat chcesz, by aplikacja **wyglądała i działała inaczej** w zależności od tego, czy zostanie uruchomiona na *telefonie*, czy na *tablecie*? W tej sytuacji będziesz potrzebował **fragmentów**, modularnych komponentów, które mogą być **wielokrotnie używane w różnych aktywnościach**. Pokażemy, jak można tworzyć **proste fragmenty** oraz **fragmenty list**, jak **dodawać fragmenty do aktywności** oraz w jaki sposób zapewniać wzajemną **komunikację** pomiędzy fragmentami i aktywnościami.



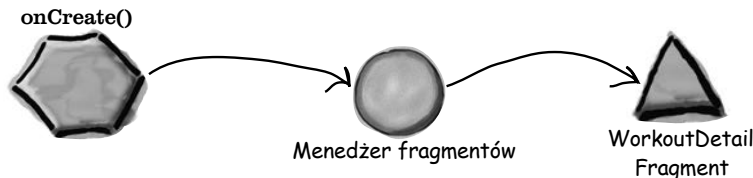
activity_detail.xml

Fragment listy dysponuje swoim własnym widokiem listy, dzięki czemu nie musimy sami tworzyć takiej listy. Wystarczy, że dostarczymy niezbędnych danych.



Rozciąganie kończyn
Ogólna agonia
Tylko dla mięczaków
Siła i dystans

Twoja aplikacja musi wyglądać świetnie na WSZYSTKICH urządzeniach	340
Może się zdarzyć, że aplikacja będzie musiała także działać inaczej	341
Fragmenty umożliwiają wielokrotne stosowanie kodu	342
Aplikacja w wersji na telefony	343
Utworzenie projektu i aktywności	345
Dodanie przycisku do układu aktywności	346
Jak dodać fragment do projektu?	348
Metoda onCreateView() fragmentu	350
Dodawanie fragmentu do układu aktywności	352
Zapewnienie interakcji fragmentu i aktywności	359
Klasa Workout	360
Przekazywanie identyfikatora treningu do fragmentu	361
Określenie identyfikatora treningu w kodzie aktywności	363
Cykl życia fragmentów	365
Określenie zawartości widoków w metodzie onStart() fragmentu	367
Jak utworzyć fragment typu ListFragment?	374
Zaktualizowany kod klasy WorkoutListFragment	377
Kod układu activity_main.xml	381
Powiązanie listy z widokiem szczegółów	384
Kod pliku WorkoutListFragment.java	387
Aktywność MainActivity musi implementować interfejs	388
Aktywność DetailActivity musi przekazać identyfikator do fragmentu WorkoutDetailFragment	389
Twój przybornik do Androida	392



10

Fragmenty dla większych interfejsów

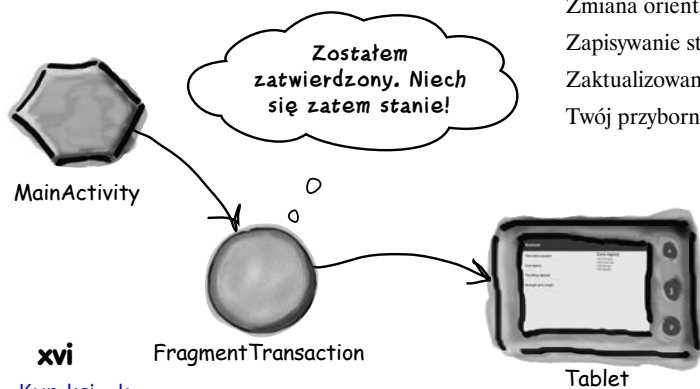
Różne wielkości, różne interfejsy

Jak na razie uruchamialiśmy nasze aplikacje wyłącznie na urządzeniach z małymi ekranami.

A co, jeśli użytkownicy będą mieli tablety? W tym rozdziale dowiesz się, w jaki sposób, dzięki zróżnicowaniu wyglądu i sposobu działania aplikacji w zależności od urządzenia, na jakim została ona uruchomiona, można projektować **elastyczne interfejsy użytkownika**. Pokażemy także, jak kontrolować działanie aplikacji po naciśnięciu przycisku *Wstecz*, prezentując przy okazji dwa nowe rozwiązania: **stos cofnięć** i **transakcje fragmentów**. I w końcu dowiesz się też, jak można **zapisywać i odtwarzać** stan fragmentów.



Nasza aplikacja Trener wygląda tak samo na telefonie i tablecie	394
Projektowanie z myślą o większych interfejsach	395
Wersja aplikacji na telefony	396
Wersja aplikacji na tablety	397
Utwórz AVD tabletu	399
Umieszczaj zasoby przeznaczone dla różnych rodzajów ekranów w odpowiednich katalogach	402
Różne opcje katalogów	403
Tablety używają układów zapisanych w katalogu layout-large	408
Jak działa zaktualizowany kod?	410
Musimy zmienić kod metody itemClicked()	412
Chcemy, by fragmenty współpracowały z przyciskiem Wstecz	413
Witamy stos cofnięć	414
Transakcje na stosie cofnięć nie muszą być aktywnościami	415
Użyj układu FrameLayout, by programowo zmieniać fragmenty	416
Skorzystaj z różnic w układach, aby określić, który z nich został użyty	417
Zmodyfikowany kod aktywności MainActivity	418
Stosowanie transakcji fragmentów	419
Zaktualizowany kod aktywności MainActivity	423
Zmiana orientacji tabletu wywołuje problem w aplikacji	427
Zapisywanie stanu aktywności (po raz wtóry)	428
Zaktualizowany kod pliku WorkoutDetailFragment.java	430
Twój przyborek do Androida	432



11

Fragmenty dynamiczne

Zagnieżdżanie fragmentów

Jak na razie dowiedziałeś się, jak można tworzyć i stosować fragmenty statyczne.

Ale co zrobić, jeśli zechcemy, by nasze fragmenty były nieco bardziej **dynamiczne**? Fragmenty dynamiczne mają wiele wspólnego z aktywnościami dynamicznymi, choć występują pomiędzy nimi pewne kluczowe różnice, z którymi będziemy musieli umieć odpowiednio postępować. W tym rozdziale dowiesz się, jak **przekształcać aktywności dynamiczne na działające fragmenty dynamiczne**. Dowiesz się także, jak korzystać z **transakcji fragmentów** w celu **zachowania stanu fragmentów**. I w końcu odkryjesz, jak można **zagnieżdżać jedne fragmenty w innych** oraz w jaki sposób **menedżer fragmentów podrzędnych** ułatwia zachowanie kontroli nad niesformym stosem cofnięć.

Zawsze, gdy widzę atrybut `android:onClick`, zakładam, że chodzi o mnie. To moje metody mają zostać wywołane, a nie metody fragmentu.



Aktywność

Dodawanie fragmentów dynamicznych	434
Nowa wersja aplikacji	436
Utwórz aktywność TempActivity	437
Klasa TempActivity musi dziedziczyć po AppCompatActivity	438
Kod fragmentu StopwatchFragment	444
Układ fragmentu StopwatchFragment	447
Dodanie fragmentu StopwatchFragment do układu aktywności TempActivity	449
Atrybut <code>onClick</code> wywołuje metody aktywności, a nie fragmentu	452
Powiązanie obiektu nasłuchującego <code>OnClickListener</code> z przyciskami	457
Kod fragmentu StopwatchFragment	458
Obrócenie urządzenia zeruje stoper	462
Używaj <code><fragment></code> dla statycznych fragmentów...	463
W układzie <code>activity_temp.xml</code> zastosuj układ <code>FrameLayout</code>	464
Kompletny kod aktywności <code>TempActivity.java</code>	467
Dodanie stopera do fragmentu <code>WorkoutDetailFragment</code>	469
Kompletny kod pliku <code>WorkoutDetailFragment.java</code>	476
Twój przybornik do Androida	480

Wyświetlam szczegółowe informacje o treningu, jak również stoper.

Transakcja wyświetlająca fragment `StopwatchFragment` zostaje zagnieżdżona wewnątrz transakcji, która wyświetla fragment `WorkoutDetailFragment`.



12

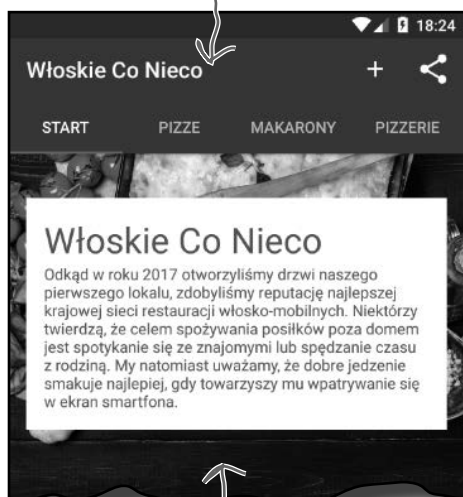
Biblioteka wsparcia wzornictwa

Przecignięcie w prawo

Czy kiedykolwiek zastanawiałeś się, jak napisać aplikację z bogatym i ładnym interfejsem użytkownika?

Dzięki udostępnieniu biblioteki wsparcia wzornictwa, **Android Design Support Library**, teraz znacznie łatwiej można już tworzyć aplikacje o intuicyjnym interfejsie użytkownika. W tym rozdziale przedstawimy najważniejsze informacje na ten temat. Dowiesz się, jak dodawać **karty**, dzięki którym użytkownicy będą mogli łatwiej poruszać się po aplikacji. Poznasz sposoby **animowania pasków narzędzi**, by na żądanie mogły się związać lub przewijać. Nauczysz się także dodawać do aplikacji **plywające przyciski akcji**, ułatwiające dostęp do najczęściej wykonywanych operacji. I w końcu przedstawimy Ci **snackbar** — mechanizm do wyświetlania krótkich, informacyjnych komunikatów, w którymi użytkownik może wchodzić w interakcje.

Zapewnimy, że pasek narzędzi będzie przewijany wraz z zawartością fragmentu `TopFragment`.



Do fragmentu `TopFragment` dodamy możliwość przewijania zawartości.

Aplikacja Włoskie Co Nieco w nowej odsłonie	482
Struktura aplikacji	483
Użycie klasy <code>ViewPager</code> do przewijania fragmentów	489
Dodajemy <code>ViewPager</code> do układu aktywności <code>MainActivity</code>	490
Przekaż kontrolce informacje o stronach przy użyciu odpowiedniego adaptera	491
Kod naszego adaptera <code>FragmentPagerAdapter</code>	492
Pełny kod pliku <code>MainActivity.java</code>	494
Dodanie kart do aktywności <code>MainActivity</code>	498
Jak dodać karty do układu?	499
Połączenie układu kart z kontrolką <code>ViewPager</code>	501
Pełny kod pliku <code>MainActivity.java</code>	502
Biblioteka wsparcia wzornictwa pomaga implementować <code>Material Design</code>	506
Zapewnienie reagowania paska narzędzi na przewijanie	508
Dodanie <code>CoordinatorLayout</code> do układu aktywności <code>MainActivity</code>	509
Jak koordynować przewijanie?	510
Dodanie do fragmentu zawartości do przewijania	512
Pełny kod pliku <code>fragment_top.xml</code>	515
Dodanie zwijanego paska narzędzi do aktywności <code>OrderActivity</code>	517
Jak stworzyć prosty zwiżany pasek narzędzi?	518
Jak dodać obrazek do zwiżanego paska narzędzi?	523
Zaktualizowany kod układu <code>activity_order.xml</code>	524
Przyciski FAB i paski <code>snackbar</code>	526
Zaktualizowany kod pliku <code>activity_order.xml</code>	528
Pełny kod pliku <code>OrderActivity.java</code>	533
Twój przybornik do Androida	535

13

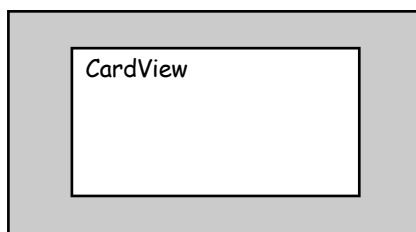
Widoki RecyclerView i CardView

Stosuj recykling

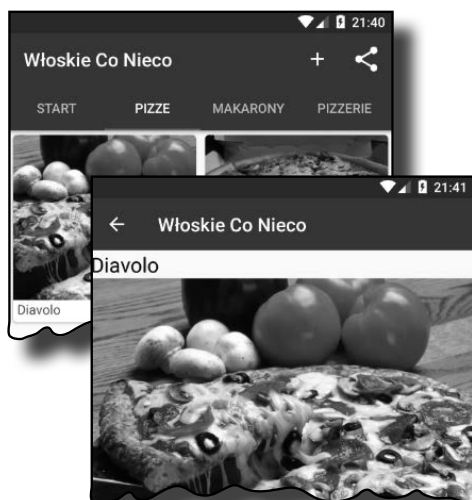
Przekonałeś się już, że kluczowym elementem większości aplikacji jest skromny widok listy.

Jednak w porównaniu z poznanymi wcześniej komponentami *Material Design* takie listy są bardzo proste. W tym rozdziale przedstawimy widok `RecyclerView` — bardziej zaawansowany typ widoku listy, który zapewnia *znacznie większą elastyczność* i idealnie pasuje do etosu wzornictwa *Material Design*. Nauczysz się tworzyć **adaptery** dostosowane do używanych danych i dowiesz się, jak całkowicie odmieniać wygląd list, używając do tego *jedynie dwóch wierszy kodu*. Pokażemy Ci także, jak korzystać z **widoków kart** (ang. *card views*) *przestrzenną prezentację* danych zgodną z wytycznymi *Material Design*.

ViewHolder



↑
Każdy z naszych obiektów ViewHolder będzie zawierać jeden widok CardView. Plik układu dla tych widoków CardView przygotowaliśmy już we wcześniejszej części rozdziału.



Wciąż jest wiele do zrobienia w aplikacji Włoskie Co Nieco	538
Widoki RecyclerView z wysokości 3000 metrów	539
Dodanie danych pizz	541
Wyświetlenie danych pizzy na karcie	542
Jak utworzyć widok karty?	543
Kompletny kod pliku <code>card_captioned_image.xml</code>	544
Dodanie adaptera widoku RecyclerView	546
Zdefiniowanie obiektu ViewHolder	548
Przesłonięcie metody <code>onCreateViewHolder()</code>	549
Dodanie danych do widoków CardView	550
Kompletny kod pliku <code>CaptionedImagesAdapter.java</code>	551
Utworzenie widoku RecyclerView	553
Dodanie widoku RecyclerView do układu fragmentu <code>PizzaFragment</code>	554
Kompletny kod pliku <code>PizzaFragment.java</code>	555
RecyclerView rozmieszcza swoje widoki, używając menedżera układu	556
Określanie menedżera układu	557
Pełny kod fragmentu <code>PizzaFragment.java</code>	558
Zapewnienie reakcji obiektu RecyclerView na kliknięcia	566
Utworzenie aktywności <code>PizzaDetailActivity</code>	567
Kod pliku <code>PizzaDetailActivity.java</code>	569
Zapewnienie reakcji widoku RecyclerView na kliknięcia	570
Można nasłuchiwać zdarzeń z widoków w adapterze	571
Zapewnianie możliwości wielokrotnego stosowania adapterów	572
Dodanie interfejsu do adaptera	573
Implementacja interfejsu we fragmencie <code>PizzaFragment</code>	575
Twój przyborek do Androida	578

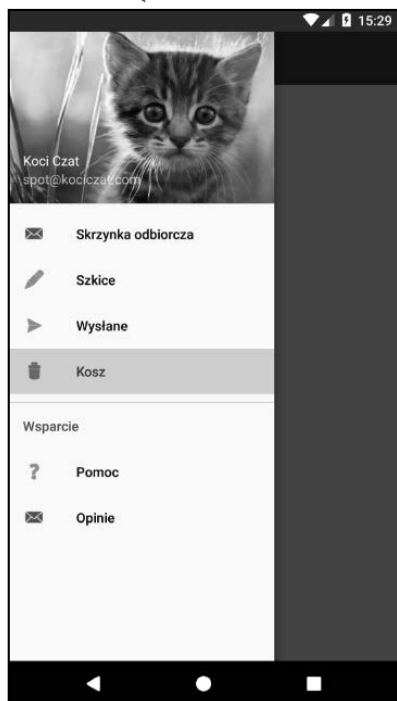
14

Szuflady nawigacyjne
Z miejsca na miejsce

Przekonałeś się już, w jaki sposób karty ułatwiają użytkownikom poruszanie się po aplikacji.

Jeśli jednak będziemy ich potrzebowali *bardzo dużo* lub jeśli *trzeba je rozdzielić na sekcje*, to **szuflada nawigacyjna** będzie Twoim nowym najlepszym przyjacielem. W tym rozdziale pokażemy Ci jak tworzyć szufladę nawigacyjną, która *wysuwa się z boku ekranu po jednym kliknięciu*. Dowiesz się, jak przygotować nagłówek takiej szuflady przy użyciu **widoku nawigacyjnego**, jak wypełnić ją **zestawem elementów o zadanej strukturze**, które pozwolą użytkownikom docierać do wszystkich głównych punktów aplikacji. I w końcu dowiesz się, jak przygotować **obiekt nasłuchujący widoku nawigacyjnego**, dzięki któremu szuflada będzie w stanie *reagować na najdelikatniejsze dotknięcia i przeciągnięcia*.

To jest aplikacja Koci Czat.



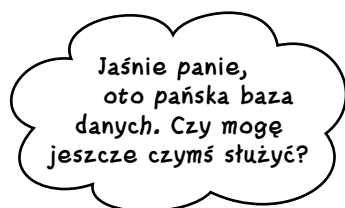
Widoki kart zapewniają łatwą nawigację...	580
Planujemy utworzenie szuflady nawigacyjnej w nowej aplikacji pocztowej	581
Szuflady nawigacyjne rozmontowane na czynniki pierwsze	582
Utworzenie projektu Koci Czat	584
Utworzenie fragmentu InboxFragment	585
Utworzenie fragmentu DraftsFragment	586
Utworzenie fragmentu SentItemsFragment	587
Utworzenie fragmentu TrashFragment	588
Przygotowanie układu paska narzędzi	589
Aktualizacja motywu aplikacji	590
Utworzenie aktywności HelpActivity	591
Utworzenie aktywności FeedbackActivity	592
Utworzenie nagłówka szuflady nawigacyjnej	594
Kompletny kod pliku nav_header.xml	595
Jak można grupować elementy?	598
Sekcję wsparcia dodamy jako podmenu	600
Kompletny kod pliku menu_nav.xml	601
Jak utworzyć szufladę nawigacyjną?	602
Kompletny kod układu aktywności activity_main.xml	603
Dodanie fragmentu InboxFragment do układu aktywności MainActivity	604
Dodanie przełącznika szuflady	607
Reagowanie na klikanie elementów szuflady	608
Implementacja metody onNavigationItemSelectedListener()	609
Zamknięcie szuflady po naciśnięciu przycisku Wstecz	614
Kompletny kod aktywności MainActivity	615
Twój przyborek do Androida	619

15

Bazy danych SQLite
Odpal bazę danych

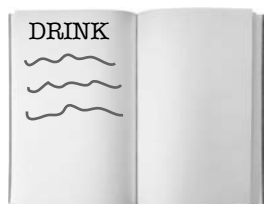
Jeśli rejestrujesz najlepsze wyniki lub przesyłane komunikaty, to Twoja aplikacja będzie musiała przechowywać dane.

A w Androidzie dane są zazwyczaj bezpiecznie i trwale przechowywane w **bazach danych SQLite**. W tym rozdziale pokażemy Ci, jak **utworzyć bazę danych, dodawać do niej tabele, wypełnić ją wstępnie danymi**, a wszystko to za pomocą **pomocnika SQLite**. Dowiesz się też, w jaki sposób można bezproblemowo przeprowadzać **aktualizacje** struktury bazy danych i jak w razie konieczności wycofania zmian wrócić do jej **wcześniejszych wersji**.



onCreate()

Pomocnik SQLite



Baza danych SQLite

Nazwa: Coffeina
Wersja: 1

Znowu w kafeterii Coffeina	622
Android trwale przechowuje dane, używając baz danych SQLite	623
Android udostępnia kilka klas związanych ze SQLite	624
Obecna struktura aplikacji kafeterii Coffeina	625
Zmienimy aplikację, by korzystała z bazy danych	626
Pomocnik SQLite zarządza Twoją bazą danych	627
Tworzenie pomocnika SQLite	628
Wnętrze bazy danych SQLite	630
Tabele tworzymy w języku SQL	631
Wstawianie danych za pomocą metody insert()	632
Wstawianie wielu rekordów	633
Kod klasy CoffeinaDatabaseHelper	634
Co robi kod pomocnika SQLite?	635
Co zrobić, gdy trzeba będzie zmienić bazę?	636
Bazy danych SQLite mają numer wersji	637
Co się dzieje w przypadku zmiany numeru wersji?	638
Aktualizacja bazy w metodzie onUpgrade()	640
Przywracanie starszej wersji bazy za pomocą metody onDowngrade()	641
Zaktualizujemy bazę danych	642
Aktualizacja istniejącej bazy danych	645
Aktualizacja rekordów za pomocą metody update()	646
Stosowanie warunków odnoszących się do wielu kolumn	647
Modyfikacja struktury bazy danych	649
Usuwanie tabeli	650
Pełny kod pomocnika SQLite	651
Twój przybornik do Androida	656

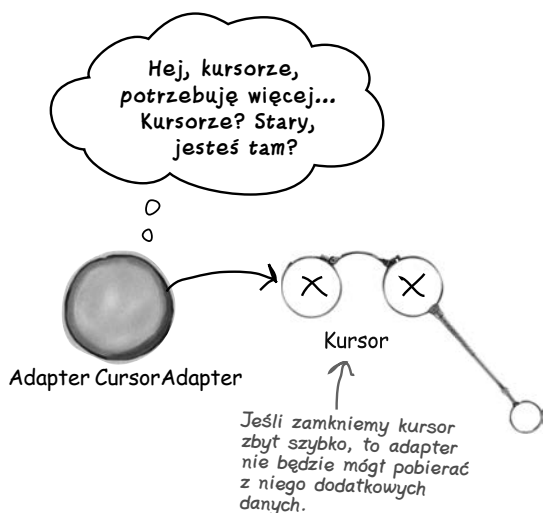
16

Proste kursory

Pobieranie danych

Jak łączysz swoje aplikacje z bazami danych SQLite?

Dotychczas dowiedziałeś się, jak tworzyć bazy danych, używając pomocnika SQLite. Kolejnym krokiem będzie uzyskanie dostępu do tych baz danych w aktywnościach. W tym rozdziale skoncentrujemy się na sposobach odczytywania danych bazy danych. Dowiesz się w nim, **jak używać kursora do odczytywania danych z bazy**. Nauczysz się także poruszać po kursorach i **uzyskiwać dostęp do umieszczonych w nich danych**. I w końcu dowiesz się, jak stosować **adaptery operujące na kursorach**, aby używać wspólnie kursorów i widoków list.



Co się wydarzyło wcześniej...	658
Struktura nowej wersji aplikacji kafeтерии Coffeina	659
Co zrobimy, by aktywność DrinkActivity zaczęła korzystać z bazy danych?	660
Aktualny kod aktywności DrinkActivity	661
Pobranie referencji do bazy danych	662
Pobieranie danych z bazy za pomocą kursora	663
Zwracanie wszystkich wierszy tabeli	664
Zwracanie wierszy w określonej kolejności	665
Zwracanie wybranych rekordów	666
Dotychczasowy kod aktywności DrinkActivity	669
Aby odczytać rekord z kursora, najpierw należy do niego przejść	670
Poruszanie się po kursorze	671
Pobieranie wartości z kursora	672
Kod aktywności DrinkActivity	673
Co udało się nam zrobić?	675
Aktualny kod aktywności DrinkCategoryActivity	677
Pobranie referencji do bazy danych kafeтерии...	678
Jak zastąpić tablicę przekazywaną do komponentu ListView?	679
SimpleCursorAdapter odwzorowuje dane na widoki	680
Stosowanie adaptera SimpleCursorAdapter	681
Zamykanie kursora i bazy danych	682
Ciąg dalszy opowieści	683
Zmodyfikowany kod aktywności DrinkCategoryActivity	688
Kod aktywności DrinkCategoryActivity (ciąg dalszy)	689
Twój przybornik do Androida	691

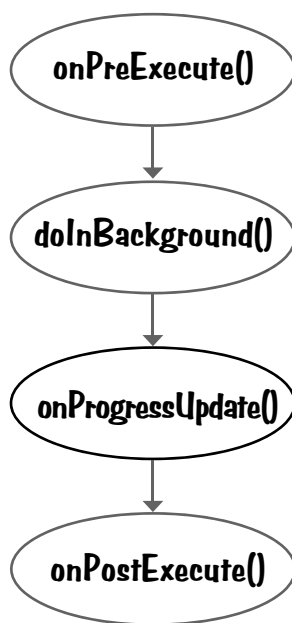
Kursory i zadania asynchroniczne

17

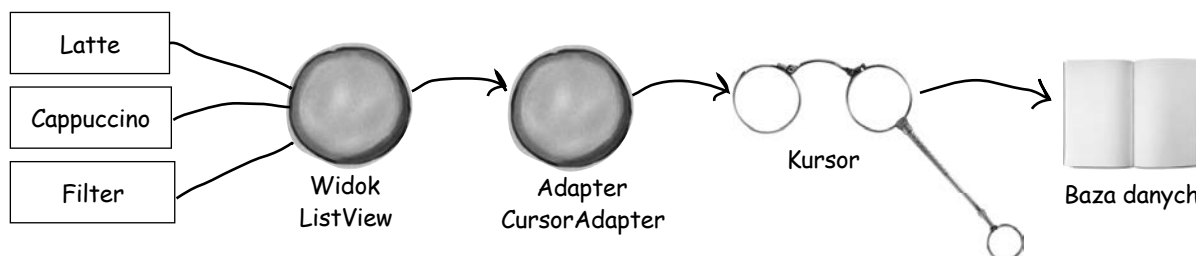
Pozostając w tle

Przeważająca większość aplikacji musi aktualizować swoje dane.

W poprzednim rozdziale dowiedziałeś się, jak pisać aplikacje, które odczytują dane z bazy danych SQLite. A co w przypadku, kiedy chcemy zaktualizować dane aplikacji? W tym rozdziale dowiesz się, jak sprawić, by aplikacja **reagowała na poczynania użytkownika** i **aktualizowała informacje zapisane w bazie danych**. Dowiesz się także, jak **odświeżać wyświetlone dane** po ich aktualizacji. I w końcu przekonasz się, że pisanie **wydajnego, wielowątkowego kodu** przy użyciu klasy **AsyncTask** pozwala zapewnić odpowiednią szybkość działania aplikacji.



Chcemy, by nasza aplikacja aktualizowała dane w bazie	694
Dodanie pola wyboru do układu aktywności DrinkActivity	696
Wyświetlanie wartości kolumny FAVORITE	697
Odpowiadanie na kliknięcia w celu aktualizacji bazy	698
Kompletny kod aktywności DrinkActivity	701
Wyświetlanie ulubionych napojów w aktywności TopLevelActivity	705
Refaktoryzacja pliku TopLevelActivity.java	707
Nowy kod aktywności TopLevelActivity	710
Kursor można zmieniać za pomocą metody changeCursor()	715
Który kod umieścić w którym wątku?	723
Klasa AsyncTask służy do wykonywania operacji asynchronicznych	724
Metoda onPreExecute()	725
Metoda doInBackground()	726
Metoda onProgressUpdate()	727
Metoda onPostExecute()	728
Parametry klasy AsyncTask	729
Kompletny kod klasy UpdateDrinkTask	730
Kompletny kod pliku DrinkActivity.java	732
Twój przybornik do Androida	737
Podsumowanie etapów działania zadań AsyncTask	737



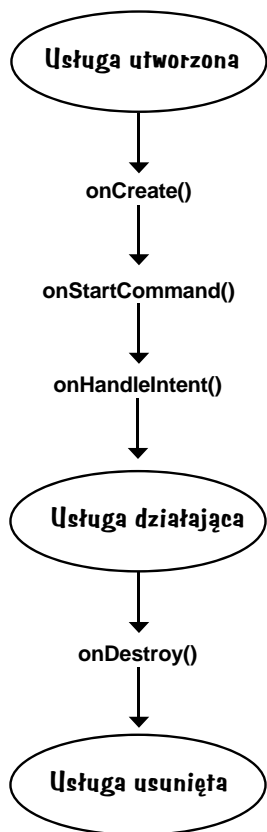
18

Usługi uruchomione

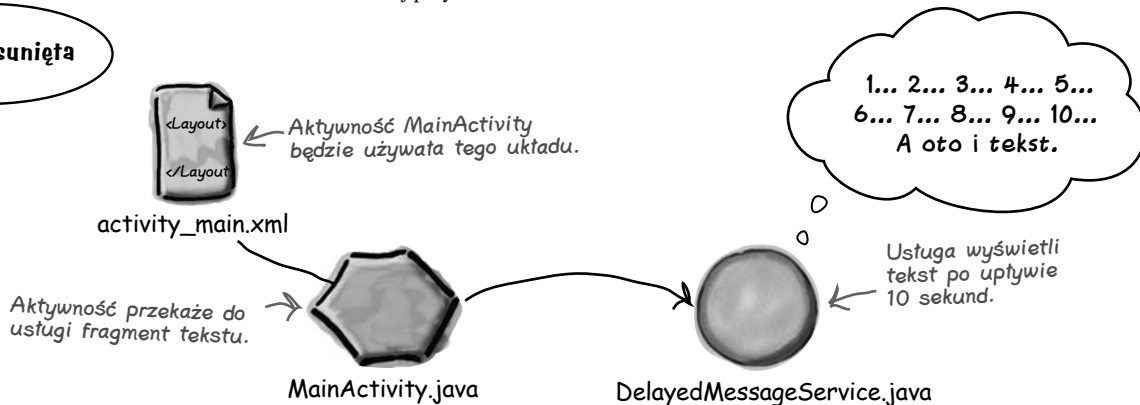
Do usług

Są operacje, które będziemy chcieli wykonywać niezależnie od tego, która aplikacja jest widoczna na ekranie.

Na przykład jeśli rozpoczniesz pobieranie pliku, to zapewne *nie będziesz chciał, by proces pobierania był przerywany, kiedy przełączysz się do innej aplikacji*. W tym rozdziale przedstawimy **usługi uruchomione**, komponenty, które *wykonyją operacje w tle*. Dowiesz się, jak stworzyć usługę uruchomioną, używając klasy `IntentService`, oraz w jaki sposób cykl życia takiej usługi jest powiązany z cyklem życia aktywności. W międzyczasie odkryjesz, jak można **rejestrować komunikaty** i zadbać o to, by *użytkownik zawsze był dobrze poinformowany*, wykorzystując do tego **usługę powiadomień**.



Usługi działają w tle	740
Utworzymy usługę URUCHOMIONĄ	741
Użycie klasy <code>IntentService</code> do utworzenia prostej usługi uruchomionej	742
Jak rejestrować komunikaty?	743
Kompletny kod usługi <code>DelayedMessageService</code>	744
Usługi są deklarowane w pliku <code>AndroidManifest.xml</code>	745
Dodajemy przycisk do układu <code>activity_main.xml</code>	746
Usługę uruchamiamy, wywołując metodę <code>startService()</code>	747
Stany usług uruchomionych	750
Cykl życia usług uruchomionych: od utworzenia do usunięcia	751
Nasza usługa dziedziczy metody cyklu życia	752
Android dysponuje wbudowaną usługą obsługi powiadomień	755
Użyjemy powiadomień z biblioteki wsparcia <code>AppCompat</code>	756
W pierwszej kolejności tworzymy budowniczego powiadomień	757
Wysyłanie powiadomień przy użyciu wbudowanej usługi systemowej	759
Kompletny kod usługi <code>DelayedMessageService</code>	760
Twój przybornik do Androida	765



19

Usługi powiązane i uprawnienia

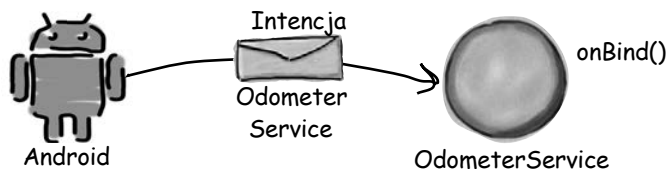
Powiązane ze sobą

Usługi uruchomione są doskonale do wykonywania operacji w tle, ale co zrobić, gdy potrzebujemy usługi, która będzie bardziej interaktywna?

W tym rozdziale dowiesz się, jak tworzyć **usługi powiązane**, czyli *usługi kolejnego typu*, w których *aktywności mogą wchodzić w interakcje*. Dowiesz się także, jak **powiązać** usługę, kiedy to będzie potrzebne, oraz jak ją **odłączyć** w celu oszczędzania zasobów, kiedy nie będzie już potrzebna. Przy okazji nauczysz się korzystać z **usług lokalizacyjnych** Androida, by *pobierać z odbiornika GPS urządzenia aktualne informacje o położeniu*. I w końcu dowiesz się, jak używać **modelu uprawnień Androida**, w tym także jak obsługiwać *żądania przydzielenia uprawnień zgłaszane podczas działania aplikacji*.



Usługi powiązane są skojarzone z innymi komponentami	768
Utworzenie nowej usługi	770
Zdefiniowanie obiektu Binder	771
Dodanie metody getDistance() do usługi	772
Aktualizacja układu aktywności MainActivity	773
Utworzenie obiektu ServiceConnection	775
Użycie metody bindService() do powiązania usługi	778
Użycie metody unbindService() do odłączenia aktywności od usługi	779
Wyświetlenie przebytego dystansu	780
Kompletny kod aktywności MainActivity	781
Stany usług powiązanych	787
Dodanie biblioteki wsparcia AppCompatActivity	790
Dodanie do usługi OdometerService obiektu nasłuchującego danych o lokalizacji	792
Zaktualizowany kod usługi OdometerService	795
Wyliczenie przebytego dystansu	796
Kompletny kod pliku OdometerService.java	798
Jak poprosić o uprawnienia z poziomu aplikacji?	802
Sprawdzenie odpowiedzi na prośbę	805
Dodanie kodu wyświetlającego powiadomienia do metody onRequestPermissionsResult()	809
Kompletny kod pliku MainActivity.java	811
Twój przybornik do Android	815
Świetnie, że odwiedziliście nas w Androidowie	816



Układy względne i układy siatki

Poznaj krewnych

A

Istnieją jeszcze dwa inne układy często stosowane w Androidowie.

W tej książce koncentrowaliśmy się na stosowaniu prostych *układów liniowych* i *układów ramek*, jak również przedstawiliśmy *nowy układ z ograniczeniami*. Jednak istnieją także dwa inne rodzaje układów, które chcielibyśmy Ci zaprezentować: **układ względny** oraz **układ siatki**. W większości zostały one zastąpione układem z ograniczeniami, niemniej jednak i tak pozostaje wiele okazji do ich stosowania, dlatego też przypuszczamy, że będą one używane jeszcze przez ładnych parę lat.



Gradle

B

Program do budowy Gradle

Większość aplikacji na Androida jest budowana przy użyciu programu narzędziowego o nazwie Gradle.

Program **Gradle** działa za kulisami: odnajduje i pobiera biblioteki, kompiluje i wdraża kod, wykonuje testy, czyści fugi i tak dalej... W większości przypadków możemy sobie nawet nie zdawać sprawy, że Gradle istnieje, gdyż Android Studio udostępnia do jego obsługi graficzny interfejs użytkownika. Jednak czasami warto zajrzeć mu pod maskę i **ręcznie go podregulować**.

W tym dodatku pokażemy Ci niektóre spośród wielu talentów i umiejętności programu Gradle.

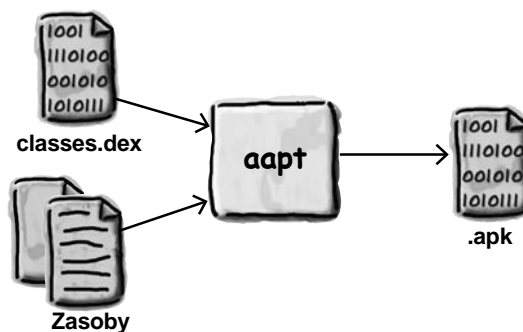
ART



Środowisko uruchomieniowe Androida

Czy kiedykolwiek zastanawiałeś się, jak to się dzieje, że aplikacje na Androida mogą działać na tak wielu rodzajach urządzeń?

Aplikacje na Androida są wykonywane wirtualnej maszynie nazywanej środowiskiem uruchomieniowym Androida (ART — *Android runtime*), a nie wirtualnej maszynie Javy firmy Oracle. Oznacza to, że będą one szybciej uruchamiane na niewielkich urządzeniach o niewielkiej mocy obliczeniowej i będą na nich działać bardziej efektywnie. W tym dodatku dowiesz się, jak działa ART.



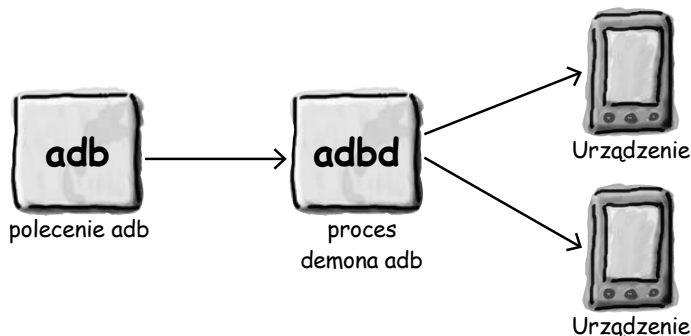
ADB



Android Debug Bridge

W tej książce skoncentrowaliśmy się na zaspokajaniu wszystkich potrzeb związanych z pisaniem aplikacji na Androida z wykorzystaniem IDE.

Zdarzają się jednak sytuacje, w których zastosowanie narzędzi obsługiwanych z poziomu wiersza poleceń jest po prostu przydatne. Mamy tu na myśli na przykład przypadki, gdy Android Studio nie jest w stanie zauważyć urządzenia z Androidem, choć my wiemy, że ono istnieje. W tym rozdziale przedstawimy **Android Debug Bridge** (w skrócie **ADB**) — obsługiwany z poziomu wiersza poleceń program narzędziowy, którego można używać do komunikacji z emulatorem lub z rzeczywistymi urządzeniami zaopatrzonymi w Androida.



Emulator

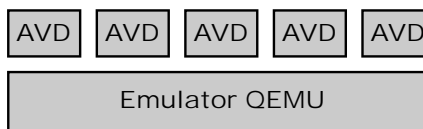
E

Przyspieszanie emulatora

Czy miałeś kiedyś wrażenie, że cały swój czas spędzasz, czekając na emulator?

Nie ma najmniejszych wątpliwości co do tego, że emulator Androida jest bardzo przydatny. Dzięki niemu możemy sprawdzić, jak nasza aplikacja będzie działała na urządzeniach innych niż te, do których mamy fizyczny dostęp. Niekiedy jednak można odnieść wrażenie, że emulator działa... wolno. W tym dodatku wyjaśnimy, dlaczego tak się dzieje. Ale to nie wszystko, damy Ci bowiem także kilka wskazówek, jak przyspieszyć jego działanie.

Wszystkie wirtualne urządzenia z Androidem działają na emulatorze nazywanym QEMU.



Pozostałości

F

Dziesięć najważniejszych zagadnień (których nie opisaliśmy)

Nawet po tym wszystkim, co opisaliśmy w tej książce, wciąż pozostaje wiele innych interesujących zagadnień.

Jest jeszcze kilka dodatkowych spraw, o których musisz się dowiedzieć. Czuliśmy się nie w porządku, gdybyśmy je pominęli, a jednocześnie chcieliśmy oddać w Twoje ręce książkę, którą dasz radę podnieść bez intensywnego treningu na siłowni. Dlatego zanim odłożysz tę książkę, przeczytaj kilka dodatkowych zagadnień opisanych w tym dodatku.

Jeśli kogoś to interesuje, to bateria niedługo się rozładuje.



- | | |
|--------------------------------|-----|
| 1. Rozpowszechnianie aplikacji | 862 |
| 2. Dostawcy treści | 863 |
| 3. Klasy Loader | 864 |
| 4. Adaptery synchronizujące | 864 |
| 5. Odbiorycy komunikatów | 865 |
| 6. Klasa WebView | 866 |
| 7. Ustawienia | 867 |
| 8. Animacje | 868 |
| 9. Widżety aplikacji | 869 |
| 10. Testy zautomatyzowane | 870 |

S

Skorowidz

873

14. Szuflady nawigacyjne

* Z miejsca na miejsce *



Wiem, że nigdy się nie zgubię, póki mam swoje szczęśliwe szuflady nawigacyjne.

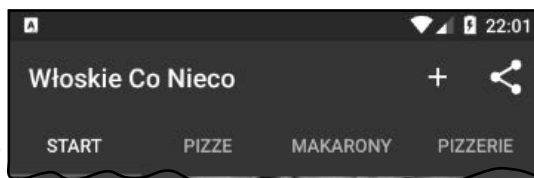
Zobaczyłeś się już, w jaki sposób karty ułatwiają użytkownikom poruszanie się po aplikacji.

Jeśli jednak będziemy ich potrzebowali *bardzo dużo* lub jeśli *trzeba je rozdzielić na sekcje*, to **szuflada nawigacyjna** będzie Twoim nowym najlepszym przyjacielem. W tym rozdziale pokażemy Ci, jak stworzyć szufladę nawigacyjną, która *wysuwa się z boku ekranu po jednym kliknięciu*. Dowiesz się, jak przygotować nagłówek takiej szuflady przy użyciu **widoku nawigacyjnego**, jak wypełnić ją **zestawem elementów o zadanej strukturze**, które pozwolą użytkownikom docierać do wszystkich głównych punktów aplikacji. I w końcu dowiesz się, jak przygotować **obiekt nasłuchujący widoku nawigacyjnego**, dzięki któremu szuflada będzie w stanie *reagować na najdelikatniejsze dotknięcia i przeciągnięcia*.

Widoki kart zapewniają łatwą nawigację...

W rozdziale 12. przedstawiliśmy układ kart jako sposób ułatwiania użytkownikom poruszania się po aplikacji. W tym rozdziale dodaliśmy do naszej aplikacji *Włoskie Co Nieco* ekran startowy oraz karty dla wszystkich głównych kategorii: *Pizze*, *Makarony* oraz *Pizzerie*:

To są karty, które utworzyliśmy w rozdziale 12. →



Układy kart świetnie się sprawdzają w sytuacjach, gdy mamy niewielką liczbę ekranów kategorii, znajdujących się na tym samym poziomie hierarchii aplikacji. Ale co zrobić w przypadkach, gdy chcemy używać dużej liczby kart bądź też grupować karty w sekcje?

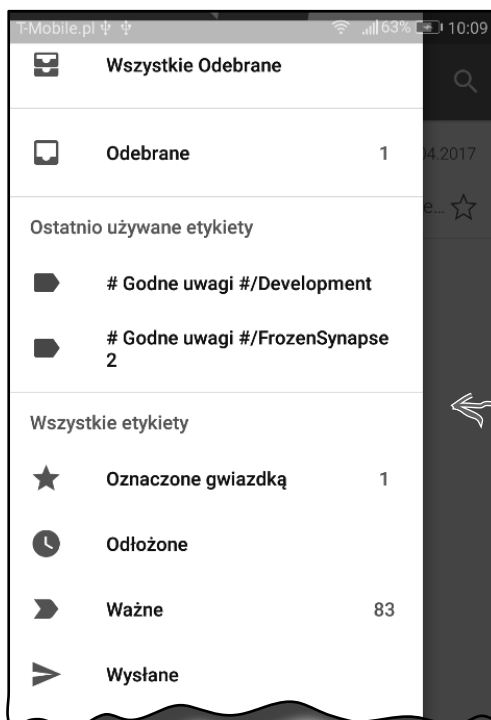
...ale szuflady nawigacyjne pozwalają wyświetlać więcej opcji

Jeśli chcemy zapewnić użytkownikom możliwość wybierania spośród dużej liczby opcji bądź też grupowania ich w sekcje, to lepszym rozwiązaniem niż karty mogą się okazać **szuflady nawigacyjne**. Szuflada nawigacyjna to panel wysuwany spoza pionowej krawędzi ekranu i zawierający odnośniki do innych miejsc aplikacji; pozwala on także grupować te odnośniki w sekcje. Na przykład aplikacja Gmail używa szuflady nawigacyjnej zawierającej sekcje takie jak kategorie e-maili oraz wszystkie etykiety:

Główne kategorie e-maili są umieszczone na górze szuflady.

W osobnej sekcji są wyświetlane kategorie, które użytkownik ostatnio wybrał.

I w końcu u dołu znajduje się długa lista wszystkich etykiet wiadomości.



To jest aplikacja Gmail. Jest ona wyposażona w szufladę nawigacyjną, wysuwaną z boku i przestaniającą główną zawartość ekranu. Szuflada ta udostępnia wiele opcji, których można używać do przechodzenia do różnych miejsc w aplikacji.

Kliknięcie któregoś z elementów umieszczonych w szufladzie powoduje jej zamknięcie i wyświetlenie zawartości odpowiadającej wybranej opcji.

Planujemy utworzenie szufłady nawigacyjnej w nowej aplikacji pocztowej

W tym rozdziale planujemy utworzenie szufłady nawigacyjnej w nowej aplikacji do obsługi poczty elektronicznej, którą nazwiemy Koci Czat. Nasza szufłada nawigacyjna będzie zawierać nagłówek (prezentujący obrazek i trochę tekstu) oraz zestaw opcji. Podstawowe opcje będą zapewniać dostęp do skrzynki odbiorczej, szkiców, wysłanych oraz kosza. Dostępna będzie także odrębna sekcja wsparcia, z odnośnikami do systemu pomocy oraz ekranu do przesyłania informacji zwrotnych.

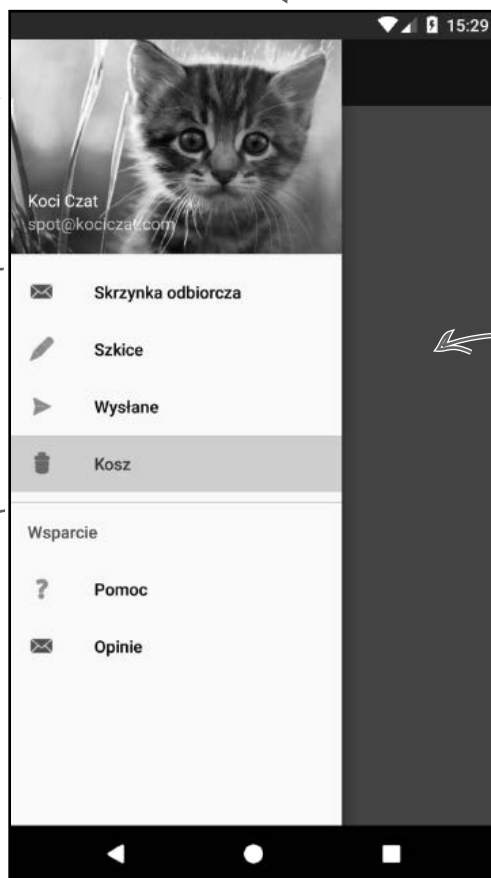
To jest aplikacja Koci Czat.

To jest szufłada nawigacyjna.

To są główne opcje szufłady.

Opcje Pomoc oraz Opinie zostały umieszczone w odrębnej sekcji.

Tu jest wyświetlana główna zawartość aplikacji.



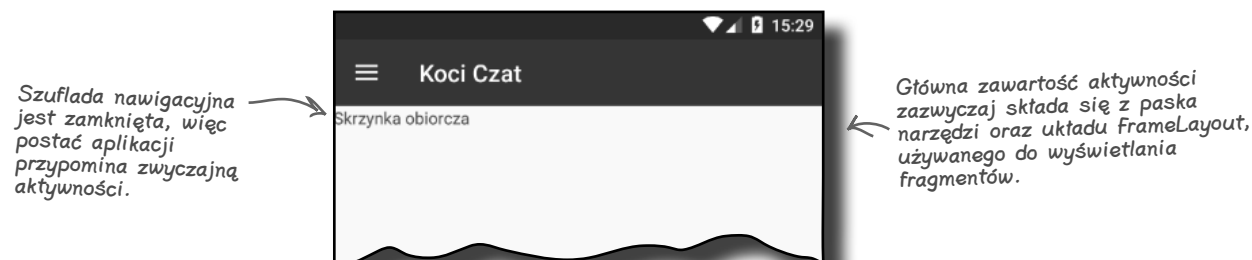
Szufłada nawigacyjna składa się z kilku różnych komponentów. Przedstawimy je wszystkie na następnej stronie.

Szuflady nawigacyjne rozmontowane na czynniki pierwsze

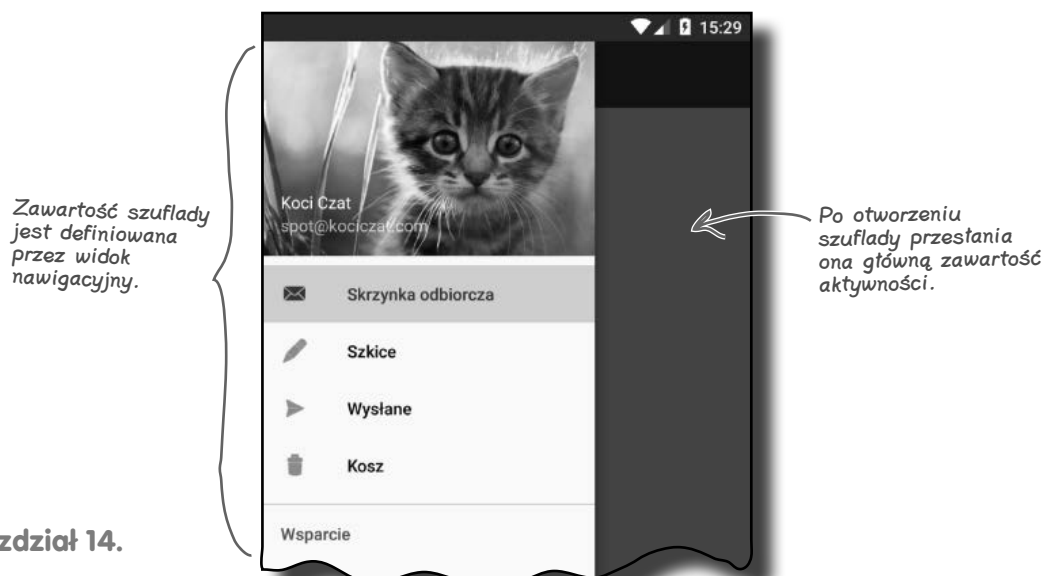
Szufladę nawigacyjną tworzymy poprzez dodanie do układu aktywności **układu szuflady**. Definiuje on szufladę, którą można otwierać i zamykać, a dodatkowo musi ona zawierać dwa widoki:

- 1 Widok do prezentacji głównej zawartości aplikacji.**
Zazwyczaj jest to układ zawierający pasek narzędzi oraz układ ramki, którego można używać do wyświetlania fragmentów.
- 2 Widok do prezentacji zawartości szuflady.**
Zazwyczaj jest to widok nawigacyjny, kontrolujący przeważającą większość możliwości funkcjonalnych szuflady.

Gdy szuflada jest zamknięta, jej układ przypomina układ standardowej aktywności — prezentuje wyłącznie układ głównej zawartości:



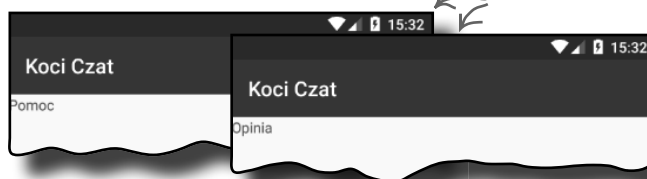
Po otwarciu szuflady nawigacyjnej zostaje ona wysunięta w celu wyświetlenia jej zawartości i częściowo przesłania główną zawartość aktywności. Zawartość szuflady zazwyczaj składa się z widoku nawigacyjnego prezentującego obrazek stanowiący nagłówek oraz listę opcji. Kliknięcie jednej z tych opcji powoduje uruchomienie nowej aktywności bądź też wyświetlenie odpowiedniego fragmentu w układzie `FrameLayout` aktywności.



Oto co zamierzamy zrobić

Zamierzamy zaimplementować szufladę nawigacyjną w aplikacji KociCzat. Cały proces będzie się składać z czterech głównych etapów:

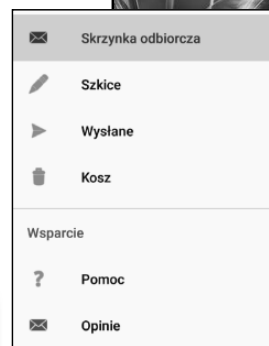
- 1 Utworzenia prostych fragmentów i aktywności dla treści aplikacji.**
 Chcemy, by po kliknięciu jednej z opcji dostępnych w szufladzie nawigacyjnej został wyświetlony odpowiedni fragment lub została uruchomiona odpowiednia aktywność. Na potrzeby aplikacji utworzymy następujące fragmenty: `InboxFragment`, `DraftsFragment`, `SentFragment` oraz `TrashFragment`, a także aktywności: `HelpActivity` oraz `FeedbackActivity`.



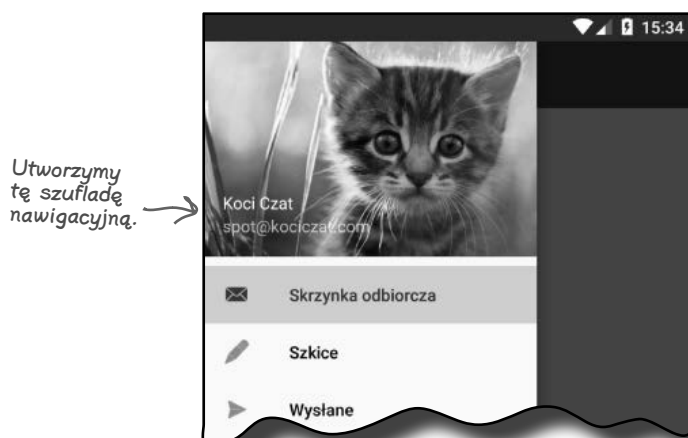
- 2 Utworzenia nagłówka szuflady nawigacyjnej.**
 Na potrzeby nagłówka szuflady nawigacyjnej utworzymy plik układu `nav_header.xml`; będzie on zawierał obrazek i tekst.

- 3 Utworzenia opcji szuflady nawigacyjnej.**
 W celu określenia opcji wyświetlanych w szufladzie przygotujemy menu `menu_nav.xml`.

- 4 Utworzenia szuflady nawigacyjnej.**
 Szufladę nawigacyjną dodamy do głównej aktywności aplikacji i zadamy o wyświetlanie jej nagłówka i opcji. Następnie napiszemy kod, który będzie kontrolować działanie szuflady.



↑
Nagłówek
i opcje
szuflady
nawigacyjnej
←



A zatem zaczynamy.

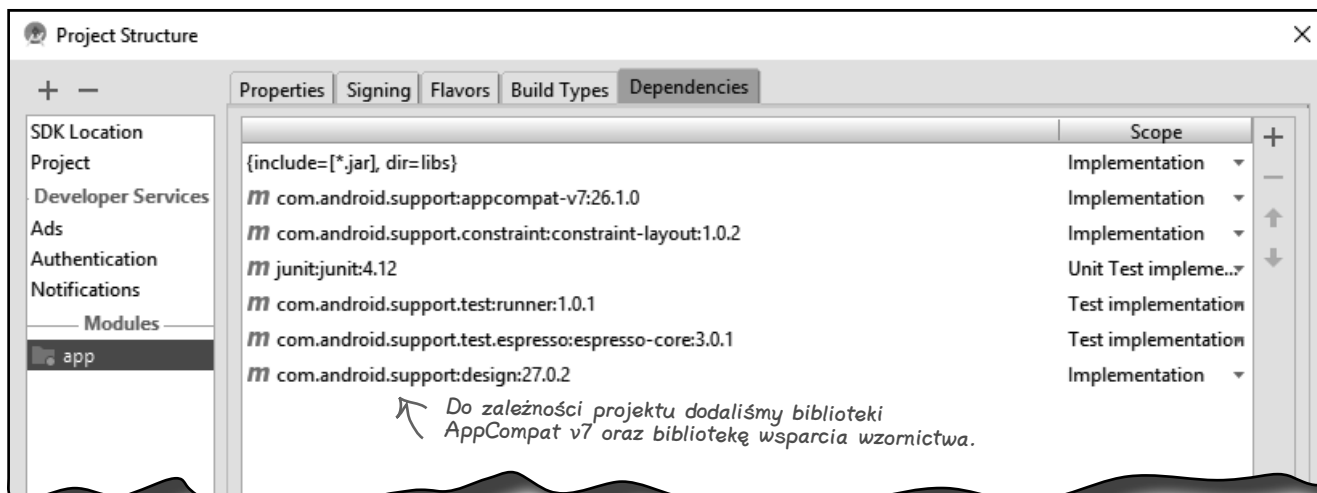
Utworzenie projektu Koci Czat

Zanim zaczniemy, musimy utworzyć nowy projekt na potrzeby prac nad aplikacją. Utwórz zatem nowy projekt aplikacji na Androida, nadaj jej nazwę KociCzat i użyj domeny firmowej hfad.com, dzięki czemu pakiet aplikacji będzie miał nazwę com.hfad.kociczat. Jako minimalny poziom SDK wybierz API poziomu 19., dzięki czemu aplikacja będzie działać na większości urządzeń. Podczas tworzenia aplikacji dodaj do niej pustą aktywność, nadaj jej nazwę MainActivity, a używany przez nią plik układu nazwij activity_main. **Konieczniesz zaznaczyć pole wyboru *Backwards Compatibility (AppCompat)*.**

- **Fragmenty i aktywności**
- Nagłówki**
- Opcje**
- Szuflada**

Dodaj bibliotekę AppCompat v7 oraz bibliotekę wsparcia wzornictwa

W aplikacji będziemy używać komponentów i motywów z biblioteki wsparcia AppCompat v7 oraz z biblioteki wsparcia wzornictwa, dlatego musimy dodać je jako zależności naszego projektu. W tym celu wybierz z menu głównego Android Studio opcję *File/Project Structure*, kliknij moduł *app* i przejdź na kartę *Dependencies*. Po wyświetleniu karty z zależnościami projektu kliknij przycisk „+”, umieszczony u dołu lub z prawej strony okna dialogowego. Następnie wybierz opcję *Library Dependency* i z listy dostępnych bibliotek wybierz *Design Library*. Te same czynności powtórz, by dodać bibliotekę AppCompat v7, o ile Android Studio nie dodało jej automatycznie podczas tworzenia projektu. W końcu kliknij przycisk *OK*, aby zapisać zmiany.



Teraz zajmiemy się przygotowaniem czterech prostych fragmentów dla: skrzynki odbiorczej, szkiców, wiadomości wysłanych oraz kosza. Zastosujemy je w dalszej części rozdziału, kiedy napiszemy kod obsługujący działanie szuflady nawigacyjnej.

Utworzenie fragmentu InboxFragment

Fragment `InboxFragment` będziemy wyświetlać, kiedy użytkownik kliknie opcję *Skrzynka odbiorcza* w szufladzie nawigacyjnej. Zaznacz zatem pakiet `com.hfad.kociczat` w katalogu `app/src/main/java` i wybierz z menu głównego opcję *File/New.../Fragment/Fragment (Blank)*. Tworzonemu fragmentowi nadaj nazwę `InboxFragment`, a jego plikowi układu nazwę `fragment_inbox`. Następnie zaktualizuj kod w pliku `InboxFragment.java`, by był taki sam jak ten przedstawiony poniżej:

```
package com.hfad.kociczat;

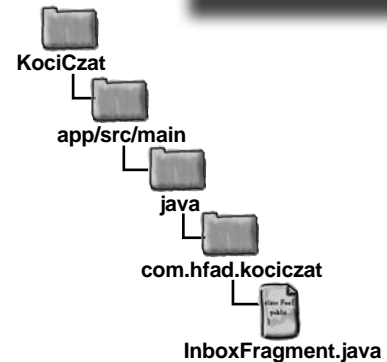
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class DraftsFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_drafts, container, false);
    }
}
```

← Wszystkie fragmenty używają klasy `Fragment` z biblioteki wsparcia.

Oto jak będzie wyglądać fragment `InboxFragment`.

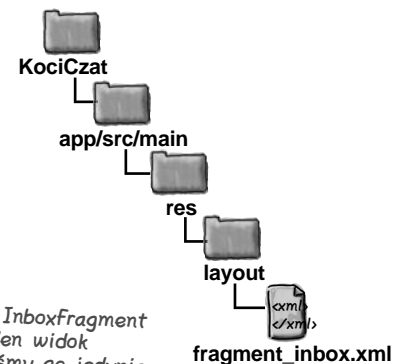


A oto kod pliku `fragment_inbox.xml` (zaktualizuj swoją wersję tego pliku, tak by była identyczna z naszą):

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.kociczat.InboxFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Skrzynka odbiorcza" />
</LinearLayout>
```

Układ fragmentu `InboxFragment` zawiera tylko jeden widok `TextView`. Dodaliśmy go jedynie po to, aby można było łatwo określić, kiedy będzie on wyświetlany.



Utworzenie fragmentu DraftsFragment

Fragment DraftsFragment będziemy wyświetlać, kiedy użytkownik kliknie opcję *Szkice* w szufladzie nawigacyjnej. Zaznacz zatem pakiet `com.hfad.kociczat` w katalogu `app/src/main/java` i wybierz z menu głównego opcję *File/New.../Fragment/Fragment (Blank)*. Tworzonemu fragmentowi nadaj nazwę `DraftsFragment`, a jego plikowi układu nazwę `fragment_drafts`. Następnie zaktualizuj kod w pliku `DraftsFragment.java`, by był taki sam jak ten przedstawiony poniżej:

```
package com.hfad.kociczat;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class DraftsFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_drafts, container, false);
    }
}
```

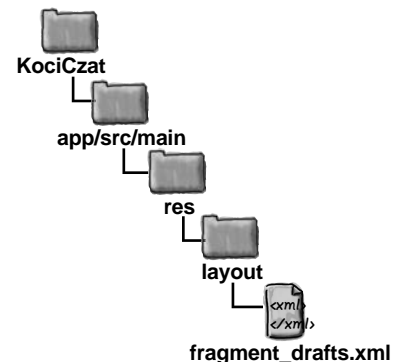
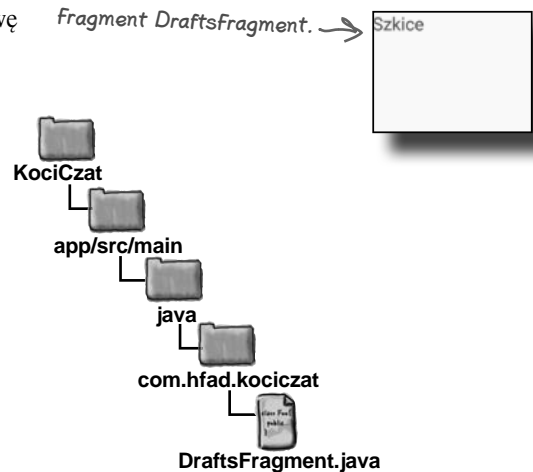
A oto kod pliku `fragment_drafts.xml` (zaktualizuj swoją wersję tego pliku, tak by była identyczna z naszą):

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.kociczat.DraftsFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Szkice" />

</LinearLayout>
```

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada



Utworzenie fragmentu SentItemsFragment

Fragment `SentItemsFragment` będziemy wyświetlać, kiedy użytkownik kliknie opcję *Wysłane* w szufladzie nawigacyjnej. Zaznacz zatem pakiet `com.hfad.kociczat` w katalogu `app/src/main/java` i wybierz z menu głównego opcję *File/New.../Fragment/Fragment (Blank)*. Tworzonemu fragmentowi nadaj nazwę `SentItemsFragment`, a jego plikowi układu nazwę `fragment_sent_items`. Następnie zaktualizuj kod w pliku `SentItemsFragment.java`, by był taki sam jak ten przedstawiony poniżej:

```
package com.hfad.kociczat;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class SentItemsFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_sent_items, container, false);
    }
}
```

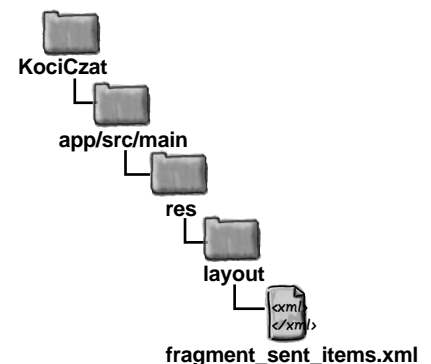
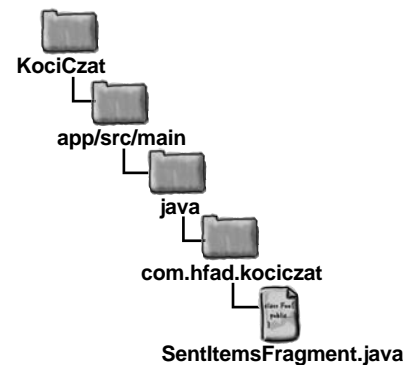
A oto kod pliku `fragment_sent_items.xml` (zaktualizuj swoją wersję tego pliku, tak by była identyczna z naszą):

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.kociczat.SentItemsFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Wiadomości wysłane" />

</LinearLayout>
```

- Fragmenty i aktywności
- Nagłówkek
- Opcje
- Szuflada



Utworzenie fragmentu TrashFragment

Fragment TrashFragment będziemy wyświetlać, kiedy użytkownik kliknie opcję *Kosz* w szufladzie nawigacyjnej. Zaznacz zatem pakiet `com.hfad.kociczat` w katalogu `app/src/main/java` i wybierz z menu głównego opcję *File/New.../Fragment/Fragment (Blank)*. Tworzonemu fragmentowi nadaj nazwę TrashFragment, a jego plikowi układu nazwę `fragment_trash`. Następnie zaktualizuj kod w pliku `TrashFragment.java`, by był taki sam jak ten przedstawiony poniżej:

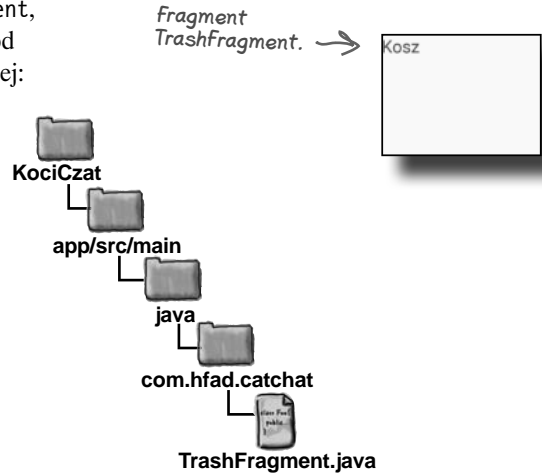
- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada

```
package com.hfad.kociczat;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class TrashFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_trash, container, false);
    }
}
```

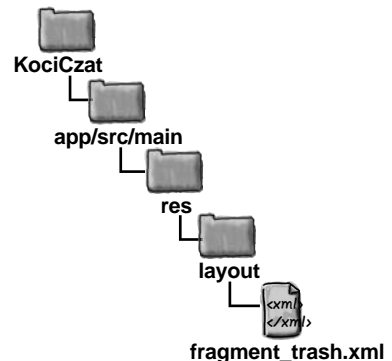


Teraz zastąp także kod w pliku `fragment_trash.xml`:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.kociczat.TrashFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Kosz" />

</LinearLayout>
```



W ten sposób utworzyliśmy już wszystkie fragmenty, których będziemy potrzebować. W kolejnym kroku zajmiemy się przygotowaniem paska narzędzi, który będziemy mogli wykorzystać w aktywnościach.

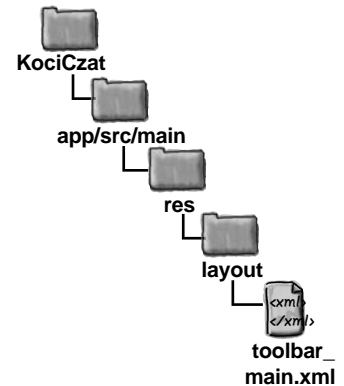
Przygotowanie układu paska narzędzi

Teraz zajmiemy się przygotowaniem paska narzędzi. Jego kod zapiszemy w odrębnym pliku układu, tak byśmy mogli łatwo dołączyć go do układu każdej z aktywności (nimi zajmiemy się w dalszej kolejności). W eksploratorze Android Studio przejdź do widoku *Project*, wybierz z katalogu *app/src/res/main/layout*, a następnie wybierz z menu opcję *File/New/Layout resource file*. Kiedy zostaniesz poproszony o podanie nazwy, wpisz *toolbar_main*. W końcu kliknij przycisk *OK*.

Kiedy wykonasz powyższe czynności, otwórz plik *toolbar_main.xml* i zastąp w nim kod wygenerowany przez Android Studio kodem zamieszczonym poniżej:

```
<android.support.v7.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" />
```

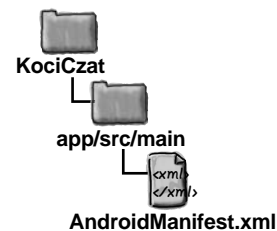
To jest ten sam kod paska zadań, którego używaliśmy w poprzednich rozdziałach.



Zanim będziemy mogli użyć tego paska narzędzi w którejkolwiek z naszych aktywności, musimy zmienić wykorzystywany w nich motyw. Odpowiednie zmiany należy wprowadzić w zasobie stylów aplikacji.

Otwórz plik manifestu, *AndroidManifest.xml*, i upewnij się, że wartością atrybutu *theme* jest *"@style/AppTheme"*. Może się zdarzyć, że Android Studio samo ustawi wartość tego atrybutu; jeśli jednak tak się nie stanie, to będziesz musiał zmodyfikować ją samodzielnie, tak jak pokazaliśmy w poniższym przykładzie:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      ...
    </activity>
  </application>
</manifest>
```



Może się zdarzyć, że Android Studio samo ustawi tę wartość.

Aktualizacją stylu zajmiemy się na następnej stronie.

Aktualizacja motywu aplikacji

W kolejnym kroku zmodyfikujemy styl AppTheme, tak by korzystał on z motywu "Theme.AppCompat.Light.NoActionBar". Poza tym zmienimy parę kolorów używanych w domyślnym motywie.

W pierwszej kolejności otwórz katalog `app/src/main/res/values` i sprawdź, czy Android Studio utworzyło w nim plik o nazwie `styles.xml`. Jeśli taki plik nie istnieje, to go utwórz. W tym celu zaznacz katalog `values` i wybierz z menu opcję `File/New.../Values resource file`. Jako nazwę pliku wpisz `styles` i kliknij przycisk `OK`.

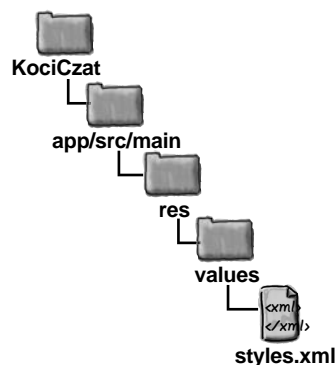
Następnie zaktualizuj zawartość pliku `styles.xml`, by była identyczna z poniższym fragmentem kodu:

```
<resources>
  <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
  </style>
</resources>
```

Ten motyw usuwa domyślny pasek aplikacji (zastępujemy go paskiem narzędzi).

Android Studio mogło samo określić te kolory.

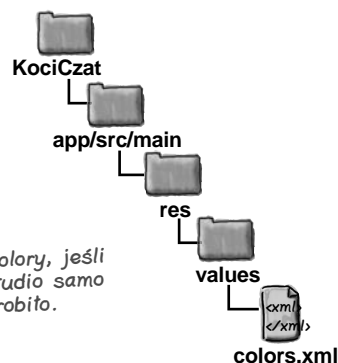
- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada



Styl AppTheme używa zasobów kolorów, a te powinny zostać zdefiniowane w pliku `colors.xml`. W pierwszej kolejności upewnij się, że Android Studio utworzyło ten plik w katalogu `app/src/main/res/values` (a jeśli go nie utworzyło, to zrób to samodzielnie). Następnie zapisz w pliku `colors.xml` poniższy fragment kodu:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="colorPrimary">#3F51B5</color>
  <color name="colorPrimaryDark">#303F9F</color>
  <color name="colorAccent">#FF4081</color>
</resources>
```

Dodaj te kolory, jeśli Android Studio samo tego nie zrobiło.



Skoro już przygotowaliśmy style pozwalające na zastosowanie paska narzędzi, utworzymy dwie aktywności: jedną na potrzeby systemu pomocy aplikacji oraz drugą do podawania opinii. Aktywności te będziemy uruchamiać, kiedy użytkownik wybierze odpowiednie opcje z szuflady nawigacyjnej.

Utworzenie aktywności HelpActivity

Zacniemy od utworzenia aktywności `HelpActivity`. Zaznacz pakiet `com.hfad.kociczat`, a następnie wybierz z menu opcję *File/New*. Potem wybierz opcję utworzenia pustej aktywności, nadaj jej nazwę `HelpActivity`, a używanemu przez nią plikowi układu nazwę `activity_help`. Upewnij się, że klasa aktywności będzie należeć do pakietu `com.hfad.kociczat` oraz że **jest zaznaczone pole wyboru *Backwards Compatibility (AppCompat)***. Następnie zaktualizuj swoją wersję pliku `activity_help.xml`, tak by zawierał poniższy kod:

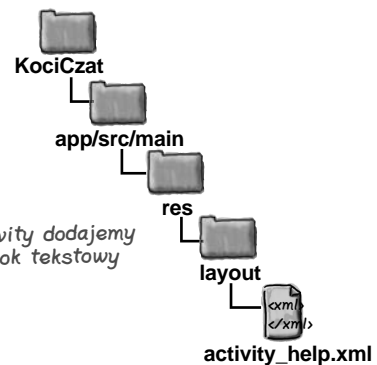
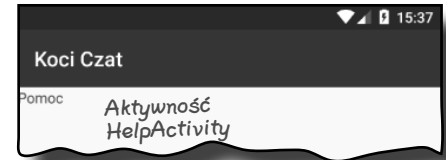
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.kociczat.HelpActivity">

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Pomoc" />

</LinearLayout>
```

Do aktywności `HelpActivity` dodajemy pasek narzędzi oraz widok tekstowy ze słowem „Pomoc”.



Teraz zaktualizuj plik `HelpActivity.java`, zapisując w nim poniższy kod:

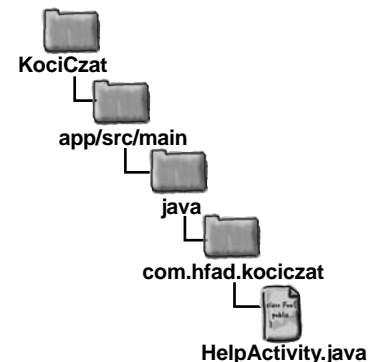
```
package com.hfad.kociczat;

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

public class HelpActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_help);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }
}
```

Używamy motywu `AppCompat`, więc aktywność musi dziedziczyć po klasie `AppCompatActivity`.



Utworzenie aktywności FeedbackActivity

Teraz zaznacz pakiet com.hfad.kociczat i wybierz z menu opcję *File/New*. Następnie wybierz opcję utworzenia pustej aktywności, nadaj jej nazwę FeedbackActivity, a używanemu przez nią plikowi układu nazwę activity_feedback. Upewnij się, że klasa aktywności będzie należeć do pakietu com.hfad.kociczat oraz że **jest zaznaczone pole wyboru Backwards Compatibility (AppCompat)**. Następnie zaktualizuj swoją wersję pliku *activity_feedback.xml*, tak by zawierał poniższy kod:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.kociczat.FeedbackActivity">

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Opinia" />
</LinearLayout>
```

Teraz zaktualizuj plik *FeedbackActivity.java*, zapisując w nim poniższy kod:

```
package com.hfad.kociczat;

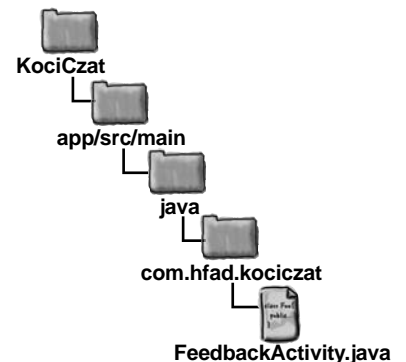
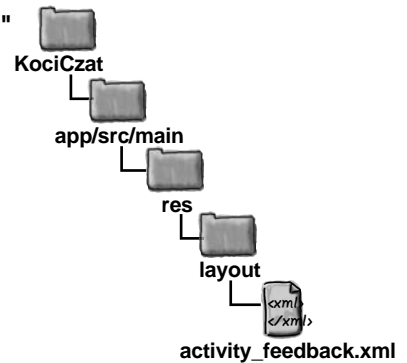
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;

public class FeedbackActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_feedback);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }
}
```

Także ta aktywność musi dziedziczyć po klasie AppCompatActivity.

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada



Musimy przygotować szufładę nawigacyjną

Dodaliśmy już do projektu wszystkie fragmenty i aktywności; będą one powiązane z opcjami, które znajdują się w szufładzie nawigacyjnej. Teraz zajmiemy się utworzeniem samej szufłady.

Szufłada nawigacyjna składa się z dwóch odrębnych komponentów:

★ Nagłówka szufłady.

To układ wyświetlany na samej górze szufłady nawigacyjnej. Zazwyczaj zawiera on obrazek i jakiś tekst, na przykład zdjęcie użytkownika i jego adres poczty elektronicznej.

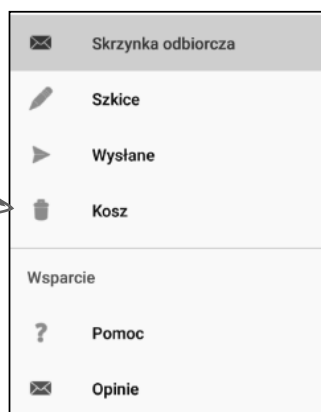
To jest nagłówek szufłady, który utworzymy w naszej aplikacji. Składa się on z obrazka i dwóch fragmentów tekstu.



★ Zestawu opcji.

Zestaw opcji jest wyświetlany w szufładzie nawigacyjnej poniżej jej nagłówka. Kiedy użytkownik kliknie jedną z tych opcji, ekran skojarzony z daną opcją zostanie wyświetlony, bądź to jako fragment w aktywności zawierającej szufładę, bądź jako nowa aktywność.

Szufłada nawigacyjna będzie zawierać te opcje.



Zajmiemy się teraz przygotowaniem obu tych komponentów, a następnie użyjemy ich w aktywności `MainActivity`, aby zaimplementować szufładę nawigacyjną. Zaczniemy od nagłówka szufłady.

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szufłada

Utworzenie nagłówka szuflady nawigacyjnej

Nagłówek szuflady nawigacyjnej składa się z prostego układu, który dodamy do nowego pliku układu. Utworzymy w tym celu nowy plik, o nazwie `nav_header.xml`. A zatem, aby to zrobić, zaznacz w Android Studio katalog `app/src/main/res/layout`, po czym wybierz opcję `File/New/Layout resource file`. Kiedy zostaniesz poproszony o podanie nazwy pliku, wpisz `nav_header`.

Nasz układ nagłówka składa się z trzech widoków: `ImageView` oraz dwóch `TextView`. Oznacza to, że do projektu musimy dodać plik obrazka jako zasób graficzny oraz dwa zasoby łańcuchowe. Zaczniemy od pliku obrazka.

Dodanie pliku obrazka

Aby dodać obrazek, w pierwszej kolejności w eksploratorze Android Studio musisz przełączyć się do widoku `Project`, o ile nie zrobiłeś tego już wcześniej. Następnie sprawdź, czy w projekcie istnieje katalog o nazwie `app/src/main/res/drawable`. Jeśli go nie ma, to zaznacz katalog `app/src/main/res`, wybierz z menu opcję `File/New...`, a następnie opcję pozwalającą utworzyć nowy katalog zasobów aplikacji na Androida. Gdy zostaniesz poproszony o podanie nazwy katalogu, wpisz `drawable` i kliknij `OK`.

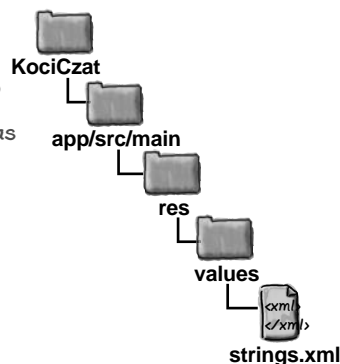
Po utworzeniu katalogu `drawable` w przykładach dołączonych do książki odszukaj plik `kitten_small.jpg` — będzie on umieszczony w analogicznym podrozdziale `drawable`, w katalogu `rozdzial_14` — i skopiuj go do katalogu `drawable` swojego projektu.

Dodanie zasobów łańcuchowych

Kolejnym krokiem będzie dodanie zasobów łańcuchowych, których będziemy używać do określenia tekstów wyświetlanych w nagłówku. Otwórz plik `app/src/main/res/values/strings.xml` i dodaj do niego dwa poniższe zasoby:

```
<resources>
...
<string name="app_name">KociCzat</string>
<string name="user_name">spot@kociczat.com</string>
</resources>
```

Android Studio mogło dodać ten zasób automatycznie podczas tworzenia projektu.



Skoro dodaliśmy już zasoby, możemy się zająć pisaniem kodu układu. Już doskonale potrafisz napisać cały niezbędny kod, dlatego na następnej stronie przedstawimy go w całości.

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada

Nagłówek zawiera widok `ImageView`...



...oraz dwa widoki `TextView`.

Kompletny kod pliku nav_header.xml

Poniżej przedstawiliśmy kompletny kod pliku `nav_header.xml`; zaktualizuj plik w swoim projekcie, tak by był identyczny z naszym:

```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="180dp" ←
    android:theme="@style/ThemeOverlay.AppCompat.Dark" >

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:scaleType="centerCrop"
        android:src="@drawable/kitten_small" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:gravity="bottom|start" ←
        android:layout_margin="16dp" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/app_name"
            android:textAppearance="@style/TextAppearance.AppCompat.Body1" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/user_name" />
    </LinearLayout>
</FrameLayout>

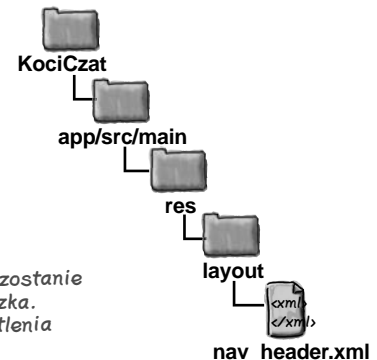
```

Jawnie określamy, że wysokość układu wynosi 180dp, tak by nie zabierał on zbyt dużo miejsca w szufladzie nawigacyjnej.

Ta obrazka jest dość ciemne, więc używamy tego wiersza, który sprawi, że tekst będzie jasny.

Ten układ `LinearLayout` zostanie wyświetlony na tle obrazka. Używamy go do wyświetlenia tekstów u dołu obrazka.

To jest wbudowany styl, który powoduje, że wyświetlany w nim tekst jest nieco bardziej wyfuszczony. Jest on dostępny w bibliotece wsparcia `AppCompat`.



- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada

Skoro już utworzyliśmy nagłówek szuflady nawigacyjnej, możemy przejść do przygotowania listy opcji.

Szuflada pobiera opcje z menu

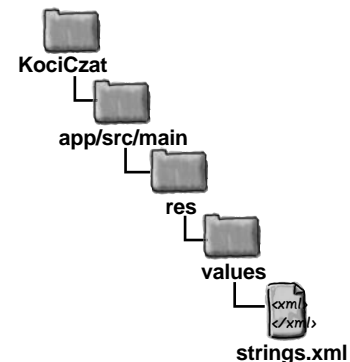
- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada

Szuflada nawigacyjna pobiera listę swoich opcji z pliku zasobu menu. Kod realizujący tę operację jest bardzo podobny do tego, którego używaliśmy do dodawania opcji do paska aplikacji.

Zanim przyjrzymy się kodowi służącemu do dodawania opcji do szuflady nawigacyjnej, musimy dodać do projektu plik zasobu menu. W tym celu zaznacz w Android Studio katalog `app/src/main/res` i z menu głównego wybierz opcję `File/New`. Następnie wybierz opcję pozwalającą na utworzenie nowego pliku zasobu. W efekcie zostaniesz poproszony o podanie nazwy pliku oraz typu zasobu. Jako nazwę wpisz `menu_nav`, a na liście typów zasobów wybierz opcję `Menu`. Upewnij się także, że w polu `Directory` jest wybrany katalog `menu`. Po kliknięciu przycisku `OK` Android Studio utworzy plik zasobu.

W kolejnym kroku dodamy do projektu zasoby łańcuchowe określające tytuły poszczególnych opcji, dzięki czemu będziemy mogli ich użyć w dalszej części rozdziału. Otwórz plik `strings.xml` i dodaj do niego poniższe zasoby:

```
<resources>
...
<string name="nav_inbox">Skrzynka odbiorcza</string>
<string name="nav_drafts">Szkice</string>
<string name="nav_sent">Wysłane</string>
<string name="nav_trash">Kosz</string>
<string name="nav_support">Wsparcie</string>
<string name="nav_help">Pomoc</string>
<string name="nav_feedback">Opinie</string>
</resources>
```



Teraz możemy już przystąpić do pisania kodu menu.

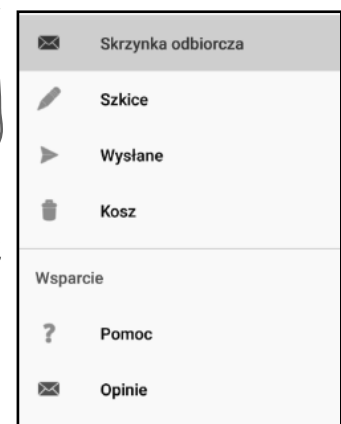
Tworzone menu ma mieć dwie sekcje

Jak już zaznaczyliśmy wcześniej, chcemy podzielić opcje w szufladzie nawigacyjnej na dwie sekcje. Pierwsza z nich będzie zawierać opcje pozwalające użytkownikom na przejście w kilka głównych miejsc aplikacji, a konkretnie do: skrzynki odbiorczej, szkiców, wiadomości wysłanych oraz kosza. Poniżej będzie umieszczona odrębna sekcja zawierająca opcje związane z systemem pomocy oraz opiniami.

Zacznijmy od dodania opcji głównych.

To są główne opcje szuflady nawigacyjnej.

To jest sekcja wsparcia.



Dodawaj opcje w kolejności, w jakiej mają być wyświetlane

Podczas projektowania zestawu opcji mających się znaleźć w szufladzie nawigacyjnej zazwyczaj te z nich, które najprawdopodobniej będą najczęściej klikane przez użytkownika, powinny się znaleźć na samej górze listy. W naszym przykładzie będą to opcje: skrzynki odbiorczej, szkiełców, wiadomości wysłanych i kosza.

Poszczególne elementy należy dodawać do pliku zasobu w takiej kolejności, w jakiej później mają być wyświetlane w szufladzie nawigacyjnej. Dla każdego z elementów należy podać identyfikator, dzięki któremu będzie można odwoływać się do niego w kodzie Javy, oraz tytuł określający tekst, jaki chcemy wyświetlić w szufladzie. Dodatkowo można także określić ikonę, która zostanie wyświetlona obok tytułu opcji. W ramach przykładu poniżej pokazaliśmy kod dodający do szuflady nawigacyjnej opcję „Skrzynka odbiorcza”:

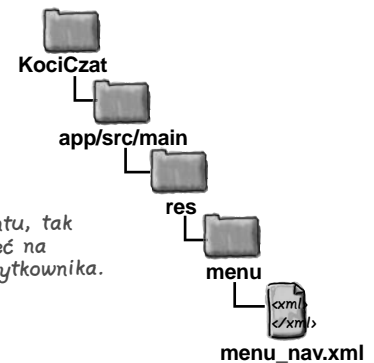
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
```

```
<item
  android:id="@+id/nav_inbox"
  android:icon="@android:drawable/sym_action_email"
  android:title="@string/nav_inbox" />
```

... To jest tekst, który zostanie wyświetlony w szufladzie nawigacyjnej.

Musisz określić identyfikator elementu, tak by kod aktywności mógł odpowiedzieć na kliknięcie danego elementu przez użytkownika.

To jest wbudowany zasób graficzny, którego można użyć do wyświetlenia ikony e-maila.



W powyższym kodzie zastosowaliśmy jedną z wbudowanych ikon dostępnych w systemie Android: "@android:drawable/sym_action_email". System Android zawiera wbudowany zestaw ikon, których można używać w tworzonych aplikacjach. O tym, że chcemy użyć jednej z tych ikon informuje początkowy fragment identyfikatora: "@android:drawable". Pełną listę dostępnych ikon można wyświetlić, rozpoczynając wpisywanie identyfikatora w Android Studio:



Jak można grupować elementy?

Oprócz dodawania do szuflady nawigacyjnej pojedynczych elementów można je także dodawać w grupach. Grupy definiuje się przy użyciu znacznika `<group>`, jak pokazano poniżej:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <group>
    ...
  </group>
</menu>
```

← Tutaj umieszczamy wszystkie elementy, które chcemy dodać do grupy.

Rozwiązanie to jest przydatne, gdy chcemy zastosować jakiś atrybut dla całej grupy elementów. Na przykład aby wyróżnić w szufladzie element wybrany przez użytkownika, można dodać do znacznika `<group>` atrybut `android:checkableBehavior` o wartości `"single"`. Ten sposób działania szuflady jest przydatny, jeśli planujemy wyświetlanie ekranów poszczególnych opcji jako fragmentów podmienianych aktywności, do której została dodana szuflada nawigacyjna (w naszym przykładzie jest to aktywność `MainActivity`), gdyż ułatwia określenie obecnie wybranej opcji:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <group android:checkableBehavior="single">
    ...
  </group>
</menu>
```

↑ Użycie tej wartości atrybutu oznacza, że w grupie będzie wyróżniona pojedyncza opcja (ta, którą wybrał użytkownik).

Wybrany element szuflady nawigacyjnej można domyślnie wyróżnić, dodając do niego atrybut `android:checked` o wartości `"true"`. Poniższy przykład pokazuje, w jaki sposób można wyróżnić element „Skrzynka odbiorcza”:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <group android:checkableBehavior="single">
    <item
      android:id="@+id/nav_inbox"
      android:icon="@android:drawable/sym_action_email"
      android:title="@string/nav_inbox"
      android:checked="true" />
    ...
  </group>
</menu>
```

↑ Zastosowanie tego atrybutu powoduje domyślne wyróżnienie danego elementu w szufladzie nawigacyjnej.

Na następnej stronie pokażemy pełny kod definiujący cztery początkowe elementy szuflady nawigacyjnej.

Opcje pierwszej sekcji zostaną umieszczone w grupie

Opcje dla skrzynki odbiorczej, szkiców, wiadomości wysłanych oraz kosza dodamy do naszego pliku zasobu menu w formie jednej grupy i domyślnie wyróżnimy pierwszą z nich. Opcje te umieścimy w grupie, ponieważ ekrany dla każdej z nich są fragmentami wyświetlanymi w ramach aktywności MainActivity.

Poniżej przedstawiliśmy kodu menu; zaktualizuj swoją wersję pliku `menu_nav.xml`, by była identyczna z naszą:

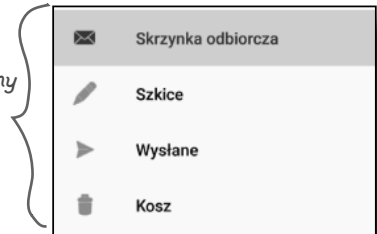
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
```

```
<group android:checkableBehavior="single">
  <item
    android:id="@+id/nav_inbox"
    android:icon="@android:drawable/sym_action_email"
    android:title="@string/nav_inbox"
    android:checked="true" />
  <item
    android:id="@+id/nav_drafts"
    android:icon="@android:drawable/ic_menu_edit"
    android:title="@string/nav_drafts" />
  <item
    android:id="@+id/nav_sent"
    android:icon="@android:drawable/ic_menu_send"
    android:title="@string/nav_sent" />
  <item
    android:id="@+id/nav_trash"
    android:icon="@android:drawable/ic_menu_delete"
    android:title="@string/nav_trash" />
</group>
```

```
</menu>
```

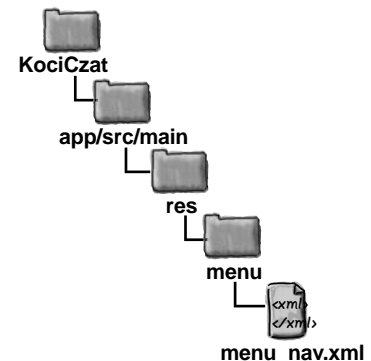
W ten sposób uporaliśmy się z pierwszą grupą opcji. Kolejnymi zajmiemy się w dalszej kolejności.

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szufłada



Kod zamieszczony na tej stronie dodaje do szufłady nawigacyjnej te cztery opcje.

Dodaj tę grupę i cztery umieszczone w niej opcje do pliku zasobu menu, aby zostały wyświetlone w szufładzie nawigacyjnej.



Sekcję wsparcia dodamy jako podmenu

Drugi zestaw elementów tworzonej szuflady nawigacyjnej ma postać odrębnej sekcji. Znajdują się w niej nagłówek „Wsparcie” oraz dwie opcje, „Pomoc” i „Opinie”, które użytkownik będzie mógł klikać.

Tworzenie tej sekcji rozpoczniemy od dodania nagłówka „Wsparcie”, który zdefiniujemy jako osobny element. Ponieważ ma to być nagłówek, zdefiniujemy w nim jedynie tytuł — pominiemy ikonę oraz identyfikator, gdyż ten element szuflady nie będzie musiał reagować na kliknięcie:

```
...
<item android:title="@string/nav_support">
</item>
...
```

Ten element dodaje do szuflady nawigacyjnej nagłówek „Wsparcie”.

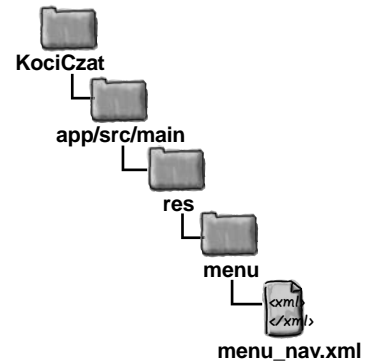
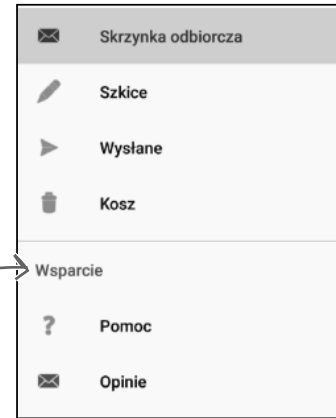
Chcemy, by opcje zapewniające dostęp do systemu pomocy i opinii były wyświetlone w sekcji wsparcia, dlatego dodamy je wewnątrz tej sekcji jako niezależne opcje umieszczone w podmenu:

```
...
<item android:title="@string/nav_support">
  <menu>
    <item
      android:id="@+id/nav_help"
      android:icon="@android:drawable/ic_menu_help"
      android:title="@string/nav_help"/>
    <item
      android:id="@+id/nav_feedback"
      android:icon="@android:drawable/sym_action_email"
      android:title="@string/nav_feedback" />
  </menu>
</item>
...
```

Ten blok kodu definiuje podmenu wewnątrz sekcji Wsparcie.

Ten kod powoduje dodanie do szuflady tych dwóch elementów.

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada



Warto zwrócić uwagę na to, że tych opcji nie dodawaliśmy w formie jednej grupy; a zatem kiedy użytkownik kliknie jedną z nich, nie zostanie ona wyróżniona w szufladzie nawigacyjnej. Zastosowaliśmy takie rozwiązanie, ponieważ obie te opcje będą wyświetlane jako aktywności, a nie fragmenty prezentowane wewnątrz aktywności zawierającej szufladę nawigacyjną.

Pełny kod menu przedstawimy na następnej stronie.

Kompletny kod pliku menu_nav.xml

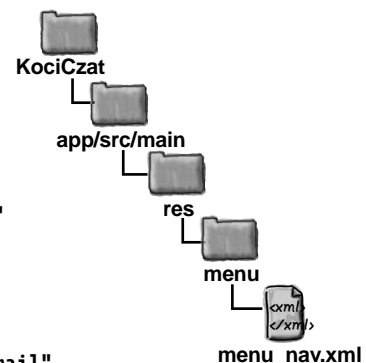
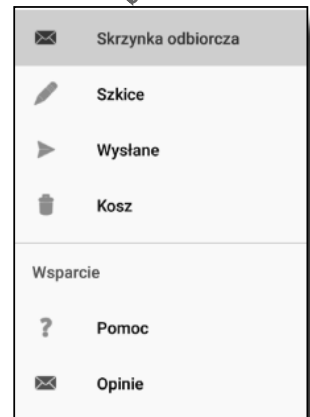
Oto kompletny kod pliku *menu_nav.xml*; zaktualizuj ten plik w swoim projekcie, tak by był identyczny z naszym:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <group android:checkableBehavior="single">
    <item
      android:id="@+id/nav_inbox"
      android:icon="@android:drawable/sym_action_email"
      android:title="@string/nav_inbox"
      android:checked="true" />
    <item
      android:id="@+id/nav_drafts"
      android:icon="@android:drawable/ic_menu_edit"
      android:title="@string/nav_drafts" />
    <item
      android:id="@+id/nav_sent"
      android:icon="@android:drawable/ic_menu_send"
      android:title="@string/nav_sent" />
    <item
      android:id="@+id/nav_trash"
      android:icon="@android:drawable/ic_menu_delete"
      android:title="@string/nav_trash" />
  </group>
  <item android:title="@string/nav_support">
    <menu>
      <item
        android:id="@+id/nav_help"
        android:icon="@android:drawable/ic_menu_help"
        android:title="@string/nav_help" />
      <item
        android:id="@+id/nav_feedback"
        android:icon="@android:drawable/sym_action_email"
        android:title="@string/nav_feedback" />
    </menu>
  </item>
</menu>
```

To są
główne
opcje
szuflady.

To jest
sekcja
wsparcia.

Kod zamieszczony na tej stronie tworzy pełne menu.



Skoro dodaliśmy już do układu menu oraz nagłówek szuflady nawigacyjnej, możemy przejść do utworzenia jej samej.

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada

Jak utworzyć szufladę nawigacyjną?

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada

Szufladę nawigacyjną tworzymy poprzez zastosowanie układu `DrawerLayout` jako głównego elementu układu aktywności. Układ `DrawerLayout` musi zawierać dwa elementy: pierwszym z nich ma być widok bądź też grupa widoków określające zawartość aktywności, a drugim widok `NavigationView`, definiujący szufladę:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
        ...
    </LinearLayout>
    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:headerLayout="@layout/nav_header"
        app:menu="@menu/menu_nav" />
</android.support.v4.widget.DrawerLayout>
```

← Układ `DrawerLayout` definiuje szufladę.

← Określamy identyfikator układu, by móc odwoływać się do niego w kodzie aktywności.

Pierwszym widokiem umieszczonym w `DrawerLayout` jest układ określający główną zawartość aktywności. Jest ona widoczna, gdy szuflada nawigacyjna jest ukryta.

← Widok `NavigationView` definiuje zawartość szuflady.

← Ten atrybut umieszcza szufladę przy początkowej krawędzi aktywności (w przypadku języków, w których piszemy od lewej do prawej, będzie to lewa krawędź ekranu).

← To jest układ nagłówka szuflady.

← A to plik zasobu menu definiującego opcje szuflady.

Wygląd szuflady nawigacyjnej określają dwa kluczowe atrybuty elementu `<NavigationView>`: `headerLayout` oraz `menu`.

Atrybut `app:headerLayout` określa układ, który powinien zostać zastosowany jako nagłówek szuflady (w naszym przykładzie jest to plik `nav_header.xml`). Jest to atrybut opcjonalny.

Z kolei atrybut `app:menu` służy do określania pliku zasobu menu zawierającego opcje szuflady (w naszej aplikacji jest to plik `menu_nav.xml`). Jeśli pominiemy ten atrybut, szuflada nawigacyjna nie będzie zawierać żadnych elementów.

Kompletny kod układu aktywności activity_main.xml

- Fragmenty i aktywności
- Nagłówkek
- Opcje
- Szuflada

Dodamy teraz do układu aktywności MainActivity szufladę nawigacyjną, korzystającą z przygotowanego wcześniej układu nagłówka oraz pliku zasobu menu. Główna zawartość układu będzie się składać z paska narzędzi oraz układu FrameLayout. Tego układu użyjemy w dalszej części rozdziału do wyświetlania fragmentów.

Poniżej przedstawiliśmy kod pliku activity_main.xml; zaktualizuj zawartość tego pliku w swoim projekcie, tak by była identyczna z tą ukazaną przez nas:

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout ← Głównym elementem układu jest DrawerLayout.
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout" ← Układ ma zdefiniowany identyfikator,
    android:layout_width="match_parent"      dzięki czemu będziemy mogli odwoływać
    android:layout_height="match_parent" >   się do niego w kodzie aktywności.

    <LinearLayout ← Ten fragment definiuje główną zawartość aktywności.
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

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

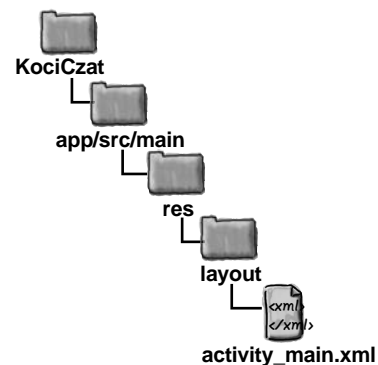
        <FrameLayout
            android:id="@+id/content_frame"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

    </LinearLayout>

    <android.support.design.widget.NavigationView ←
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:headerLayout="@layout/nav_header"
        app:menu="@menu/menu_nav" /> ← Tu używamy utworzonych wcześniej
</android.support.v4.widget.DrawerLayout>   plików zasobów: układu określającego
                                              postać nagłówka oraz menu definiującego
                                              zawartość listy opcji.

```

Na główną zawartość aktywności składają się komponent Toolbar oraz układ FrameLayout, w którym będziemy wyświetlali fragmenty.



Zanim uruchomimy aplikację, by przekonać się, jak wygląda nasza szuflada nawigacyjna, zmienimy jeszcze kod aktywności MainActivity, aby bezpośrednio po utworzeniu aktywności wyświetlany był w niej fragment InboxFragment.

Dodanie fragmentu InboxFragment do układu aktywności MainActivity

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada

Tworząc plik zasobu menu, zdefiniowaliśmy opcję skrzynki odbiorczej jako domyślnie wyróżnioną. Dlatego powinniśmy teraz wyświetlić fragment InboxFragment podczas tworzenia aktywności, tak by jej zawartość odpowiadała zaznaczonej opcji szuflady nawigacyjnej. Dodatkowo użyjemy paska narzędzi jako paska aplikacji aktywności, tak by został na nim wyświetlony tytuł aplikacji.

Poniżej przedstawiliśmy naszą wersję pliku *MainActivity.java*; zmodyfikuj ten sam plik w swoim projekcie, by był identyczny z naszym:

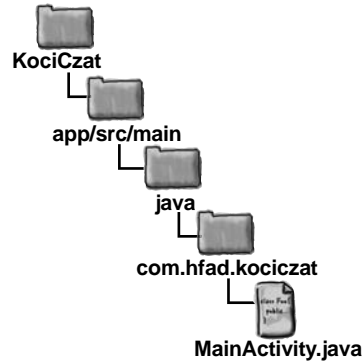
```
package com.hfad.kociczat;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentTransaction;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar); ← Ustawiamy komponent Toolbar jako pasek aplikacji aktywności.

        Fragment fragment = new InboxFragment();
        FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
        ft.add(R.id.content_frame, fragment);
        ft.commit();
    }
}
```



Upewnij się, że aktywność dziedziczy po klasie *AppCompatActivity*, gdyż używamy motywu *AppCompatActivity* oraz fragmentów z biblioteki wsparcia.

Stosujemy transakcję fragmentu, by wyświetlić instancję *InboxFragment*.

A teraz sprawdźmy, co się stanie po uruchomieniu aplikacji.



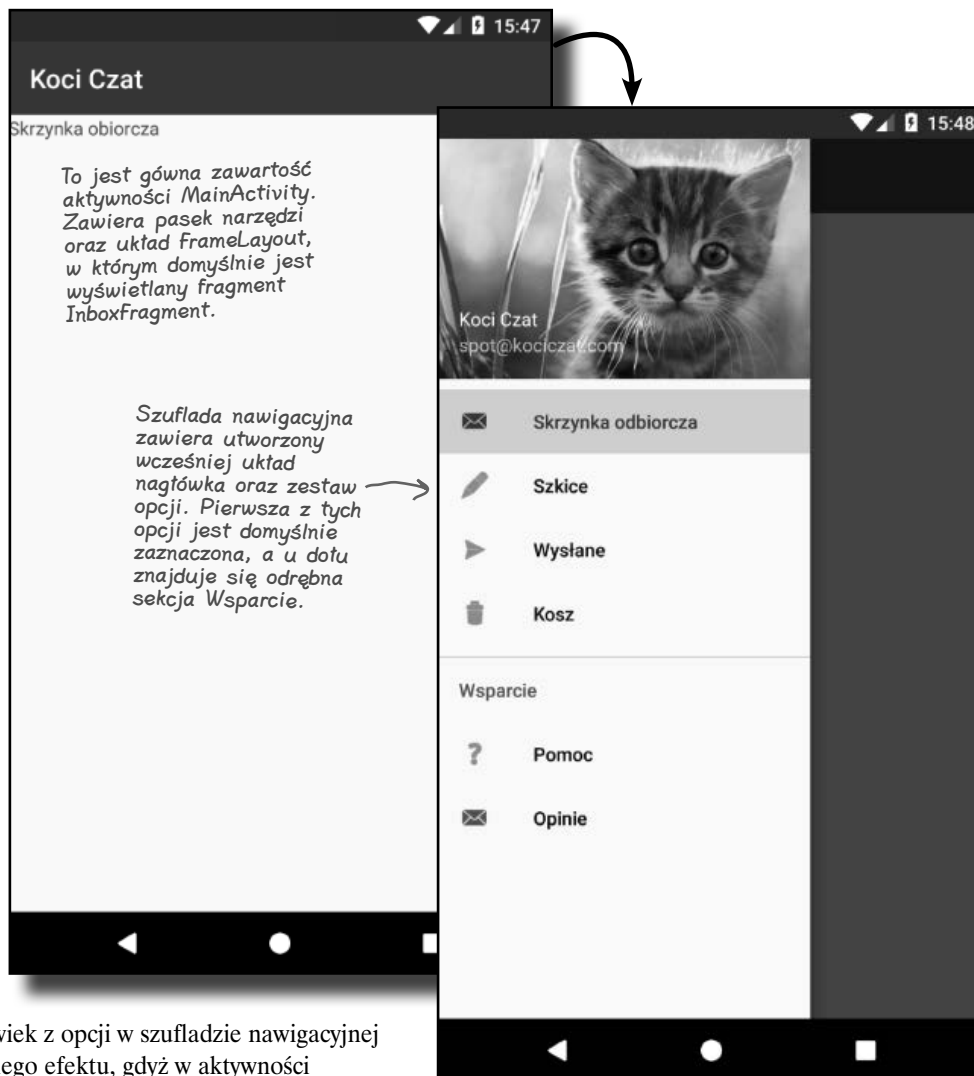
Jazda próbna aplikacji

Po uruchomieniu aplikacji w aktywności `MainActivity` zostaje wyświetlony fragment `InboxFragment`. Kiedy wykonamy gest przeciągnięcia od lewej krawędzi ekranu do jego środka (przynajmniej w przypadku języków takich jak polski, w których piszemy od strony lewej do prawej), zostanie wyświetlona szuflada nawigacyjna. Szuflada ta zawiera układ nagłówka oraz listę opcji zdefiniowaną w pliku zasobu menu. Pierwsza z tych opcji jest automatycznie wyróżniona:

Szuflady nawigacyjne

- Fragmenty i aktywności
- Nagłówkek
- Opcje
- Szuflada

← W językach, w których pisze się od strony prawej do lewej, szuflada jest wyświetlana przy prawej krawędzi ekranu.



Kliknięcie którejkolwiek z opcji w szufladzie nawigacyjnej nie daje jeszcze żadnego efektu, gdyż w aktywności `MainActivity` nie napisaliśmy jeszcze żadnego kodu, który by kontrolował działanie szuflady. Zajmiemy się tym w następnej kolejności.

Co musi robić kod aktywności?

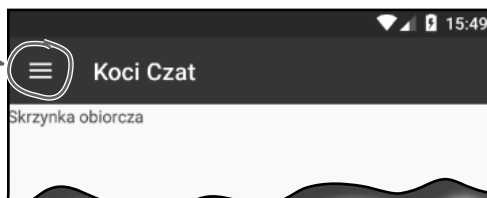
Kod aktywności musi wykonać trzy operacje:

- ☑ Fragmenty i aktywności
- ☑ Nagłówek
- ☑ Opcje
- ☐ Szuflada

1 Dodać przełącznik szuflady.

Będzie on stanowić wizualny sygnał dla użytkownika informujący o tym, że aktywność zawiera szufladę nawigacyjną. Przełącznik ten ma postać ikony „hamburgera” umieszczonej na pasku narzędzi, którą można kliknąć, aby otworzyć szufladę.

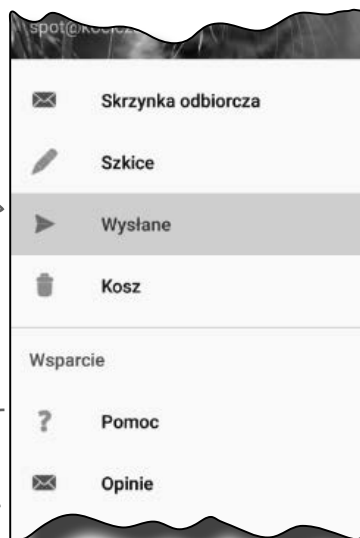
To jest ikona „hamburgera”. Jej kliknięcie powoduje wyświetlenie szuflady nawigacyjnej.



2 Zapewnić, że szuflada będzie reagować na kliknięcia.

Kiedy użytkownik kliknie jedną z opcji w szufladzie nawigacyjnej, wyświetlimy odpowiedni fragment lub odpowiednią aktywność i zamkniemy szufladę.

Kiedy użytkownik kliknie jedną z głównych opcji szuflady, wyświetlimy odpowiadający jej fragment i zamkniemy szufladę. Kiedy użytkownik ponownie otworzy szufladę, wybrana wcześniej opcja będzie wyróżniona.



Kiedy użytkownik kliknie jedną z tych opcji, uruchomimy odpowiednią aktywność.

3 Zamykać szufladę, kiedy użytkownik naciśnie klawisz Wstecz.

Jeśli szuflada będzie widoczna, to po kliknięciu przycisku *Wstecz* zamkniemy ją. Jeśli szuflada będzie zamknięta, to kliknięcie przycisku *Wstecz* da standardowe efekty.

Zacniemy od dodania przełącznika szuflady.

Dodanie przełącznika szuflady

Pierwszą rzeczą, jaką musimy zrobić, jest dodanie przełącznika szuflady, dzięki któremu będziemy mogli ją wyświetlić, klikając ikonę na pasku narzędzi.

Zacniemy od dodania dwóch zasobów łańcuchowych, których użyjemy do opisanía akcji „otwórz szufladę” oraz „zamknij szufladę”, wymaganych w celu ułatwienia dostępu. Dodaj poniższe dwa zasoby do pliku `strings.xml`:

```
<string name="nav_open_drawer">Otwórz szufladę nawigacyjną</string>
<string name="nav_close_drawer">Zamknij szufladę nawigacyjną</string>
```

Przełącznik szuflady nawigacyjnej konstruujemy w metodzie `onCreate()` aktywności, tworząc nową instancję klasy `ActionBarDrawerToggle` i dodając ją do układu `DrawerLayout`. Najpierw pokażemy Ci kod realizujący tę operację, a później, w dalszej części rozdziału, dodamy go do kodu aktywności `MainActivity`.

Konstruktor `ActionBarDrawerToggle` ma aż pięć parametrów: bieżącą aktywność, układ `DrawerLayout`, obiekt paska narzędzi oraz dwa zasoby łańcuchowe opisujące czynności otwierania i zamykania szuflady (zasoby te dodaliśmy powyżej):

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
```

...

```
DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
```

```
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(this, ← Bieżąca aktywność.
```

↑
Ta instrukcja dodaje ikonę „hamburgera” do paska narzędzi.

→ Układ `DrawerLayout` używany w aktywności.

Te łańcuchy znaków są niezbędne ze względu na wymogi dotyczące ułatwienia dostępu.

```
drawer,
toolbar, ← Pasek narzędzi aktywności.
{
R.string.nav_open_drawer,
R.string.nav_close_drawer);
```

Po utworzeniu przełącznika szuflady musimy go dodać do układu — w tym celu należy wywołać metodę `addDrawerListener()` klasy `DrawerLayout` i przekazać w jej wywołaniu utworzoną instancję przełącznika:

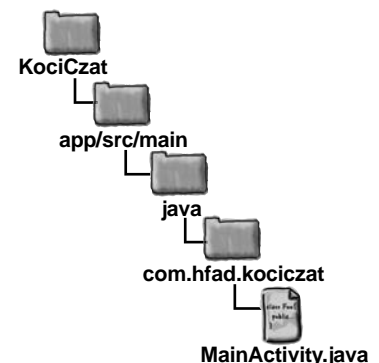
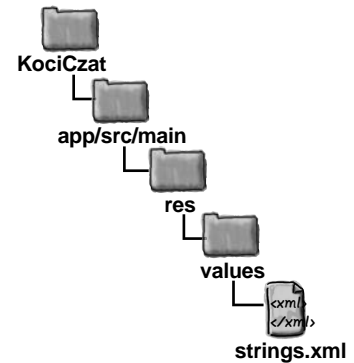
```
drawer.addDrawerListener(toggle);
```

Następnie należy jeszcze wywołać metodę `syncState()`, aby zsynchronizować stan ikony na pasku narzędzi ze stanem szuflady nawigacyjnej. To konieczne, gdyż stan ikony zmienia się, gdy klikniemy ją, by otworzyć szufladę:

```
toggle.syncState();
```

Już niebawem dodamy cały ten kod do metody `onCreate()` aktywności `MainActivity`.

- Fragmenty i aktywności
- Nagłówki
- Opcje
- Szuflada



Reagowanie na klikanie elementów szuflady

Teraz zajmiemy się zapewnieniem odpowiedniej reakcji aktywności MainActivity na klikanie opcji umieszczonych w szufladzie.

W tym celu zaimplementujemy w niej interfejs **NavigationView**.

OnNavigationItemSelectedListener. W ten sposób sprawimy, że każde kliknięcie jednej z opcji dostępnych w szufladzie nawigacyjnej spowoduje wywołanie nowej metody, `onNavigationItemSelectedListener()`, którą zaimplementujemy w aktywności MainActivity. Użyjemy tej metody, by wyświetlać ekran odpowiedni dla wybranej opcji.

W pierwszej kolejności, używając poniższego kodu, zaimplementujemy interfejs w kodzie aktywności MainActivity. W ten sposób aktywność stanie się obiektem nasłuchującym zdarzeń generowanych przez widok **NavigationView**:

```
...
import android.support.design.widget.NavigationView;

public class MainActivity extends AppCompatActivity
    implements NavigationView.OnNavigationItemSelectedListener {
    ...
}
```

↑
Implementacja tego interfejsu oznacza, że aktywność może odpowiadać na kliknięcia opcji w szufladzie nawigacyjnej.

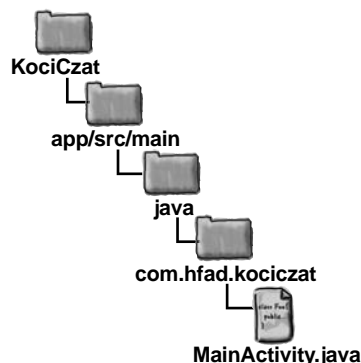
Kolejną czynnością będzie zarejestrowanie obiektu nasłuchującego, czyli aktywności MainActivity, w widoku **NavigationView**. Dzięki temu aktywność będzie informowana o tym, kiedy użytkownik kliknie jedną z opcji dostępnych w szufladzie nawigacyjnej. W tym celu w metodzie `onCreate()` aktywności musimy pobrać referencję do obiektu **NavigationView** i wywołać jego metodę `setNavigationItemSelectedListener()`:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);
}
```

↑
To wywołanie rejestruje aktywność jako obiekt nasłuchujący zdarzeń generowanych przez widok **NavigationView**, dzięki czemu zostanie ona poinformowana, kiedy użytkownik kliknie jedną z opcji dostępnych w szufladzie nawigacyjnej.

I w końcu pozostaje nam zaimplementowanie metody `onNavigationItemSelectedListener()`.

- Fragmenty i aktywności
- Nagłówkek
- Opcje
- Szuflada



Implementacja metody `onNavigationItemSelectedListener()`

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szufłada

Metoda `onNavigationItemSelectedListener()` jest wywoływana, kiedy użytkownik kliknie jeden z elementów wyświetlonych w szufładzie nawigacyjnej. Ma ona jeden parametr — obiekt `MenuItem`, reprezentujący kliknięty element menu — i zwraca wartość logiczną określającą, czy dany element szufłady należy wyróżnić:

```
@Override
public boolean onNavigationItemSelectedListener(MenuItem item) {
    // kod obsługujący kliknięcie elementu szufłady
}
```

Ta metoda zostaje wywołana po każdym kliknięciu elementu szufłady nawigacyjnej. Jej parametrem jest kliknięty element.

W naszej aplikacji kod w tej metodzie musi wyświetlać odpowiedni ekran, odpowiadający klikniętemu elementowi szufłady. Jeśli element szufłady będzie powiązany z aktywnością, to musimy ją uruchomić przy użyciu intencji. Jeśli natomiast kliknięty element jest powiązany z fragmentem, to wystarczy, że wyświetlimy go w układzie `FrameLayout` aktywności `MainActivity`, używając do tego transakcji fragmentu.

Ogólnie rzecz biorąc, w przypadku wyświetlania fragmentów w reakcji na klikanie opcji w szufładzie nawigacyjnej zazwyczaj nie umieszcza się przy tym transakcji na stosie cofnięć, jak robiliśmy to wcześniej. Wynika to z tego, że klikając przycisk *Wstecz*, użytkownik nie będzie oczekiwał wyświetlania kolejno fragmentów odpowiadających wszystkim wybranym wcześniej opcjom szufłady nawigacyjnej. Dlatego przeważnie używany jest kod taki jak ten przedstawiony poniżej:

```
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
ft.replace(R.id.content_frame, fragment);
ft.commit();
```

← To jest ten sam kod transakcji fragmentu, którego używaliśmy wcześniej, z tą różnicą, że nie dodajemy transakcji do stosu cofnięć aktywności.

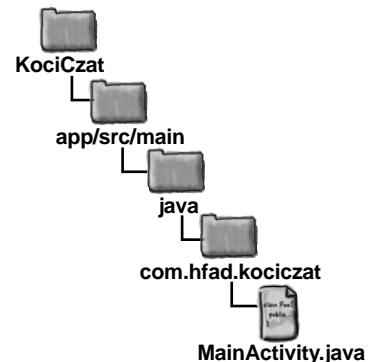
I w końcu musimy zamknąć szufładę. W tym celu musimy pobrać referencję do układu `DrawerLayout` i wywołać metodę `closeDrawer()` uzyskanego obiektu:

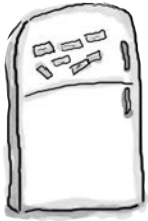
```
DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
drawer.closeDrawer(GravityCompat.START);
```

← Używamy stałej `GravityCompat.START`, gdyż umieszciliśmy szufładę przy początkowej krawędzi aktywności. Gdybyśmy umieszcili szufładę przy końcowej krawędzi, musieliśmy użyć stałej `GravityCompat.END`.

Wykonanie powyższego fragmentu kodu spowoduje wysunięcie szufłady poza początkową krawędź aktywności.

Teraz już wiesz wszystko, co trzeba wiedzieć, żeby napisać kod metody `onNavigationItemSelectedListener()`. Napiszesz tę metodę w ramach ćwiczenia zamieszczonego na kilku kolejnych stronach.





Magnesiki z kodem

Po kliknięciu przez użytkownika któregoś z elementów w szufladzie nawigacyjnej chcemy wyświetlić ekran skojarzony z tym elementem. Jeśli element będzie skojarzony z fragmentem, wyświetlimy go w układzie `content_frame`. Jeśli element będzie skojarzony z aktywnością, to będziemy chcieli ją uruchomić. Na końcu musimy zamknąć szufladę nawigacyjną.

Ciekawe, czy będziesz potrafił uzupełnić kod zamieszczony na tej oraz następnej stronie. Nie będziesz musiał użyć wszystkich dostępnych magnesików.

```
@Override

public boolean onNavigationItemSelected(MenuItem item) {

    int id = item. .... ;

    Fragment fragment = null;

    Intent intent = null;

    switch( ..... ){

        case R.id.nav_drafts:

            fragment = ..... ;

            ..... ;

        case R.id.nav_sent:

            fragment = ..... ;

            .....;

        case R.id.nav_trash:

            fragment = ..... ;

            ..... ;

        case R.id.nav_help:

            intent = new Intent( ..... , ..... );

            ..... ;
```

```

case R.id.nav_feedback:
    intent = new Intent( ..... , ..... );
    .....;

default:
    fragment = ..... ;

}

if (..... != null) {
    FragmentTransaction ft = getSupportFragmentManager(). ..... ;
    ft.replace(R.id.content_frame, ..... );
    ft. .... ;
} else {
    startActivity( ..... );
}

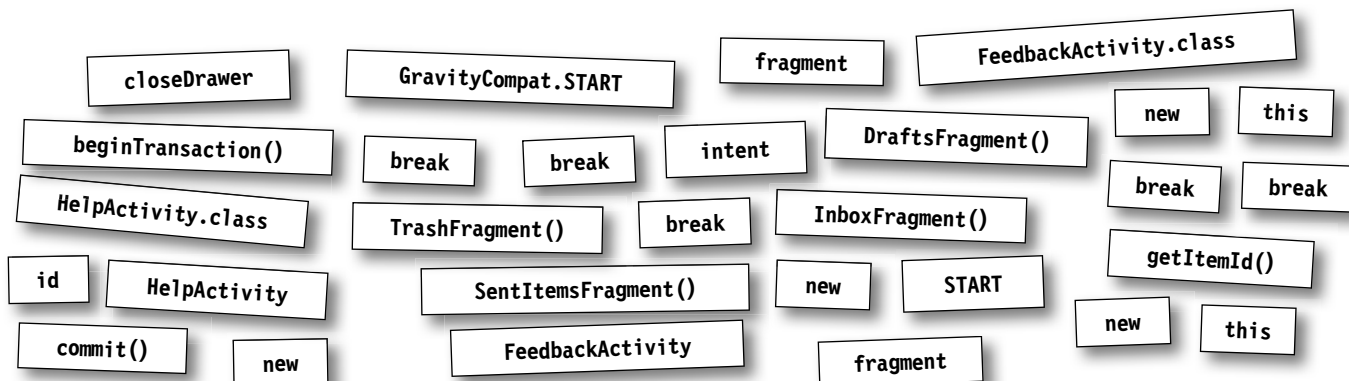
DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);

drawer. .... ( ..... );

return true;

}

```





Magnesiki z kodem. Rozwiązanie

Po kliknięciu przez użytkownika któregoś z elementów w szufladzie nawigacyjnej chcemy wyświetlić ekran skojarzony z tym elementem. Jeśli element będzie skojarzony z fragmentem, wyświetlimy go w układzie `content_frame`. Jeśli element będzie skojarzony z aktywnością, to będziemy chcieli ją uruchomić. Na końcu musimy zamknąć szufladę nawigacyjną.

Ciekawe, czy będziesz potrafił uzupełnić kod zamieszczony na tej oraz następnej stronie. Nie będziesz musiał użyć wszystkich dostępnych magnesików.

```
@Override
```

```
public boolean onNavigationItemSelected(MenuItem item) {
```

```
    int id = item. getId() ;
```

← Pobieramy identyfikator wybranego elementu.

```
    Fragment fragment = null;
```

```
    Intent intent = null;
```

```
    switch( id ){
```

```
        case R.id.nav_drafts:
```

```
            fragment = new DraftsFragment() ;
```

```
            break ;
```

```
        case R.id.nav_sent:
```

```
            fragment = new SentItemsFragment() ;
```

```
            break ;
```

```
        case R.id.nav_trash:
```

```
            fragment = new TrashFragment() ;
```

```
            break ;
```

```
        case R.id.nav_help:
```

```
            intent = new Intent( this , HelpActivity.class );
```

```
            break ;
```

← Zapisujemy w zmiennej `fragment` instancję fragmentu, który chcemy wyświetlić.

↑ Tworzymy intencję do uruchomienia aktywności `HelpActivity`, o ile została wybrana opcja `Pomoc`.

```

case R.id.nav_feedback:
    intent = new Intent( this , FeedbackActivity.class );
    break ;
default:
    fragment = new InboxFragment() ;
}
if ( fragment != null ) {
    FragmentTransaction ft = getSupportFragmentManager(). beginTransaction() ;
    ft.replace(R.id.content_frame, fragment );
    ft. commit() ;
} else {
    startActivity( intent );
}

```

Jeśli kliknięto opcję *Opinia*, musimy uruchomić intencję *FeedbackActivity*.

Domyślnie wyświetlamy fragment *InboxFragment*, gdyż jest on powiązany z pierwszą opcją w szufladzie nawigacyjnej.

Jeśli musimy wyświetlić fragment, to robimy to, używając transakcji fragmentu.

Jeśli musimy wyświetlić aktywność, uruchamiamy ją używając utworzonej wcześniej intencji

```

DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);

```

```

drawer. closeDrawer() (. GravityCompat.START );
return true;
}

```

W końcu zamykamy szufladę.

Już niebawem dodamy ten kod do pliku *MainActivity.java*.

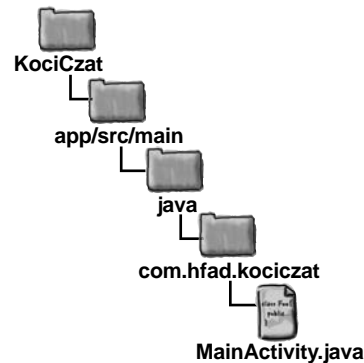


Zamknięcie szuflady po naciśnięciu przycisku *Wstecz*

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada

I w końcu przesłonimy czynności wykonywane po naciśnięciu przycisku *Wstecz*. Jeśli użytkownik naciśnie ten przycisk, kiedy szuflada nawigacyjna będzie otworzona, zamkniemy ją. Jeśli natomiast szuflada już będzie zamknięta, to naciśnięcie przycisku *Wstecz* da standardowe rezultaty.

W celu obsługi przycisku *Wstecz* zaimplementujemy w aktywności metodę `onBackPressed()`, która jest wywoływana za każdym razem, kiedy użytkownik naciśnie ten przycisk. Poniżej przedstawiliśmy kod tej metody:



```

@Override
public void onBackPressed() {
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        super.onBackPressed();
    }
}
    
```

Ta metoda jest wywoływana po naciśnięciu przycisku *Wstecz*.

Jeśli szuflada nawigacyjna jest obecnie otworzona, to ją zamykamy.

W przeciwnym razie, jeśli szuflada jest zamknięta, wywołujemy metodę `onBackPressed()` klasy bazowej.

I to już wszystkie zmiany, które należy wprowadzić w kodzie aktywności `MainActivity`. Na kilku następnych stronach przedstawimy jej pełny kod.

Nie istnieją
głupie pytania

P: Czy zawartość szuflady musi być określana przy użyciu widoku `NavigationView`?

O: Nie, jednak takie rozwiązanie jest *znacznie* prostsze. Przed udostępnieniem biblioteki wsparcia wzornictwa powszechnie stosowaną praktyką było wykorzystanie do tego widoku listy. Wciąż można stosować takie rozwiązanie, jednak wymaga ono napisania znacznie bardziej rozbudowanego kodu.

P: Czy aktywność może zawierać więcej niż jedną szufladę nawigacyjną?

O: Aktywność może zawierać po jednej szufladzie nawigacyjnej dla każdej pionowej krawędzi układu. W celu dodania drugiej szuflady do układu `DrawerLayout` trzeba dodać drugi widok `NavigationView` poniżej pierwszego.

Kompletny kod aktywności MainActivity

Poniżej przedstawiliśmy pełny kod aktywności *MainActivity.java*; zaktualizuj swoją wersję tego pliku, by była identyczna z naszą:

```
package com.hfad.kociczat;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentTransaction;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.ActionBarDrawerToggle;
import android.support.design.widget.NavigationView;
import android.view.MenuItem;
import android.content.Intent;
import android.support.v4.view.GravityCompat;
```

```
public class MainActivity extends AppCompatActivity
    implements NavigationView.OnNavigationItemSelectedListener {
```

@Override

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
```

```
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
```

```
    ActionBarDrawerToggle toggle =
```

```
        new ActionBarDrawerToggle(this,
```

```
        drawer,
```

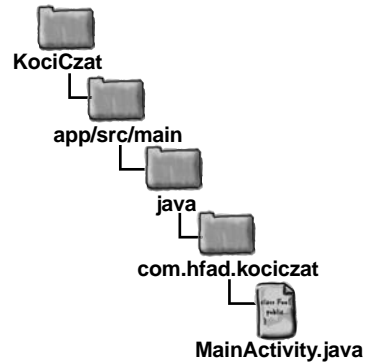
```
        toolbar,
```

```
        R.string.nav_open_drawer,
```

```
        R.string.nav_close_drawer);
```

```
    drawer.addDrawerListener(toggle);
```

```
    toggle.syncState();
```



Używamy tych dodatkowych klas, więc musimy je zaimportować.



Zaimplementowanie tego interfejsu oznacza, że aktywność może nastuchiwać zdarzeń kliknięć.

Ten fragment kodu dodaje przetącznik szufłady.

Dalsza część kodu znajduje się na następnej stronie. →

MainActivity.java (ciąg dalszy)

- ☑ Fragmenty i aktywności
- ☑ Nagłówek
- ☑ Opcje
- ☐ Szufłada

```
NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
```

```
navigationView.setNavigationItemSelectedListener(this);
```

↖ To wywołanie rejestruje aktywność w widoku NavigationView jako obiekt nasłuchujący zdarzeń.

```
Fragment fragment = new InboxFragment();
```

```
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
```

```
ft.add(R.id.content_frame, fragment);
```

```
ft.commit();
```

```
}
```

```
@Override
```

```
public boolean onNavigationItemSelected(MenuItem item) {
```

```
    int id = item.getItemId();
```

```
    Fragment fragment = null;
```

```
    Intent intent = null;
```

```
    switch(id){
```

```
        case R.id.nav_drafts:
```

```
            fragment = new DraftsFragment();
```

```
            break;
```

```
        case R.id.nav_sent:
```

```
            fragment = new SentItemsFragment();
```

```
            break;
```

```
        case R.id.nav_trash:
```

```
            fragment = new TrashFragment();
```

```
            break;
```

```
        case R.id.nav_help:
```

```
            intent = new Intent(this, HelpActivity.class);
```

```
            break;
```

```
        case R.id.nav_feedback:
```

```
            intent = new Intent(this, FeedbackActivity.class);
```

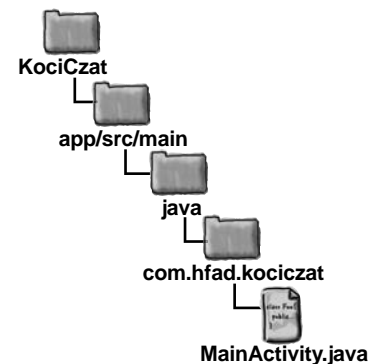
```
            break;
```

```
        default:
```

```
            fragment = new InboxFragment();
```

```
    }
```

↖ Ta metoda jest wywoływana, kiedy użytkownik kliknie jeden z elementów umieszczonych w szufładzie nawigacyjnej.



↗ Dalsza część kodu znajduje się na następnej stronie.

MainActivity.java (ciąg dalszy)

- Fragmenty i aktywności
- Nagłówki
- Opcje
- Szufłada

```

if (fragment != null) {
    FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
    ft.replace(R.id.content_frame, fragment);
    ft.commit();
} else {
    startActivity(intent);
}

DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
drawer.closeDrawer(GravityCompat.START);
return true;
}

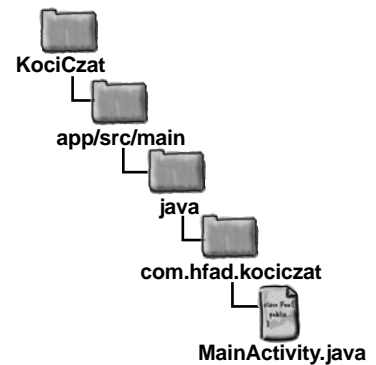
@Override
public void onBackPressed() {
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        super.onBackPressed();
    }
}
}

```

Wyświetlamy odpowiedni fragment lub odpowiednią aktywność, zależnie od opcji wybranej przez użytkownika w szufladzie nawigacyjnej.

To wywołanie zamyka szufladę po wybraniu jednej z opcji.

Kiedy użytkownik naciśnie przycisk Wstecz, zamykamy szufladę, jeśli będzie otworzona.



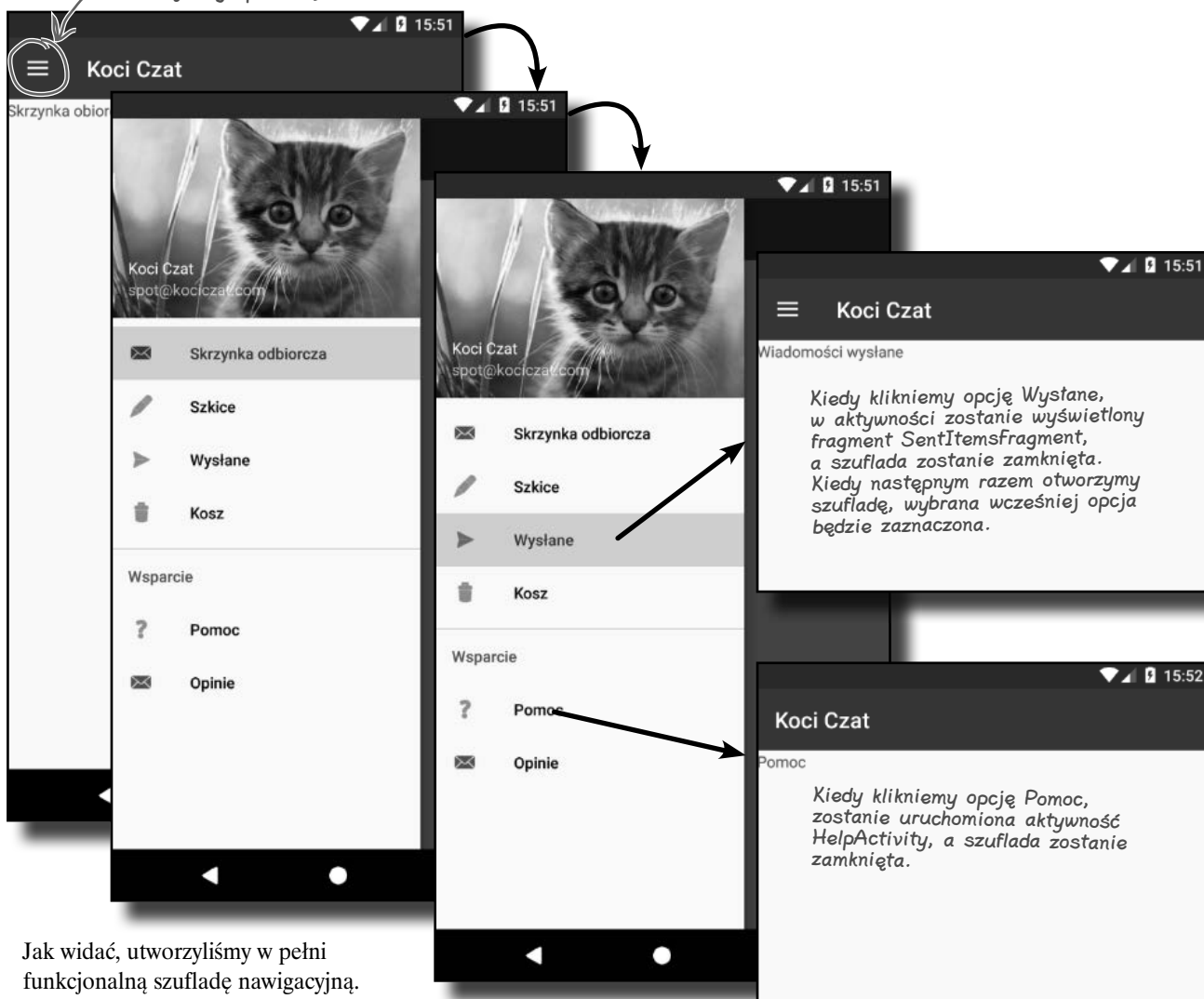
Sprawdźmy teraz, co się stanie po uruchomieniu naszej aplikacji.

Jazda próbna aplikacji

- Fragmenty i aktywności
- Nagłówek
- Opcje
- Szuflada

Po uruchomieniu aplikacji ikona przełącznika szuflady zostaje wyświetlona na pasku narzędzi. Kliknięcie tej ikony powoduje otwarcie szuflady. Kliknięcie jednej z pierwszych czterech opcji widocznych w szufladzie powoduje wyświetlenie w aktywności `MainActivity` odpowiedniego fragmentu i zamknięcie szuflady; kiedy użytkownik ponownie wyświetli szufladę, wybrana wcześniej opcja będzie wyróżniona. Kliknięcie którejś z dwóch ostatnich opcji widocznych w szufladzie powoduje uruchomienie odpowiedniej aktywności.

Aktywność `MainActivity` zawiera przełącznik szuflady. Kliknięcie go powoduje otwarcie szuflady.



Jak widać, utworzyliśmy w pełni funkcjonalną szufladę nawigacyjną.



Twój przyborek do Androida

Opanowałeś już rozdział 14.
i dodałeś do swojego przyborenika
z narzędziami szuflady nawigacyjne

Pełny kod przykładowej aplikacji prezentowanej w tym rozdziale możesz pobrać z serwera FTP wydawnictwa Helion:
`ftp://ftp.helion.pl/przyklady/andrr2.zip`



CELNE SPOSTRZEŻENIA

- Używaj szuflady nawigacyjnej, jeśli chcesz udostępnić użytkownikom znaczną liczbę skrótów lub pogrupować je w sekcje.
- Aby utworzyć szufladę nawigacyjną, dodaj do układu aktywności układ **DrawerLayout**. Pierwszym elementem tego układu musi być widok definiujący główną zawartość aktywności; zazwyczaj jest to układ z komponentami **ToolBar** i **FrameLayout**. Drugi element definiuje zawartość szuflady, przy czym zazwyczaj jest to widok **NavigationView**.
- Widok **NavigationView** pochodzi z biblioteki wsparcia wzornictwa. Obsługuje on większość możliwości funkcjonalnych szuflady.
- Nagłówek szuflady określa się, tworząc kolejny układ i podając jego identyfikator w atrybucie **headerLayout** widoku **NavigationView**.
- Elementy dodajemy do menu, tworząc zasób menu i podając jego identyfikator w atrybucie **menu** widoku **NavigationView**.
- Elementy w pliku menu dodajemy w takiej kolejności, w jakiej mają być widoczne w szufladzie.
- Jeśli chcesz wyróżnić element szuflady kliknięty przez użytkownika, to dodaj elementy menu do grupy i dodaj do niej atrybut **checkableBehavior** o wartości **"single"**.
- Użyj obiektu **ActionBarDrawerToggle**, aby wyświetlić na pasku narzędzi aktywności ikonę „hamburgera”. Stanowi ona wizualny sygnał informujący o tym, że aktywność udostępnia szufladę nawigacyjną. Kliknięcie tej ikony otwiera szufladę.
- Aby odpowiadać na kliknięcia opcji szuflady, zaimplementuj w aktywności interfejs **NavigationView.OnNavigationItemSelectedListener**. Następnie zarejestruj aktywność w widoku **NavigationView** jako obiekt nasłuchujący i zaimplementuj w niej metodę **onNavigationItemSelectedListener()**.
- Aby zamknąć szufladę, należy wywołać metodę **closeDrawer()** obiektu **DrawerLayout**.

Skorowidz

A

adapter, 247, 269, 288, 375
 ArrayAdapter, 269–271, 288, 376, 382
 CaptionedImagesAdapter, 546
 CursorAdapter, 691
 FragmentPagerAdapter, 491–493, 535
 RecyclerView, 545, 547
adaptery
 synchronizujące, 864
 tablicowe, 376
 wielokrotnego użycia, 572
akceleracja sprzętowa, 859
akcja, 77, 100, 319
 ACTION_DIAL, 101
 ACTION_SEND, 100–105, 109, 118
 ACTION_WEB_SEARCH, 101
aktualizacja
 aplikacji Trener, 398
 bazy danych, 636, 639–645, 698
 rekordów, 646
 układu, 39, 90, 704, 714, 719
aktywności, 2, 31, 97, 140, 156, 250
 cykl życia, 119
 dodawanie, 13
 domyślne, 41
 dostosowywanie, 14
 działające, 120, 138
 kategorii, 249, 250, 267
 kategorii z listą, 252
 nadrzędne, 328
 poziomu głównego, 249, 250, 252
 przeniesione do pierwszego planu, 155
 restartowanie, 146
 stany, 137
 szczegółów, 253, 279
 szczegółów/edycji, 249
 tworzenie, 146
 uruchomione, 138, 146, 364
 usunięte, 146, 155, 364
 utworzone, 364
 widoczne, 155
 widzialny czas życia, 147
 wstrzymane, 364
 wznowione, 364
 zapisywanie stanu, 428
 zatrzymane, 146, 364

aktywność, activity, 12
 AppCompatActivity, 307
 CreateMessageActivity, 79, 89, 92, 95
 DetailActivity, 347, 384, 389, 391
 DrinkActivity, 255, 277, 283, 658–661, 673, 701–703, 732
 DrinkCategoryActivity, 255, 267–278, 658, 675, 688
 FeedbackActivity, 592
 HelpActivity, 591
 MainActivity, 334, 774
 OrderActivity, 315, 317, 507, 517, 526
 OrderDetail, 507
 PizzaDetailActivity, 566–568
 ReceiveMessageActivity, 91, 97, 105
 StopwatchActivity, 130, 141, 151, 160, 435
 TempActivity, 437
 TopLevelActivity, 258, 274, 625, 658, 694, 705–712
Android, 3
 Debug Bridge, 846, 849
 SDK, 5, 23
 Studio, 5
 instalacja, 6
 Virtual Device Manager, 24
animacja
 widoków, 508, 535, 868
 właściwości, 868
API, 3
aplikacja
 dla kafeтерии Coffeina, 248, 255, 622, 625, 658, 695
 Doradca Piwny, 38
 Drogomierz, 769
 KociZcat, 583
 Komunikator, 79
 Stoper, 122
 Trener, 341, 398, 435
 Układ z ograniczeniami, 223
 Wic, 741
 Włoskie Co Nieco, 290, 295, 482, 538
aplikacje
 częściowo widoczne, 154
 interaktywne, 37

wersje na tablety, 411
wersje na telefony, 343
ART, Androidruntime, 841
atrybut
 cardCornerRadius, 578
 drawableLeft, 213
 exported, 745
 gravity, 182–184, 220
 hint, 202
 icon, 299
 id, 44, 91
 inputType, 202
 label, 299
 layout_above, 822
 layout_alignBottom, 822
 layout_alignLeft, 822
 layout_alignParentBottom, 820
 layout_alignParentEnd, 819
 layout_alignParentLeft, 820
 layout_alignParentRight, 820
 layout_alignRight, 822
 layout_alignTop, 822
 layout_behavior, 510
 layout_below, 822
 layout_centerHorizontal, 820
 layout_centerVertical, 820
 layout_column, 827
 layout_gravity, 184, 187, 193, 220
 layout_height, 44, 171, 175, 188
 layout_margin, 176
 layout_marginEnd, 176
 layout_marginLeft, 176
 layout_marginTop, 176
 layout_row, 827
 layout_scrollFlags, 510
 layout_toEndOf, 822
 layout_toLeftOf, 822
 layout_toRightOf, 822
 layout_width, 44, 175, 188
 name, 745
 onClick, 203, 452, 480
 orderInCategory, 322
 orientation, 172
 padding, 173
 paddingEnd, 173
 roundIcon, 299

- scorllbars, 554
 - text, 44, 50, 203
 - theme, 300
 - title, 322
 - atrybuty
 - padding*, 220
 - komponentu Toolbar, 519
 - układu
 - AppBarLayout, 519
 - CollapsingToolbarLayout, 519
 - widoku NestedScrollView, 519
 - automatyczne odświeżanie, 714
 - AVD, Android Virtual Device, 23
 - weryfikowanie konfiguracji, 26
 - wybór obrazu systemu, 25
 - zablokowane, 29
 - AVD Manager, 399
 - wybór komponentów, 24
 - AVD tabletu, 398, 400
 - obraz systemu, 400
 - weryfikowanie konfiguracji, 401
 - awaria aplikacji, 450
- B**
- baza danych SQLite, 621, 656
 - biblioteka Androida, 16
 - AppCompat v7, 294, 296, 306, 307, 345, 584, 756, 790
 - AWT, 4
 - cardview v7, 294
 - Constraint Layout Library, 224
 - recycleview v7, 294
 - Swing, 4
 - układu z ograniczeniami, 294
 - wsparcia wzornictwa, 294, 495–527, 533, 584
 - Blueprint, 225
 - budowniczy powiadomień, 757, 765
- C**
- cofnięcia, 414
 - cykl życia
 - aktywności, 119, 137, 146–149, 157, 165, 439
 - metody, 167
 - fragmentów, 365, 439
 - metody, 392
- D**
- Dalvik, 843
 - dane typu MIME, 101, 105
 - debugowanie USB, 109
- definiowanie
 - interfejsu, 385
 - kolorów, 304
 - kursora, 715
 - stylu, 301
 - układu FrameLayout, 193
 - układu ramki, 188
 - deklarowanie
 - aktywności, 85
 - uprawnień, 791
 - usługi, 745
 - demon, 850
 - dodanie
 - adaptera do kontrolki, 493
 - akcji, 315, 319
 - akcji do powiadomienia, 758
 - aktywności, 12, 13, 85
 - biblioteki wsparcia, 224, 345, 498, 542, 790
 - dostawcy akcji, 332
 - fragmentów, 348
 - do układu aktywności, 352, 449
 - dynamicznych, 434
 - InboxFragment, 604
 - ikony, 320
 - przycisku FAB, 527
 - kart do aktywności, 498, 500
 - klasy, 360
 - kolumn, 649
 - komponentów, 39
 - komunikatów, 743
 - kontrolki ViewPager, 490
 - łańcuchów znaków, 56, 596
 - menu, 323
 - obiektu nasłuchującego, 262
 - obrazków, 189, 211, 541 594
 - do paska narzędzi, 523
 - do przycisków, 213
 - ograniczeń, 230
 - paska narzędzi, 306, 309
 - pływających przycisków, 506
 - pola wyboru, 696
 - przełącznika szuflady, 607
 - przycisku, 43, 329, 346
 - FAB do układu, 526, 527
 - W górę
 - stopera do fragmentu, 469
 - tytułu akcji, 320
 - wagi do widoków, 180, 181
 - widoków do układu siatki, 824
 - zasobów, 320
 - łańcuchowych, 123, 225, 594
 - zwijanego paska narzędzi, 517
- dokumentacja, 5
 - dostawca
 - akcji udostępniania, 292, 331, 332
 - lokalizacji, 793
 - treści, 863
 - dostęp
 - do bazy danych, 624, 627
 - do lokalizacji, 804
 - dostosowywanie
 - aktywności, 14
 - wyglądu aplikacji, 303
 - dystrybucja, 862
 - działanie
 - aktywności, 120
 - aplikacji
 - Coffeina, 284, 736
 - Drogomierz, 783, 801
 - Stoper, 152
 - Trenażer, 368, 424
 - Wic, 762
 - Włoskie Co Nieco, 336
 - przewijania, 510
 - dziennik, 741, 743
 - logcat, 854
- E**
- edytor
 - kodu, 18, 32
 - projektu, 18, 32, 42
 - ekran
 - głównego poziomu, 290
 - kategorii, 290
 - ustawień, 867
 - element, 45
 - <Button>, 43–46, 76
 - <CheckBox>, 220, 698
 - <ConstraintLayout>, 33
 - <dimen>, 174
 - <EditText>, 80, 118, 202
 - <fragment>, 353, 392, 449, 463, 480
 - <FrameLayout>, 188
 - <HorizontalScrollView>, 215
 - <ImageView>, 220
 - <include>, 312, 314
 - <item>, 303, 332
 - <LinearLayout>, 41, 45, 90, 171, 187
 - <ListView>, 259
 - <NavigationView>, 602
 - <RadioButton>, 220
 - <ScrollView>, 215, 220
 - <service>, 745, 765
 - <Spinner>, 76, 210
 - <string-array>, 57

Skorowidz

element

- <style>, 303
 - <supported-screens>, 403
 - <Switch>, 220
 - <TextView>, 33, 36, 46, 47
 - <ToggleButton>, 204, 220
 - <uses-permission>, 791
- emulator QEMU, 23, 858
- uruchamianie aplikacji, 27
- etykieta, 318

F

- FAB, floating action button, 526
- filtr intencji, 105, 106, 117
- LAUNCHER, 437
- format
- DEX, 30, 843
 - OAT, 847
- fragment, 342, 392
- DraftsFragment, 586
 - InboxFragment, 585, 604
 - ListFragment, 374
 - PastaFragment, 487
 - PizzaFragment, 486, 553, 554, 575
 - SentItemsFragment, 587
 - StopwatchFragment, 435, 444–447, 456–460, 469
 - StoresFragment, 488
 - TopFragment, 485, 508, 514
 - TrashFragment, 588
 - WorkoutDetailFragment, 344, 348, 412, 417
 - WorkoutListFragment, 344, 372, 378, 383, 424
 - z listą, 372, 373
- fragmenty
- cykl życia, 365
 - dla większych interfejsów, 393, 403, 409–425
 - dodanie zawartości do przewijania, 512
 - dynamiczne, 433–437, 449, 461, 471, 475
 - identyfikator, 409
 - identyfikator treningu, 361
 - komunikacja z aktywnością, 384
 - metody zwrotne, 365
 - miejsce wyświetlania, 471
 - na stosie cofnięć, 473
 - programowe dodawanie, 419
 - programowe zmiany, 416
 - przewijanie, 489
 - transakcje, 472

- usuwanie, 419
- zapis stanu, 429
- zmienne lokalne, 428

G

- galeria motywów, 302
- gest przeciągania, 484, 535
- górnny wiersz układu, 240
- GPS, 793
- Gradle, 7, 833
- pliki projektu, 834
 - wtyczki, 840
 - zadanie
 - androidDependencies, 838
 - check, 837
 - clean installDebug, 837
- graficzny interfejs użytkownika, GUI, 2
- grupowanie
- elementów, 598
 - widoków, 169, 198
- GUI, graphical user interface, 2

H

- hierarchia
- klasy AppCompatActivity, 297
 - widoków, 200
 - widoku RecyclerView, 570

I

- identyfikator
- fragment_container, 417
 - klikniętego elementu, 277
 - time_view, 135
 - treningu, 361, 363
 - widoku, 241
- ikona, 320
- przycisku FAB, 527
 - ikony wbudowane, 597
- implementacja
- interfejsu, 388, 575
 - klasy, 70
 - logiki, 64
 - Material Design, 506
- informacje
- o błędzie, 450
 - o lokalizacji, 815
 - o napojach, 626, 660
 - o stronach, 491
- instalacja Android Studio, 6
- IntelliJ IDEA, 5
- intencja, 77, 86, 118, 280
- jawna, 118
 - niejawna, 101

intencje

- dodawanie informacji, 101
 - filtry, 105
 - określające
 - akcje, 101
 - treści, 333
 - tworzenie, 92
 - uruchamianie aktywności, 87, 99
 - wyznaczanie, 105
- interakcja fragmentu i aktywności, 359
- interfejs, 384, 572
- Binder, 786
 - IBinder, 771, 786
 - Listener, 388, 572
 - obiektu nasłuchującego, 385
 - OnClickListener, 455
 - programowania aplikacji, API, 3
 - użytkownika, 721, 722

J

- Java
- SE, 4
 - VM, 30
- język
- Groovy, 839
 - SQL, 631
- JVM, Java Virtual Machine, 842

K

- karta, 498, 535
- Design, 41
 - tworzenie widoku, 543
 - wyświetlenie danych pizzy, 542
- katalog
- app, 17
 - build, 17
 - databases, 623
 - java, 17, 36, 82
 - layout, 17, 36, 402
 - layout-large, 402, 408
 - main, 36
 - res, 17
 - src, 17
 - values, 174, 301
- kategoria o wartości DEFAULT, 118
- klasa, 631
- ActionBarDrawerToggle, 607
 - Activity, 21, 139, 148
 - Android.util.Log, 765
 - AppCompatActivity, 297
 - AsyncTask, 724
 - CoffeinaDatabaseHelper, 634
 - Context, 139, 376, 752

- ContextThemeWrapper, 139
 - ContextWrapper, 139, 752
 - Cursor, 624
 - Drink, 255, 256, 622, 626
 - DrinkActivity, 695, 730
 - Fragment, 366, 392
 - FragmentActivity, 354
 - FragmentPagerAdapter, 491
 - Handler, 127
 - IntentService, 742, 752, 765
 - MainActivity, 298
 - MojaAktywnosc, 139
 - MojaUsługaUruchomiona, 752
 - Object, 366
 - OdometerService, 771
 - Pasta, 562
 - Pizza, 541
 - R, 63
 - ReceiveMessageActivity, 96
 - Service, 752, 770
 - SQLiteOpenHelper, 627, 648
 - TempActivity, 438
 - TopLevelActivity, 255
 - UpdateDrinkTask, 730
 - View, 44, 48, 198
 - ViewGroup, 198
 - ViewPager, 489
 - WebView, 866
 - Workout, 360
 - WorkoutListFragment, 374, 377
 - kliknięcie, 566, 570
 - elementu listy, 386
 - przycisku, 347
 - klucz
 - główny, 630
 - RSA, 109
 - kod
 - adaptera
 - CaptionedImagesAdapter, 546
 - FragmentPagerAdapter, 492
 - aktywności, 61
 - CreateMessageActivity, 96
 - DrinkActivity, 283, 661, 669–674, 701–703, 732
 - DrinkCategoryActivity, 278, 688
 - FeedbackActivity, 592
 - MainActivity, 334, 418, 423, 494, 615–617, 781, 782
 - StopwatchActivity, 130, 142, 151, 160, 162
 - TempActivity, 438, 467
 - TopLevelActivity, 262, 263, 711, 712
 - dodający zwijany pasek narzędzi, 520
 - dołączający kontrolkę, 489
 - fragmentu, 349, 351
 - DraftsFragment, 586
 - PizzaFragment, 558
 - StopwatchFragment, 444–460
 - WorkoutDetailFragment, 430
 - klasy
 - CoffeinaDatabaseHelper, 634
 - MainActivity, 298
 - UpdateDrinkTask, 730
 - WorkoutListFragment, 377
 - pliku, *Patrz* plik
 - pomocnika SQLite, 634, 643, 651, 652
 - realizujący transakcję fragmentu, 475
 - układu, 45, 46, 192
 - activity_main.xml, 381
 - aktywności głównego poziomu, 260
 - siatki, 831, 832
 - umożliwiający przewijanie, 511
 - usługi
 - DelayedMessageService, 744, 760, 761
 - OdometerService, 795
 - wielokrotne stosowanie, 342
 - wyświetlający powiadomienia, 809, 810
 - kolumna `_id`, 681
 - kompilacja aplikacji, 837
 - komponent
 - AppBarLayout, 499
 - Plain Text, 243
 - Spinner, 38, 48
 - dodanie do odwołania, 57
 - dodanie wartości, 56
 - TabLayout, 499
 - TextView, 64
 - Toolbar, 499, 519
 - komponenty
 - GUI, 48, 198
 - Material Design, 506
 - sprzętowe, 24, 399
 - komunikacja z bazą danych, 721, 722
 - komunikat, 86, 216, 721, 722, 741, 743
 - konfiguracja
 - projektu, 9
 - urządzenia, 136
 - konsola, 28
 - konstruktor bezargumentowy, 350
 - kontrolka
 - NestedScrollView, 514, 535
 - ViewPager, 489, 490, 501, 535
 - konwersja stopera, 438, 450
 - koordynacja fragmentów, 385
 - kopiowanie plików, 855
 - kreator Create New Project, 12
 - kursor, 671, 681–723
- ## L
- lista, 251, 537
 - rozwijana, 38, 48, 210
 - z widokiem szczegółów, 384
 - logcat, 743, 854
 - logika aplikacji, 39
- ## Ł
- łańcuch znaków, 52
 - łączenie widoków, 270
- ## M
- marginesy, 176
 - przycisku, 229
 - widoku, 228
 - maszyna wirtualna
 - VirtualBox, 23
 - VMWare, 23
 - Maven, 7
 - mechanizm wnioskowania ograniczeń, 241
 - menedżer
 - GridLayoutManager, 556, 578
 - LinearLayoutManager, 556, 578
 - StaggeredGridLayoutManager, 556, 557
 - menedżery
 - fragmentów, 362
 - lokalizacji, 793
 - transakcji, 480
 - układów, 556
 - metoda
 - addToBackStack(), 432
 - bindService(), 778, 815
 - bundle.get*(), 168
 - bundle.put*(), 168
 - changeCursor(), 715, 737
 - checkSelfPermission(), 794, 800, 803, 815
 - close(), 672
 - commit(), 432
 - createChooser(), 112, 113, 117, 118
 - delete(), 647
 - displayDistance(), 780, 784
 - doInBackground(), 724–726, 731, 737
 - execSQL(), 650
 - findViewById(), 63, 76, 370, 392
 - getActivity(), 758
 - getBrands(), 71, 72
 - getChildFragmentManager(), 474
 - getChildFragmentManagerTransaction, 475

Skorowidz

metoda

getChildFragmentManager(), 474, 480
getCount(), 535
getDistance(), 772, 774, 797
getFragmentManager(), 473, 480
getInt(), 672
getIntent(), 92
getItemCount(), 547
getPageTitle(), 501
getReadableDatabase(), 691
getSelectedItem(), 69, 76
getString(), 115, 672
getStringExtra(), 96
getSupportActionBar(), 329
getSupportFragmentManager(), 362
getSystemService(), 759
getWritableDatabase(), 691
insert(), 632, 656
itemClicked(), 386, 390, 412, 424
moveToFirst(), 671
moveToLast(), 671
moveToNext(), 671
moveToPrevious(), 671
onActivityCreated(), 365, 439
onAttach(), 365, 439
onBind(), 770, 783, 787, 815
onBindViewHolder(), 550, 571
onClick(), 480
onClickDone(), 529
onClickFindBeer(), 60–63, 71, 74
onClickStop(), 455
onCreate(), 21, 61, 121, 133, 137–141, 263, 365, 439, 631, 648–656, 750, 787
onCreateOptionsMenu(), 323, 337
onCreateView(), 350, 365, 439, 392, 559
onCreateViewHolder(), 549
onDestroy(), 140, 167, 365, 439, 750, 788, 797, 815
onDestroyView(), 365, 439
onDetach(), 365, 439
onDowngrade(), 641
onFavoriteClicked(), 696, 723
onHandleIntent(), 742, 751
onItemClick(), 261
onListItemClick(), 277
onListItemClicked(), 386
onLocationChanged(), 792
onNavigationItemSelected(), 609
onOptionsItemSelected(), 324
onPause(), 155, 167, 365, 439
onPostExecute(), 724, 728, 737
onPreExecute(), 725, 737

onProgressUpdate(), 727, 737
onProviderDisabled(), 792
onProviderEnabled(), 792
onRequestPermissionsResult(), 809
onRestart(), 146, 153, 167, 439
onResume(), 155, 159, 167, 365, 439
onSaveInstanceState(), 140, 146, 168, 428, 429, 432
onSendMessage(), 88, 97, 103, 115
onServiceConnected(), 776
onServiceDisconnected(), 777
onShowDetails(), 347
onStart(), 146, 158, 167, 365, 439, 804
onStartCommand(), 750, 751, 765, 792
onStop(), 146, 155, 158, 167, 365, 439
onUnbind(), 787, 788
onUpgrade(), 640
post(), 128
postDelayed(), 128
putExtra(), 92, 118
query(), 664
removeUpdates(), 815
requestLocationUpdates(), 815
requestPermissions(), 803
runTimer(), 125, 127, 129, 133, 440
setAdapter(), 270
setContentDescription(), 212
setContentView(), 61
setDisplayHomeAsUpEnabled(), 329, 337
setImageResource(), 212
setOnClickListener(), 457
setShareIntent(), 333
setSupportActionBar(), 317
setText(), 76
setupFavoritesListView(), 709
startActivity(), 86, 121, 132
startService(), 747, 765, 786
syncState(), 607
Toast.makeText(), 216
unbindService(), 779, 815
update(), 645, 646

metody

cyklu życia
aktywności, 439
fragmentów, 392, 439
usług powiązanych, 788
prywatne, 454
zwrotne fragmentów, 365
miejsce wprowadzania, input focus, 137
modyfikacja
struktury bazy danych, 649, 645

tabel, 650, 656

tekstu, 34

motyw, 302, 590

Theme.AppCompat.Light.

DarkActionBar, 308

Theme.AppCompat.Light.

NoActionBar, 308

motywy AppCompat, 297

N

nagłówek

listy, 706, 713

szuflady nawigacyjnej, 594

narzędzia SDK, 5

narzędzie wnioskowania ograniczeń, 240

nasłuchiwanie zdarzeń, 261, 571

nawigacja, 291, 535

po aktywnościach, 250

w górę, 327

nazwa

bazy danych, 629

catalogu, 17

klasy aktywności, 85

pakietu, 9, 22, 85

projektu, 17

tabeli, 649

zasobu stylu, 300

nowy projekt, 8, 15

numer wersji

bazy danych SQLite, 629, 637

O

obiekt

aktywności, 12

Binder, 771

Bundle, 141

Loader, 864

nasłuchujący, 199, 261, 385, 797

danych o lokalizacji, 792

MainActivity, 385

OnItemClickListener, 261

ServiceConnection, 774, 775

View, 199

ViewHolder, 548, 578

Workout, 360

obraz systemu, 25

obrazki, 541

obrót ekranu, 462

zmiana konfiguracji urządzenia, 136

obsługa

kliknięcia przycisku, 347

kliknięć, 276

- powiadomień, 755
 - stopera, 125
 - transakcji fragmentu, 464
 - zdarzeń, 199, 721
 - odbiorcy komunikatów, 865
 - odczytywanie
 - wartości z kursora, 691
 - właściwości, 199
 - odłączenie aktywności od usługi, 779
 - odpowiedź użytkownika, 805
 - odświeżanie automatyczne, 714
 - odwołanie do string-array, 57
 - ograniczenie w poziomie, 227
 - okno dialogowe, 104
 - wyboru aktywności, 112, 116
 - określenie poziomu API, 10
 - opcja
 - Empty Activity, 82, 279
 - File/New/Activity, 82
 - File/Project Structure, 790
 - File/New/Layout resource, 553
 - Undo Infer Constraints, 241
 - operacje
 - asynchroniczne, 724
 - w tle, 724, 739
 - orientacja układu, 172
- P**
- pakiet Support Libraries, 294
 - panel konsoli, 28
 - parametry klasy AsyncTask, 724, 729
 - pasek
 - akcji, 293
 - aplikacji, 292, 301, 306
 - akcje, 315
 - aktywności, 313
 - dodanie akcji, 319
 - etykieta, 318
 - narzędzi, 292, 306, 309, 310, 312, 508, 589
 - dodanie obrazka, 522
 - przewijany, 512, 515
 - zwijany, 517, 535
 - snackbar, 526, 530
 - pierwsza aplikacja, 7
 - planowanie wykonania kodu, 128
 - platforma
 - Android, 3
 - SDK, 5
 - plik
 - activity_create_message.xml, 79, 80, 83, 88
 - activity_detail.xml, 355
 - activity_drink.xml, 696, 698
 - activity_drink_category.xml, 268
 - activity_find_beer.xml, 41, 44, 53
 - activity_help.xml, 591
 - activity_main.xml, 14, 17–20, 32, 189, 225, 305, 346, 381, 417, 509, 603, 773
 - activity_order.xml, 316, 520, 521, 524, 525, 528, 529
 - activity_stopwatch.xml, 123, 124
 - activity_top_level.xml, 706, 713
 - AndroidManifest.xml, 17, 36, 84, 299, 319, 568
 - build.gradle, 838
 - CaptionedImagesAdapter.java, 551, 552, 573
 - card_captioned_image.xml, 544
 - CoffemaDatabaseHelper.java, 642
 - colors.xml, 590
 - CreateMessageActivity.java, 81, 95
 - DetailActivity.java, 363, 389
 - dimens.xml, 174
 - DraftsFragment.java, 586
 - Drink.java, 274
 - DrinkActivity.java, 669, 673, 701, 732–735
 - DrinkCategoryActivity.java, 271, 681
 - FindBeerActivity.java, 67, 73
 - fragment_pizza.xml, 554
 - fragment_sent_items.xml, 587
 - fragment_top.xml, 515
 - MainActivity.java, 14, 17–22, 325, 418, 494–496, 502–504, 747, 811, 812
 - menu_nav.xml, 599, 601
 - nav_header.xml, 595
 - OdometerService.java, 796–800
 - OrderActivity.java, 317, 533
 - PastaFragment.java, 487
 - PizzaDetailActivity.java, 569
 - PizzaFragment.java, 486, 555, 558, 576
 - R.java, 63, 69, 76
 - Stopwatch.java, 438
 - StopwatchActivity.java, 126, 129, 130
 - StoreFragment.java, 488
 - strings.xml, 17, 50, 54, 76
 - TempActivity.java, 467
 - TopFragment.java, 485
 - TopLevelActivity.java, 262, 706–708, 713, 716–718
 - WorkoutDetailFragment.java, 430, 476
 - WorkoutListFragment.java, 387
 - pliki
 - .apk, 27, 845, 847
 - .class, 5
 - .class, 842
 - .dex, 843
 - .jar, 5
 - build.gradle, 834
 - graficzne, 257
 - Javy, 16
 - konfiguracyjne, 16
 - projektu, 17
 - zasobów, 16
 - kolorów, 304
 - łańcuchowych, 54
 - menu, 321
 - stylów, 301, 304
 - źródłowe, 16
 - płatności Google Play, 5
 - plywający przycisk akcji, FAB, 506, 526, 535
 - pobieranie
 - danych, 657
 - z bazy, 663
 - z intencji, 280
 - referencji
 - do adaptera kursora, 715
 - do bazy, 662
 - wartości z kursora, 672
 - wyników, 854
 - pole
 - Default Margin, 228
 - tekstowe, 38, 80, 202
 - wyboru, 206, 207, 695, 696
 - Backwards Compatibility, 40, 437, 769
 - Exported, 770
 - polecenia powłoki, 853
 - polecenie
 - adb devices, 850
 - adb logcat, 854
 - ALTER TABLE, 656
 - CREATE TABLE, 636
 - DROP TABLE, 650, 656
 - SELECT, 675
 - SQL SELECT, 691
 - pomocnik SQLite, 624, 626, 634, 644, 651–656
 - ponowne utworzenie aktywności, 135
 - powiadomienia, 755, 765, 806, 809
 - typu heads-up, 757, 765
 - powiązanie
 - aktywności z usługą, 774, 778
 - komponentu z usługą, 815
 - poziom
 - API, 10, 122
 - SDK, 223

Skorowidz

prefiks @style, 300
proces, 133, 168
 abdb, 850
program
 ANT, 7
 AVD Manager, 24
 Blueprint, 225
 Gradle, 7, 833
 Maven, 7
programowe zmieniane fragmenty, 416
prośba o uprawnienie, 802, 803
przeciąganie, 484
przejścia aktywności, 868
przekazywanie identyfikatora treningu, 361
przełącznik, 204, 205
 Pozostaw ekran włączony, 859
 szuflady, 607
przesłanie metod, 148, 549
przesunięcie, bias, 231
przewijanie, 510
 fragmentów, 489
 paska narzędzi, 507–511, 515
 treści, 513
 zagnieżdżone, 513
przycisk
 Align Left Edges, 238
 FAB, 526, 527, 535
 Finish, 41
 Infer Constraints, 240, 242, 245
 Odszukaj piwo, 59
 Run, 28
 Show Blueprint, 226
 Show Constraints, 238
 Start, 126, 134
 Stop, 126, 134
 W górę, 326, 328, 329
 Wstecz, 326, 413, 614
 Wyślij wiadomość, 86–90, 97
przyciski, 44
 dodawanie, 43
 marginesy, 229
 ograniczenie w pionie, 228
 ograniczenie w poziomie, 227
 opcji, 208, 209
 pływające, 506, 535
 przełącznika, 204
 wyświetlanie obrazu, 213
 wyświetlanie tekstu, 213
 wywołanie metody aktywności, 60
 z obrazami, 214
przywracanie starszej wersji bazy, 641
pusta aktywność, 13, 40, 188

R

reagowanie na kliknięcia, 608, 698
refaktoryzacja, 707
referencja
 do adaptera kursora, 715
 do bazy danych, 662, 670
 do listy rozwijanej, 64
 do widoku, 63
 this, 758
rejestracja
 komunikatów, 765
 obiektu nasłuchującego, 386
rodzaje aktywności, 13
rozmieszczenie widoków, 190, 227, 818, 821
rozpowszechnianie aplikacji, 862
rzutowanie, 64

S

SDK, Software Development Kit, 5
serwer adb, 850
siatka, 823
sieci, 793
słowo kluczowe
 ASC, 655
 DESC, 655
sprawdzanie
 uprawnień, 794
 w trakcie działania, 803
 odpowiedzi na prośbę, 805
SQL, Structured Query Language, 631
SQLite, 621
stany
 aktywności, 137, 364
 usług powiązanych, 787
 usług uruchomionych, 750
stoper, 435
 problem zerowania, 462
stos cofnąć, 414, 415
stosowanie
 adaptera SimpleCursorAdapter, 681, 687
 dostawcy akcji, 331
 intencji, 87
 motywów, 300
 stylu, 300
 zasobów łańcuchowych, 50, 52
struktura katalogów, 16, 404, 406
styl, 300
 AppTheme, 590
szablon
 Empty Activity, 13, 40

Fragment (Blank), 374

Fragment (List), 374

szuflady nawigacyjne, 580, 591, 611, 616
grupowanie elementów, 598
nagłówki, 583, 594
opcja Skrzynka odbiorcza, 585
opcje, 583, 597
przełącznik, 606
reagowanie na klikanie, 608
reakcja na kliknięcia, 606
sekcja wsparcia, 600
tworzenie, 583, 602
używanie, 619
zamknięcie, 606, 614

Ś

środowisko

programistyczne, 4, 5
uruchomieniowe, 3
 ART, 30, 841
 Dalvik, 30

T

tabela, 630
 Drink, 626, 633, 656
tablet, 397
 zmiana orientacji, 427
tablica, 270
 łańcuchów znaków, 76
 obiektów Drink, 272
telefon, 396
test
 aplikacji, 75
 klasy, 70
 stopera, 462
testy
 jednostkowe, 870
 na urządzeniu, 871
 zautomatyzowane, 870
tosty, 216
transakcje, 415
 dodawanie, 432
 fragmentów, 463, 464, 472
 określanie zmian, 420
 rozpoczęcie, 419
 usuwanie, 432
 zagnieżdżone, 433, 474
 zastępowanie, 432
 zatwierdzenie, 421
tworzenie
 adaptera widoku, 545
 aktywności

- FeedbackActivity, 592
- HelpActivity, 591
- PizzaDetailActivity, 567
- aplikacji, 8
- AVD, 24–26
 - tabletu, 400
- bazy danych, 624–627
- drugiej aktywności, 82
- fragmentu
 - SentItemsFragment, 587
 - TrashFragment, 588
- intencji, 92, 101
 - niejawnej, 102
- interaktywnych aplikacji, 37
- kursora, 683
- menedżera lokalizacji, 793
- nagłówka szuflady nawigacyjnej, 583, 594
- nowego projektu, 8
- obiektu ServiceConnection, 775
- opcji szuflady nawigacyjnej, 583
- pomocnika SQLite, 628
- projektu, 40
- szkicu układu, 225
- szuflady nawigacyjnej, 583, 602
- tabeli, 626
- transakcji
 - fragmentu, 419
 - zagnieżdżonych, 474
- układu siatki, 825
- widoku karty, 543
- własnej klasy, 70
- zasobu łańcuchowego, 51
- zasobu tablicowego, 56
- typ
 - danych, 630
 - BLOB, 630
 - INTEGER, 630
 - NUMERIC, 630
 - REAL, 630
 - TEXT, 630
- typy
 - nawigacji, 291
 - usług, 740
- U**
- udostępnianie aplikacji, 862
- układ, layout, 2, 12, 31, 32
 - AppBarLayout, 499, 510, 519
 - CollapsingToolbarLayout, 519
 - CoordinatorLayout, 508, 509, 535
 - ConstraintLayout, 228
 - FrameLayout, 170, 188, 193, 416, 464
 - GridLayout, 823, 824
 - layout-large, 409
 - LinearLayout, 170
 - RelativeLayout, 818
 - StopwatchFragment, 447, 448
 - TabLayout, 535
- układy
 - aktywności
 - dodawanie fragmentu, 352
 - głównego poziomu, 258
 - kart, 580
 - liniowe, 170, 178, 186, 220
 - wyświetlanie widoku, 175
 - nadrzędne, 820
 - orientacja, 172
 - ramki, 169, 220
 - siatki, 817, 823, 825, 831
 - względne, 817
 - z ograniczeniami, 223, 245
 - zagnieżdżone, 191, 222
- uprawnienie, 767, 769, 777, 787, 791–794
 - ACCESS_FINE_, 815
 - ACCESS_FINE_LOCATION, 791, 797
- uproszczenie układu, 353
- uruchamianie
 - aktywności, 87, 89, 99
 - innych aplikacji, 99
 - aplikacji, 23, 30, 88, 163
 - na prawdziwym urządzeniu, 110
 - w emulatorze, 27
 - błyskawiczne, 860
 - powłoki, 852
- urządzenia
 - wygląd aplikacji, 340
 - wirtualne, *Patrz* AVD
- usługa, 739, 765
 - DelayedMessageService, 748
 - DelayedMessageService, 741, 744, 746, 756, 759–761
 - obsługi powiadomień, 755
 - OdometerService, 770, 783, 792–797, 802
- usługi, services, 740
 - lokalizacyjne, 789, 792
 - metody cyklu życia, 752
 - powiązane, 740, 767–769, 777, 786
 - metody cyklu życia, 788
 - stany, 787
 - systemowe, 759
 - uruchomione, 740, 747–753, 759, 765, 786
 - stany, 750
 - usunięte, 751
 - zaplanowane, 740
- usprawnianie aplikacji, 31
- ustawianie właściwości, 199
- usunięcie
 - atrybutu onClick, 453
 - działającej aktywności, 135
 - paska aplikacji, 306, 308
 - rekordów, 647
 - tabeli, 650, 656
- W**
- waga, weight, 179
 - widoku, 179
- wątek
 - działający w tle, 721
 - główny, 168, 720
 - wyświetlania, 720
- wersja aplikacji
 - na tablety, 397
 - na telefony, 396
- wersje Androida, 11
- wiązanie układu z aktywnością, 39
- widok, 169
 - CardView, 540–543, 578
 - ListView, 251, 259, 270
 - NavigationView, 614
 - NestedScrollView, 513, 519
 - RecyclerView, 539–542, 553, 562, 570, 578
- widoki
 - dopasowane do ograniczeń, 230, 233
 - dopasowane do zawartości, 232
 - identyfikator fragment_container, 418
 - kart, card views, 537
 - kolejność wyświetlania, 187, 190
 - list, 247
 - obrazów, 211, 212
 - ograniczenia, 230
 - położenie, 199
 - położenie zawartości, 187
 - proporcje, 233
 - przesuwanie, 231
 - przewijanie, 215
 - ramek, 190
 - rozmieszczanie, 227, 818, 821
 - stała wielkość, 232
 - tekstowe, 201
 - ukryte, 199
 - ustawianie właściwości, 199
 - w górnym wierszu, 240
 - w siatce, 823
 - wagi, 187
 - wielkość, 199, 232
 - wypełnienie danymi, 281

Skorowidz

- usługi, services
 - wyśrodkowane, 230, 238
 - zmiana marginesów, 228
 - zmiana rozmiaru, 179
 - widżet Button, 226
 - widżety aplikacji, 869
 - wielokrotne stosowanie kodu, 342
 - wirtualna maszyna Javy, JVM, 842
 - wirtualne urządzenie, *Patrz* AVD, 23
 - własna klasa Javy, 70
 - włączanie nawigacji w górę, 327
 - wsparcie dla Androida, 5
 - wstawianie rekordów, 633
 - wybieranie
 - aktywności, 112, 114
 - komponentów sprzętowych, 399
 - obrazu systemu, 25, 400
 - wygląd
 - akcji, 322
 - aplikacji, 303
 - wyjątek
 - ActivityNotFoundException, 103
 - CalledFromWrongThreadException, 127
 - wykrywanie urządzenia, 109
 - wyliczenie przebytego dystansu, 796
 - wypełnienie, 173
 - widoku listy, 719
 - wysyłanie
 - powiadomień, 759
 - wiadomości, 98, 99
 - wyświetlenie
 - danych kategorii, 267
 - danych pizzy, 542
 - dystansu, 780
 - fragmentu, 378, 471
 - komunikatu, 721, 743, 755–759
 - w dzienniku, 741
 - w obszarze powiadomień, 741
 - listy opcji, 259
 - MainActivity, 470
 - ograniczeń, 238
 - okna dialogowego, 115
 - powiadomienia, 806, 809
 - widoku, 175
 - zależności, 838
 - wywoływanie metod
 - aktywności, 60
 - fragmentu, 452
 - wyznaczanie intencji, intent resolution, 105
- ## Z
- zadania asynchroniczne, 693, 697, 705, 709, 715, 721, 723
 - zagnieżdżanie
 - fragmentów, 433
 - transakcji, 474
 - układów, 191, 222
 - zamknięcie
 - szuflady, 614
 - bazy danych, 672, 682
 - kursora, 672, 682
 - serwera adb, 854
 - zapis stanu, 140
 - aktywności, 168, 428
 - fragmentu, 429
 - zarządzanie
 - bazą danych, 627
 - fragmentami, 362
 - zasoby, 2
 - graficzne, 257
 - kolorów, 304
 - łańcuchowe, 50, 76
 - menu, 321
 - stylów, 304
 - tablicowe, 56
 - wymiaru, 174
 - zastosowanie akcji, 102
 - zdarzenie, 199, 261
 - przewijania, 510
 - zintegrowane środowisko programistyczne, IDE, 5
 - zmiana
 - marginesów widoku, 228
 - nazwy pakietu, 40
 - nazwy tabeli, 649
 - orientacji urządzenia, 427, 462
 - paska aplikacji, 299
 - położenia widoku, 231
 - rozmiaru widoku, 179
 - stanu aktywności, 154
 - sygnatury metody, 454
 - tytułu aktywności, 317
 - wielkości widoku, 199, 232
 - zmienne lokalne, 428
 - znacznik, *Patrz* element
 - znak
 - kropki, 85
 - pionowej kreski, 183
 - zrzut stanu emulatora, 859
 - zwijany pasek narzędzi, 517, 518, 535
 - zwracanie
 - wierszy tabeli, 664
 - wierszy w kolejności, 655
 - wybranych rekordów, 656
- ## Ż
- żądanie aktualizacji, 794

PROGRAM PARTNERSKI

— GRUPY 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.hellon.pl>

GRUPA
Helion 

Rusz głową!

Android

Programowanie aplikacji



Od poprzedniego wydania tej książki minęło parę lat, a kariera Androida wciąż jest dynamiczna! Kompleksowość, otwarty kod źródłowy, modułowa architektura, znakomita elastyczność — to wszystko sprawia, że lawinowo rośnie rzesza ludzi, którzy wybierają właśnie tę platformę. Liczbę urzędzeń pracujących pod kontrolą Androida podaje się w miliardach, a najpewniej będzie ich o wiele więcej. To nie tylko telefony, komputery i tablety, ale także telewizory, inteligentne lodówki czy pralki, a nawet sztuczne satelity. Umiejętność efektywnego programowania dla Androida i dobry pomysł na świetną aplikację skazują dewelopera na sukces!

Sięgnij po ten nietypowy podręcznik! Możesz uznać jego formę i sposób przekazywania treści za dziwny, ale prędko się przekonasz, że jest wyjątkowo skuteczny. Twój mózg się zaangażuje i błyskawicznie przyswoi techniki programowania dla Androida. A to wszystko dzięki nowatorskiemu podejściu autorów, którzy uznali, że najszybciej uczy się wtedy, gdy uwzględniamy specyfikę działania własnego mózgu! Dowiesz się, jak przygotować sobie warsztat pracy, czyli Android Studio. Od razu zaprojektujesz strukturę aplikacji i zbudujesz dobry interfejs. Będziesz swobodnie posługiwać się aktywnościami, intencjami, usługami. Poznasz narzędzia: Gradle, ART i ADB, dowiesz się, jak wykorzystywać bazy danych SQLite. A potem będzie jeszcze ciekawiej...

W tej książce znajdziesz między innymi:

- zasady tworzenia aplikacji interaktywnych
- istotne koncepcje, w tym: aktywności, intencje, usługi, układy i fragmenty
- biblioteki wsparcia, zadania asynchroniczne
- bazy danych i kursory
- uprawnienia i zarządzanie uprawnieniami

**Neurony płoną.
Emocje szaleją.
Oto powstaje apka
dla Androida!**

Dawn Griffiths i David Griffiths są z wykształcenia matematykami, zawodowo — znakomitymi programistami o ogromnym doświadczeniu i autorami kilku książek z serii *Rusz głową!* — a prywatnie — dobranym małżeństwem. Dawn w wolnych chwilach zastanawia się nad kolejną książką, doskonalili tai-chi, biega i podróżuje. David niegdyś pracował jako instruktor zwinnych metod programowania. Dziś wolne chwile najbardziej lubi spędzać z żoną.

	<p>Sprawdź nasze szkolenia!</p> <p>SZKOLENIA</p>  <p>AKADEMIA IT & BUSINESS</p>	<p>KOD KORZYŚCI Sięgnij po więcej! ▶</p> 
<p>helion.pl</p> <p>HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl</p>	<p>WWW.SZKOLENIA.HELION.PL</p>	<p>ISBN 978-83-283-4079-4</p>  <p>9 788328 340794</p>
<p>INFORMATYKA W NAJLEPSZYM WYDANIU</p>		<p>Cena: 119,00 zł</p>