

Rusz głową!

Android

Programowanie aplikacji



Projektuj aplikacje,
które będą hitami
sprzedaży



Unikaj
zawstydzających
aktywności

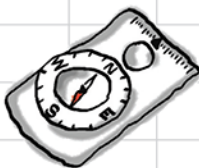


Przekonaj się, jak
Material Design
może zmienić
Twoje życie



Opanuj pojęcia
nie z tej
ziemi

Podłącz
się do usług
lokalizacyjnych
Androida



Zabaw się
z bibliotekami
Support
Libraries

Dawn Griffiths & David Griffiths

Tytuł oryginału: Head First Android Development

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-2063-5

© 2016 Helion SA.

Authorized Polish translation of the English edition of Head First Android Development, 9781449362188 © 2015 David Griffiths and Dawn Griffiths.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

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

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

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

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

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

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/andrrg>

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	xxiii
1	Zaczynamy. <i>Skok na głęboką wodę</i>	1
2	Tworzenie interaktywnych aplikacji. <i>Aplikacje, które coś robią</i>	39
3	Wiele aktywności i intencji. <i>Jakie są Twoje intencje?</i>	73
4	Cykl życia aktywności. <i>Była sobie aktywność</i>	115
5	Interfejs użytkownika. <i>Podziwiał widoki</i>	163
6	Widoki list i adaptery. <i>Zorganizuj się</i>	227
7	Fragmenty. <i>Zadbaj o modularyzację</i>	269
8	Fragmenty zagnieżdżone. <i>Zadbaj o potomstwo</i>	325
9	Paski akcji. <i>Na skrót</i>	365
10	Szuflady nawigacyjne. <i>Z miejsca na miejsce</i>	397
11	Bazy danych SQLite. <i>Odpal bazę danych</i>	437
12	Kursory i zadania asynchroniczne. <i>Nawiązywanie połączenia z bazą danych</i>	471
13	Usługi. <i>Do usług</i>	541
14	Material Design. <i>W materialistycznym świecie</i>	597
A	ART. <i>Środowisko uruchomieniowe Androida</i>	649
B	ADB. <i>Android Debug Bridge</i>	653
C	Emulator. <i>Emulator Androida</i>	659
D	Pozostałości. <i>Dziesięć najważniejszych zagadnień (których nie opisaliśmy)</i>	663

Spis treści (z prawdziwego zdarzenia)

W

Wprowadzenie

Twój mózg jest nastawiony na Androida. Jesteś tu po to, by się czegoś nauczyć, natomiast 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: 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?

	Dla kogo jest przeznaczona ta książka?	xxiv
	Wiemy, co sobie myślisz	xxv
	Wiemy, co sobie myśli Twój mózg	xxv
	Metapoznanie — myślenie o myśleniu	xxvii
	Oto co MY zrobiliśmy	xxviii
	Przeczytaj to	xxx
	Zespół recenzentów technicznych	xxxii
	Podziękowania	xxxiii

Zaczynamy

1

Skok na głęboką wodę

Android błyskawicznie podbił świat.

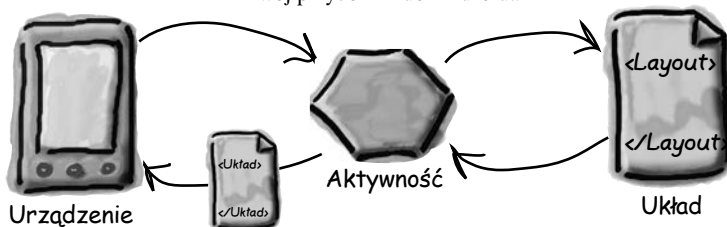
Każdy chce mieć smartfon lub tablet, a urządzenia z Androidem są niezwykle popularne.

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
Środowisko programistyczne	5
Zainstaluj Javę	6
Stwórzmy prostą aplikację	8
Aktywności z wysokości 15 tysięcy metrów	12
Tworzenie aplikacji (ciąg dalszy)	13
Tworzenie aplikacji (ciąg dalszy)	14
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
Uruchomienie aplikacji w emulatorze	27
Postępy możesz obserwować w konsoli	28
Jazda próbna	29
Ale co się właściwie stało?	30
Usprawnianie aplikacji	31
Czym jest układ?	32
Plik activity_main.xml zawiera dwa elementy	33
Plik układu zawiera odwołanie do łańcucha, a nie sam łańcuch znaków	34
Zajrzyjmy do pliku strings.xml	35
Weź swoją aplikację na jazdę próbą	37
Twój przyborek do Androida	38



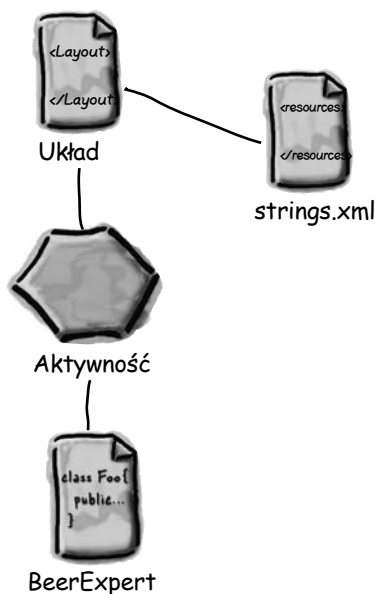
Tworzenie interaktywnych aplikacji

Aplikacje, które coś robią

2

Większość aplikacji musi w jakiś sposób reagować na poczynania użytkowników.

Z tego rozdziału dowiesz się, co zrobić, aby Twoje aplikacje były **nico 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 **nico dokładniej, jak naprawdę działa Android** — poznasz plik **R**, czyli ukryty klejnot, który spaja pozostałe elementy aplikacji.



W tym rozdziale napiszemy aplikację Doradca piwny	40
Utworzenie projektu	42
Utworzyliśmy domyślną aktywność i układ	43
Dodawanie komponentów w edytorze projektu	44
Plik activity_find_beer.xml zawiera nowy przycisk	45
Zmiany w kodzie XML układu...	48
...są uwzględniane w edytorze projektu	49
Stosuj zasoby łańcuchowe, a nie łańcuchy podawane w kodzie	50
Zmiana układu i zastosowanie w nim zasobów łańcuchowych	51
Weź swoją aplikację na jazdę próbną	52
Dodanie wartości do komponentu Spinner	53
Dodanie do komponentu Spinner odwołania do string-array	54
Jazda próbna komponentu Spinner	54
Musimy zadbać o to, by przycisk coś robił	55
Niech przycisk wywołuje metodę	56
Jak wygląda kod aktywności?	57
Dodaj do aktywności metodę onClickFindBeer()	58
Metoda onClickFindBeer() musi coś robić	59
Dysponując obiektem View, można odwoływać się do jego metod	60
Aktualizacja kodu aktywności	61
Pierwsza wersja aktywności	63
Jazda próbna — test modyfikacji	65
Tworzenie własnej klasy Javy	66
Dodaj do aktywności wywołanie metody naszej klasy, aby była wyświetlana FAKTYCZNA porada	67
Kod aktywności, wersja 2	69
Co się dzieje podczas wykonywania tego kodu?	70
Jazda próbna — test aplikacji	71
Twój przyborek do Androida	72

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 w tym celu **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



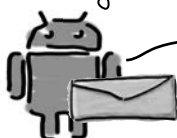
Do: InnaAktywnosc

Aplikacja może zawierać więcej niż jedną aktywność	74
Oto struktura naszej aplikacji	75
Utworzenie projektu	75
Utworzenie drugiej aktywności i układu	78
Przedstawiamy plik manifestu aplikacji na Androida	80
Użycie intencji do uruchomienia drugiej aktywności	83
Co się dzieje po uruchomieniu aplikacji?	84
Jazda próbna aplikacji	85
Przekazanie tekstu do drugiej aktywności	86
Aktualizacja właściwości widoku tekstowego	87
Metoda putExtra() zapisuje w intencji dodatkowe informacje	88
Aktualizacja kodu aktywności CreateMessageActivity	91
Zastosowanie informacji przekazanych w intencji w klasie ReceiveMessageActivity	92
Co się dzieje, gdy użytkownik kliknie przycisk Wyślij wiadomość?	93
Jazda próbna aplikacji	94
Jak działają aplikacje na Androida?	95
Co się dzieje podczas działania kodu?	99
Jak Android korzysta z filtrów intencji?	102
Musisz uruchomić aplikację na PRAWDZIwym urządzeniu	105
Jazda próbna aplikacji	107
Zmień kod, aby wyświetlać okno dialogowe	111
Jazda próbna aplikacji	112
Twój przybornik do Androida	114

Hej, użytkowniku! Każda z tych aktywności może wysyłać wiadomości. Której z nich chcesz użyć?



CreateMessageActivity



Android



Użytkownik

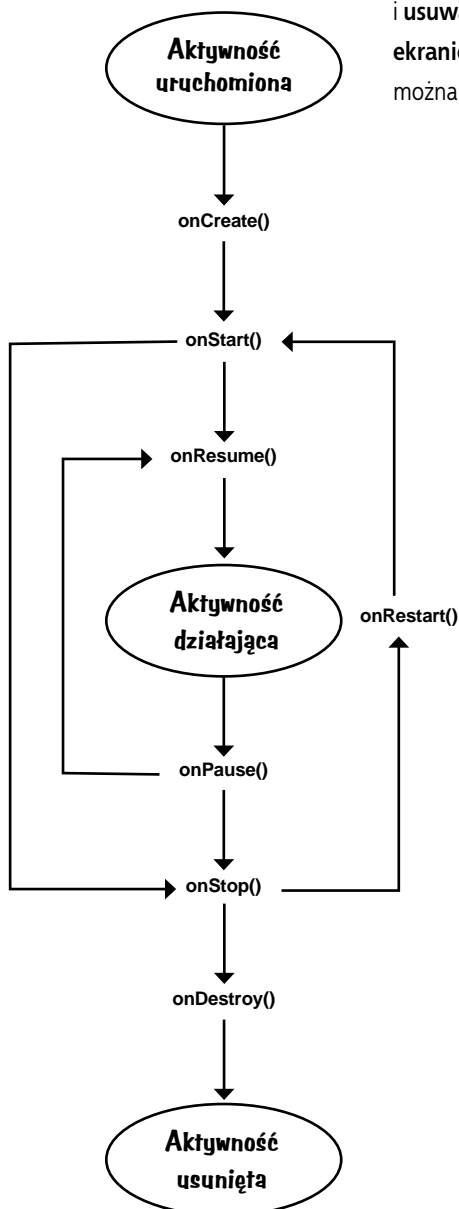
Cykl życia aktywności

Była sobie aktywność

4

Aktywności stanowią podstawę wszystkich aplikacji na Androida.

Wiesz już, jak tworzyć aktywności i jak sprawić, by jedna aktywność uruchomiła drugą, używając do tego celu intencji. Ale **co tak naprawdę dzieje się za kulisami?** W tym rozdziale nieco dokładniej poznamy **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**?



Jak właściwie działają aktywności?	116
Aplikacja stopera	118
Kod układu aplikacji stopera	119
Dodanie kodu obsługującego przyciski	122
Metoda runTimer()	123
Obiekty Handler umożliwiają planowanie wykonania kodu	124
Pełny kod metody runTimer()	125
Kompletny kod aktywności StopwatchActivity	126
Obrót ekranu zmienia konfigurację urządzenia	132
Od narodzin do śmierci: stany aktywności	133
Cykl życia aktywności: od utworzenia do usunięcia	134
W jaki sposób radzić sobie ze zmianami konfiguracji?	136
Co się stanie po uruchomieniu aplikacji?	139
Tworzenie i usuwanie to nie cały cykl życia aktywności	142
Cykl życia aktywności: widzialny czas życia	143
Zaktualizowany kod aktywności StopwatchActivity	147
Co się dzieje podczas działania aplikacji?	148
Jazda próbna aplikacji	149
A co się dzieje, jeśli aplikacja jest tylko częściowo widoczna?	150
Cykl życia aktywności: życie na pierwszym planie	151
Zatrzymanie stopera w razie wstrzymania aktywności	154
Kompletny kod aktywności	157
Wygodny przewodnik po metodach cyklu życia aktywności	161
Twój przyborek do Androida	162

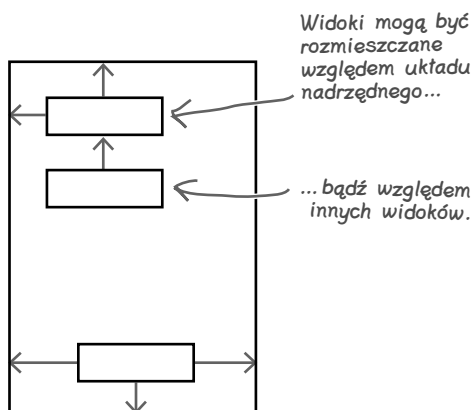
Interfejs użytkownika

5

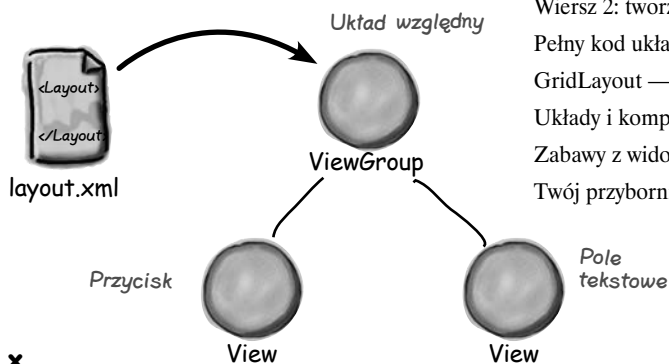
Podziwiał widoki

Nie masz innego wyjścia, musisz tworzyć szalowe układy.

Jeśli chcesz pisać aplikacje, których inni będą *używać*, musisz zadbać o to, by **wyglądały one dokładnie tak, jak sobie tego życzysz**. Zagadnienie tworzenia układów potraktowaliśmy dotychczas bardzo powierzchownie, najwyższy zatem czas, by **przyjrzeć mu się dokładnie**. W tym rozdziale pokażemy Ci różne **typy układów**, które można tworzyć, 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**.



Trzy kluczowe układy: względny, liniowy i siatki	165
Rozmieszczanie widoków względem układu nadrzędnego	168
Rozmieszczanie widoków względem innych widoków	170
Atrybuty do rozmieszczania widoków względem innych widoków	171
RelativeLayout — podsumowanie	173
Układ LinearLayout wyświetla widoki w jednym wierszu lub kolumnie	174
Zmieńmy nieco prosty układ liniowy	176
Dodawanie wagi do widoków	179
Dodawanie wagi do większej liczby widoków	180
Stosowanie atrybutu android:gravity — lista wartości	182
Inne wartości, których można używać w atrybucie android:layout_gravity	184
Kompletny układ liniowy	185
LinearLayout — podsumowanie	186
Układ GridLayout wyświetla widoki w siatce	189
Dodawanie widoków do układu siatki	190
Utwórzmy nowy układ siatki	191
Wiersz 0: dodajemy widoki do określonych wierszy i kolumn	193
Wiersz 1: tworzymy widok zajmujący komórki kilku kolumn	194
Wiersz 2: tworzymy widok zajmujący komórki kilku kolumn	195
Pełny kod układu siatki	196
GridLayout — podsumowanie	197
Układy i komponenty GUI mają wiele wspólnego	201
Zabawy z widokami	205
Twój przybornik do Androida	225



Widoki list i adaptery

6

Zorganizuj się

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

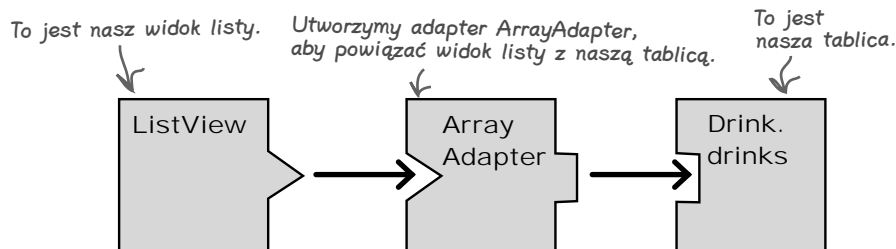
Znasz już podstawowe elementy konstrukcyjne używane do tworzenia aplikacji, więc teraz nadszedł czas, **żebyś się lepiej zorganizował**. W tym rozdziale pokażemy Ci, jak możesz **przekształcić** zbiór pomysłów **w niesamowitą aplikację**. Zobaczysz, że **listy danych** mogą stać się kluczowym elementem 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

Menu zawierające wszystko, co można u nas zjeść

Szczegółowe informacje o każdym napoju

Każda aplikacja zaczyna się od pomysłu	228
Skategoryzuj swoje pomysły — aktywności: poziom główny, kategoria i szczegóły/edycja	229
Nawigowanie po aktywnościach	230
Użyj ListViews do nawigowania po danych	231
Napišemy aplikację kafeтерии Coffeina	232
Aktywność szczegółów napoju	233
Struktura aplikacji dla kafeтерии Coffeina	234
Układ aktywności głównego poziomu składa się z obrazka i listy	238
Kompletny kod układu aktywności głównego poziomu	240
Zapewnianie reakcji ListView na kliknięcia za pomocą obiektu nasłuchującego	241
Kompletny kod aktywności TopLevelActivity	243
Jak utworzyć aktywność listy?	249
Łączenie widoków list z tablicami za pomocą adaptera ArrayAdapter	251
Dodanie adaptera ArrayAdapter do aktywności DrinkCategoryActivity	252
Co się stanie po wykonaniu kodu?	253
Jak obsługiwaliśmy kliknięcia w aktywności TopLevelActivity?	256
Kompletny kod aktywności DrinkCategoryActivity	258
Aktywność szczegółów wyświetla informacje o jednym rekordzie	259
Wypełnienie widoków danymi	261
Kod aktywności DrinkActivity	263
Jazda próbna aplikacji	266
Twój przybornik do Androida	268



7

Fragmenty

Zadbaj o modularyzację

Wiesz już, jak tworzyć aplikacje, 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 tym rozdziale pokażemy Ci, co zrobić, aby aplikacja wybierała **układ, który najlepiej pasuje do wielkości ekranu urządzenia**. Oprócz tego przedstawimy **fragmenty**, czyli **modularne komponenty kodu**, które mogą być **wielokrotnie używane przez różne aktywności**.

Struktura aplikacji Trener	273
Klasa Workout	275
Jak dodać fragment do projektu?	276
Jak wygląda kod fragmentu?	278
Przypomnienie stanów aktywności	282
Cykl życia fragmentów	283
Nasze fragmenty dziedziczą metody cyklu życia	284
Jazda próbna aplikacji	286
Jak utworzyć fragment typu ListFragment?	290
Zaktualizowany kod klasy WorkoutListFragment	292
Jazda próbna aplikacji	294
Powiązanie listy z widokiem szczegółów	295
Stosowanie transakcji fragmentu	301
Zaktualizowany kod aktywności MainActivity	302
Jazda próbna aplikacji	303
Kod fragmentu WorkoutDetailFragment	305
Struktury aplikacji na tablety i telefony	307
Różne opcje katalogów	309
Układ MainActivity dla telefonów	315
Kompletny kod aktywności DetailActivity	319
Zmodyfikowany kod aktywności MainActivity	321
Jazda próbna aplikacji	322

A zatem fragment będzie zawierał jedną listę. Kiedy chcieliśmy użyć aktywności zawierającej pojedynczą listę, zastosowaliśmy aktywność typu ListActivity. Zastanawiam się, czy przypadkiem nie istnieje jakiś typ fragmentu stanowiący odpowiednik tej klasy.



Fragmenty zagnieżdżone

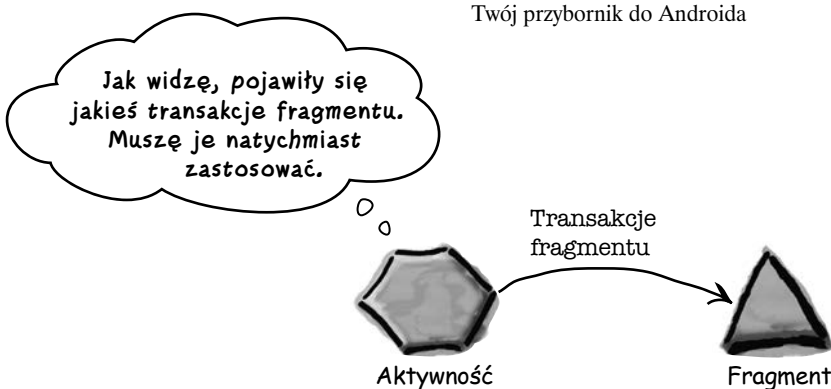
8

Zadbaj o potomstwo

Wiesz już, że stosowanie fragmentów w aktywnościach pozwala na wielokrotne wykorzystywanie kodu i zwiększa elastyczność aplikacji.

W tym rozdziale mamy zamiar pokazać Ci, jak zagnieżdżać fragmenty w innych fragmentach. Dowiesz się, jak używać menedżera fragmentów podrzędnych, by poskromić niesforne transakcje. Ponadto dowiesz się, dlaczego tak ważna jest znajomość różnic między aktywnościami i fragmentami.

Tworzenie zagnieżdżonych fragmentów	326
Kod fragmentu StopwatchFragment	332
Układ fragmentu StopwatchFragment	335
Metoda getSupportFragmentManager() tworzy transakcje na poziomie aktywności	340
Zagnieżdżone fragmenty wymagają zagnieżdżonych transakcji	341
Kompletny kod fragmentu WorkoutDetailFragment	343
Jazda próbna aplikacji	344
Dlaczego kliknięcie przycisku powoduje awarię aplikacji?	345
Przyjrzyjmy się kodowi układu StopwatchFragment	346
Zaimplementuj we fragmencie interfejs OnClickListener	349
Powiązanie obiektu nasłuchującego OnClickListener z przyciskami	351
Kod fragmentu StopwatchFragment	352
Jazda próbna aplikacji	354
Kod fragmentu WorkoutDetailFragment	358
Jazda próbna aplikacji	359
Twój przybornik do Androida	364



Paski akcji

9 Na skróty

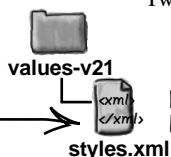
Każdy lubi chodzić na skróty.

Z tego rozdziału dowiesz się, jak korzystając z pasków akcji, wzbogacić aplikację o możliwość chodzenia na skróty. Pokażemy Ci, jak uruchamiać inne aplikacje za pomocą *elementów akcji* dodawanych do pasków akcji, 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 akcji*. Dowiesz się też, jak można nadawać tworzoną aplikacjom spójny wygląd i sposób działania, korzystając z **motywów**, i poznasz *pakiet biblioteki wsparcia systemu Android*.

Właśnie tak wygląda akcja udostępniania wyświetlona na pasku akcji. Po jej kliknięciu wyświetlana jest lista aplikacji, którym możemy udostępnić treści.



API 21?
Idealnie pasuje.



Name: AppTheme
Parent: Theme.Material.Light

Świetne aplikacje mają przejrzystą strukturę	366
Różne typy nawigacji	367
Zacznijmy od paska akcji	368
Pakiet bibliotek Support Libraries	369
Twój projekt może już używać bibliotek wsparcia	370
Zadbamy, by aplikacja używała aktualnych motywów	371
Zastosowanie motywu w pliku AndroidManifest.xml	372
Definiowanie stylów w pliku zasobów stylów	373
Określenie domyślnego motywu w pliku styles.xml	374
Co się dzieje podczas działania aplikacji?	375
Dodawanie elementów do paska akcji	376
Plik zasobów menu	377
Atrybut showAsAction menu	378
Dodawanie nowego elementu akcji	379
Utworzenie aktywności OrderActivity	382
Uruchomienie aktywności OrderActivity po kliknięciu przycisku Złóż zamówienie	383
Kompletny kod aktywności MainActivity	384
Dzielenie się treściami z poziomu paska akcji	386
Określanie treści za pomocą intencji	388
Kompletny kod aktywności MainActivity	389
Włączanie nawigacji w górę	391
Określanie aktywności nadrzędnej	392
Dodawanie przycisku W górę	393
Jazda próbna aplikacji	394
Twój przyborek do Androida	395

10

Szufłady nawigacyjne

Z miejsca na miejsce

Aplikacje są nieporównanie lepsze, gdy można się po nich łatwo poruszać.

W tym rozdziale przedstawimy Ci **szufładę nawigacyjną** — wysuwany panel, który jest wyświetlany na ekranie po przesunięciu palcem lub kliknięciu ikony umieszczonej na pasku akcji. Pokażemy Ci, jak można wyświetlać w niej **listę odnośników** umożliwiających przechodzenie do **kluczowych węzłów aplikacji**. Oprócz tego przekonasz się, że **przełączanie fragmentów** pozwala **łatwo docierać do tych węzłów** i je **szybko wyświetlać**.



Zawartość ekranu jest wyświetlana w układzie `FrameLayout`. Chcemy, by wypełniata ona cały dostępny obszar ekranu. W tym przykładzie jest ona częściowo przesłonięta przez szufładę nawigacyjną.

Zmiany w aplikacji dla restauracji Włoskie Co Nieco	398
Szufłady nawigacyjne bez tajemnic	399
Struktura aplikacji dla restauracji Włoskie Co Nieco	400
Utworzenie fragmentu <code>TopFragment</code>	401
Utworzenie fragmentu <code>PizzaFragment</code>	402
Utworzenie fragmentu <code>PastaFragment</code>	403
Utworzenie fragmentu <code>StoresFragment</code>	404
Dodanie układu <code>DrawerLayout</code>	405
Kompletna zawartość pliku <code>activity_main.xml</code>	406
Inicjalizacja listy szufłady nawigacyjnej	407
Zmiana tytułu paska akcji	412
Zamykanie szufłady nawigacyjnej	413
Zaktualizowany kod pliku <code>MainActivity.java</code>	414
Stosowanie <code>ActionBarDrawerToggle</code>	417
Modyfikowanie elementów paska akcji w trakcie działania aplikacji	418
Zaktualizowany kod aktywności <code>MainActivity</code>	419
Włączenie możliwości otwierania i zamykania szufłady nawigacyjnej	420
Synchronizacja stanu przycisku <code>ActionBarDrawerToggle</code>	421
Zaktualizowany kod aktywności <code>MainActivity</code>	422
Obsługa zmian konfiguracji	425
Reagowanie na zmiany stosu cofnięć	426
Dodawanie znaczników do fragmentów	427
Kompletny kod aktywności <code>MainActivity</code>	429
Jazda testowa aplikacji	435
Twój przybornik do Androida	436

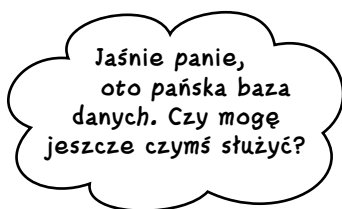
Bazy danych SQLite

11

Odpal bazę danych

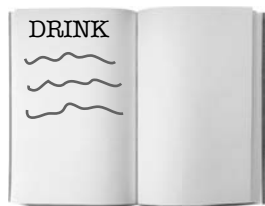
Jeśli rejestrujesz najlepsze wyniki lub przesyłane komunikaty, to Twoja aplikacja będzie musiała przechowywać dane.

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	438
Android trwale przechowuje dane, używając baz danych SQLite	439
Android udostępnia kilka klas związanych z SQLite	440
Obecna struktura aplikacji kafeterii Coffeina	441
Pomocnik SQLite zarządza Twoją bazą danych	443
Pomocnik SQLite	443
Tworzenie pomocnika SQLite	444
Wnętrze bazy danych SQLite	446
Tabele tworzymy w języku SQL	447
Wstawianie danych za pomocą metody insert()	448
Aktualizacja rekordów za pomocą metody update()	449
Określanie wielu warunków	450
Kod klasy CoffeinaDatabaseHelper	451
Co robi kod pomocnika SQLite?	452
Co zrobić, gdy trzeba będzie zmienić bazę?	455
Bazy danych SQLite mają numer wersji	456
Aktualizacja bazy danych — omówienie	457
Jak pomocnik SQLite podejmuje decyzje?	459
Aktualizacja bazy w metodzie onUpgrade()	460
Przywracanie starszej wersji bazy za pomocą metody onDowngrade()	461
Zaktualizujmy bazę danych	462
Aktualizacja istniejącej bazy danych	465
Zmiana nazwy tabeli	466
Pełny kod pomocnika SQLite	467
Kod pomocnika SQLite (ciąg dalszy)	468
Co się dzieje podczas działania kodu?	469
Twój przybornik do Androida	470

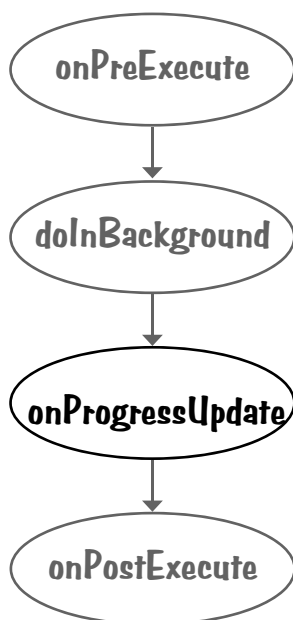
12

Kursory i zadania asynchroniczne

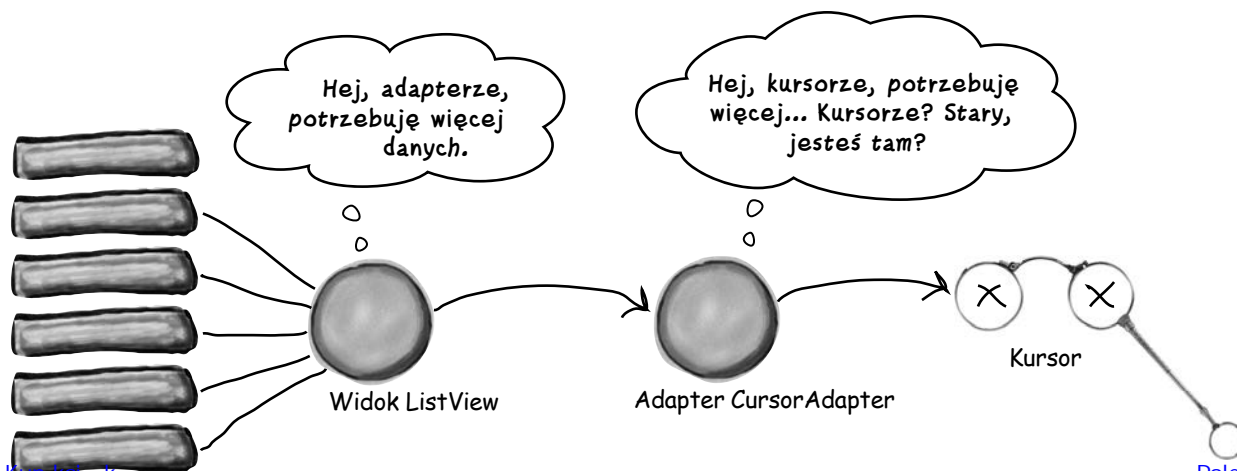
Nawiązywanie połączenia z bazą danych

Jak łączysz swoje aplikację 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. Z tego rozdziału dowiesz się, jak **tworzyć kursory**, **by pobierać dane z bazy danych**, **jak poruszać się po kursorach** oraz **jak pobierać z nich dane**. Oprócz tego dowiesz się, jak używać **adapterów kursorów**, by łączyć kursory z widokami list. A na koniec przekonasz się, że pisanie wydajnego **kodu wielowątkowego** korzystającego z klasy **AsyncTask** może zagwarantować wysoką wydajność działania aplikacji.



Aktualny kod aktywności DrinkActivity	474
Określanie tabeli i kolumn	478
Zapytania z wieloma warunkami	479
Stosowanie funkcji SQL w zapytaniach	481
Poruszanie się po kursorze	488
Pobieranie wartości z kursora	489
Kod aktywności DrinkActivity	490
Dodanie ulubionych napojów do aktywności DrinkActivity	508
Kod aktywności DrinkActivity	513
Nowy kod aktywności głównego poziomu	518
Zmodyfikowany kod aktywności TopLevelActivity	524
Metoda onPreExecute()	531
Metoda doInBackground()	532
Metoda onProgressUpdate()	533
Metoda onPostExecute()	534
Klasa AsyncTask	535
Kod aktywności DrinkActivity	537
Twój przybornik do Androida	540



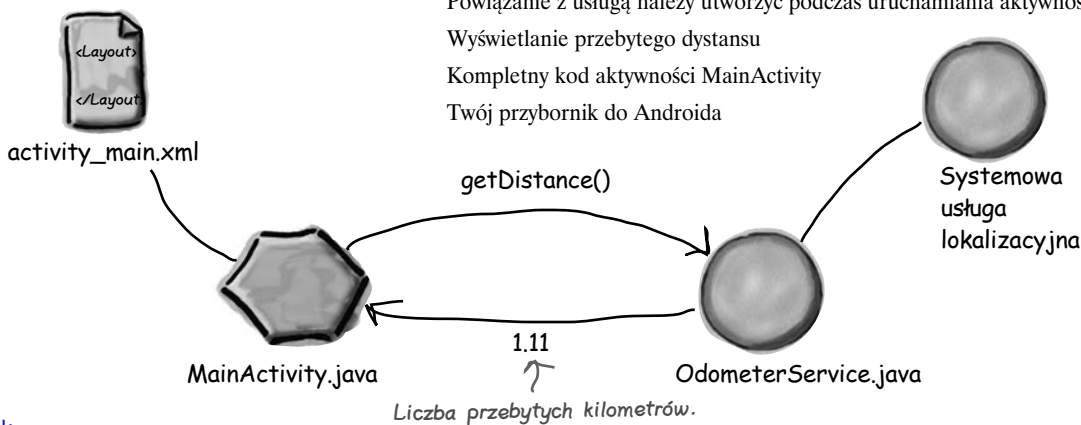
13

Usługi Do usług

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

Na przykład kiedy rozpoczniesz odtwarzanie pliku w aplikacji muzycznej, to najprawdopodobniej będziesz oczekiwać, że będzie ona kontynuowała odtwarzanie nawet wówczas, gdy przejdziesz do innej aplikacji. Z tego rozdziału dowiesz się, jak używać **usług**, by radzić sobie w sytuacjach takich jak ta. Przy okazji nauczysz się korzystać z kilku **wbudowanych usług systemu Android**. Dowiesz się, jak za pomocą **usługi powiadomień** sprawić, by użytkownik zawsze był dobrze poinformowany, i jak poznać aktualne położenie geograficzne, korzystając z **usługi lokalizacji**.

Aplikacja z usługą uruchomioną	543
Usługa IntentService z wysokości 15 tysięcy metrów	545
Jak rejestrować komunikaty?	546
Kompletny kod usługi DelayedMessageService	547
Kompletny kod usługi DelayedMessageService	554
Jak używać usługi powiadomień?	557
Uruchamianie intencji przez powiadomienie	559
Wysyłanie powiadomień za pomocą usługi systemowej	561
Kompletny kod usługi DelayedMessageService	562
Etapy tworzenia usługi Odometer	570
Zdefiniowanie obiektu Binder	573
Klasa Service ma cztery kluczowe metody	575
Dodanie obiektu LocationListener do usługi	577
Rejestracja obiektu LocationListener	578
Kompletny kod usługi OdometerService	580
Aktualizacja pliku AndroidManifest.xml	582
Aktualizacja układu aktywności MainActivity	586
Tworzenie obiektu ServiceConnection	587
Powiązanie z usługą należy utworzyć podczas uruchamiania aktywności	588
Wyświetlanie przebytego dystansu	589
Kompletny kod aktywności MainActivity	590
Twój przybornik do Androida	595



14

Material Design

W materialistycznym świecie

W API poziomu 21 Google wprowadził Material Design.

Z tego rozdziału dowiesz się, czym jest **Material Design** i jak sprawić, by nasze aplikacje do niego pasowały. Zaczniemy od przedstawienia **widoków kart** (ang. *card views*), których możemy wielokrotnie używać w całej aplikacji, zapewniając jej **spójny wygląd i sposób obsługi**. Następnie przedstawimy widok `RecyclerView` — elastycznego przyjaciela widoków list. Przy okazji dowiesz się także, jak **tworzyć własne adaptery** i jak całkowicie zmienić wygląd widoku `RecyclerView`, używając **zaledwie dwóch wierszy kodu**.



Przedstawiamy Material Design	598
Struktura aplikacji dla restauracji Włoskie Co Nieco	600
Utworzenie widoku <code>CardView</code>	603
Kompletny kod pliku <code>card_captioned_image.xml</code>	604
Utworzenie prostego adaptera	606
Zdefiniowanie obiektu <code>ViewHolder</code> na potrzeby adaptera	607
Utworzenie obiektów <code>ViewHolder</code>	608
Każdy widok <code>CardView</code> wyświetla zdjęcie i podpis	609
Dodanie danych do widoków <code>CardView</code>	610
Kompletny kod pliku <code>CaptionedImagesAdapter.java</code>	611
Utworzenie widoku <code>RecyclerView</code>	612
Dodanie widoku <code>RecyclerView</code> do układu	613
Kod klasy <code>PizzaMaterialFragment</code>	614
Do rozmieszczania zawartości <code>RecyclerView</code> używa menedżera układu	615
Określanie menedżera układu	616
Kompletny kod klasy <code>PizzaMaterialFragment</code>	617
Zastosowanie fragmentu <code>PizzaMaterialFragment</code> w aktywności <code>MainActivity</code>	618
Co się stanie po uruchomieniu kodu?	619
Utworzenie aktywności <code>PizzaDetailActivity</code>	627
Co musi robić aktywność <code>PizzaDetailActivity</code> ?	628
Aktualizacja pliku <code>AndroidManifest.xml</code>	628
Kod pliku <code>PizzaDetailActivity.java</code>	629
Obsługa kliknięć w widoku <code>RecyclerView</code>	631
Dodanie interfejsu do adaptera	634
Implementacja interfejsu <code>Listener</code> we fragmencie <code>PizzaMaterialFragment</code>	636
Umieszczenie treści na samym początku	639
Kompletny kod pliku układu <code>fragment_top.xml</code>	644
Kompletny kod klasy <code>TopFragment</code>	645
Twój przyborek do Androida	647

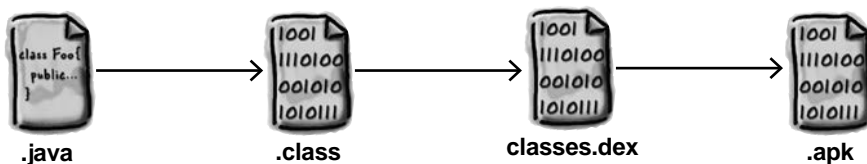
ART

A

Środowisko uruchomieniowe Androida

Aplikacje na Androida muszą działać na urządzeniach wyposażonych w słabe procesory i bardzo małą pamięć.

Aplikacje pisane w Javie mogą potrzebować sporo pamięci, a ponieważ działają wewnątrz własnej wirtualnej maszyny Javy (JVM), ich uruchomienie na komputerze o niewielkiej mocy obliczeniowej może trwać dość długo. Android rozwiązuje ten problem, nie używając JVM do uruchamiania aplikacji. Zamiast JVM używa całkowicie odmiennej wirtualnej maszyny, nazywanej **środowiskiem uruchomieniowym Androida** (ang. *Android Runtime*, w skrócie **ART**). W tym dodatku zobaczysz, jak to się dzieje, że ART umożliwia dobre działanie aplikacji napisanych w Javie nawet na małych komputerach o niewielkiej mocy obliczeniowej.



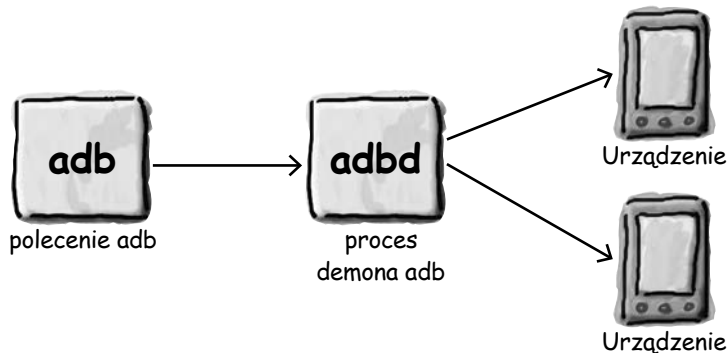
ADB

B

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

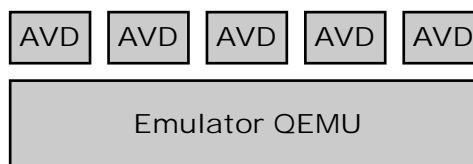
Emulator Androida



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 się przekonać, 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 (AVD) są wykonywane przez emulator QEMU.



Pozostałości

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**.



Android

1. Rozpowszechnianie aplikacji	664
2. Dostawcy treści	665
3. Klasa WebView	666
4. Animacje	667
5. Mapy	668
5. Mapy (ciąg dalszy)	669
6. Klasa CursorLoader	669
7. Odbiorcy komunikatów	670
8. Widżety aplikacji	671
9. Grafika 9-patch	672
10. Testowanie	673



Skorowidz

674

10. Szuflady nawigacyjne

Z miejsca na miejsce



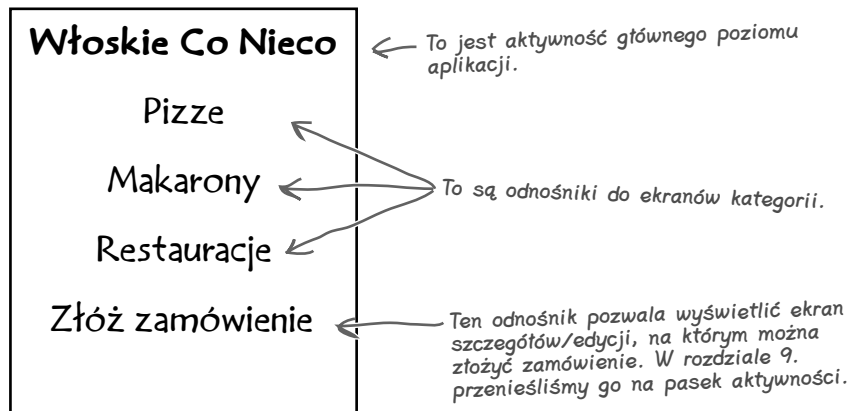
Wiem, że ze swoją cudowną szufladą nawigacyjną nigdy się nie zgubię!

Aplikacje są nieporównanie lepsze, gdy można się po nich łatwo poruszać.

W tym rozdziale przedstawimy Ci **szufladę nawigacyjną** — wysuwany panel, który jest wyświetlany na ekranie po przesunięciu palcem lub kliknięciu ikony umieszczonej na pasku akcji. Pokażemy Ci, jak można wyświetlać w niej **listę odnośników** umożliwiających przechodzenie do **kluczowych węzłów aplikacji**. Oprócz tego przekonasz się, że **przełączanie fragmentów** pozwala **łatwo docierać do tych węzłów** i je **szybko wyświetlać**.

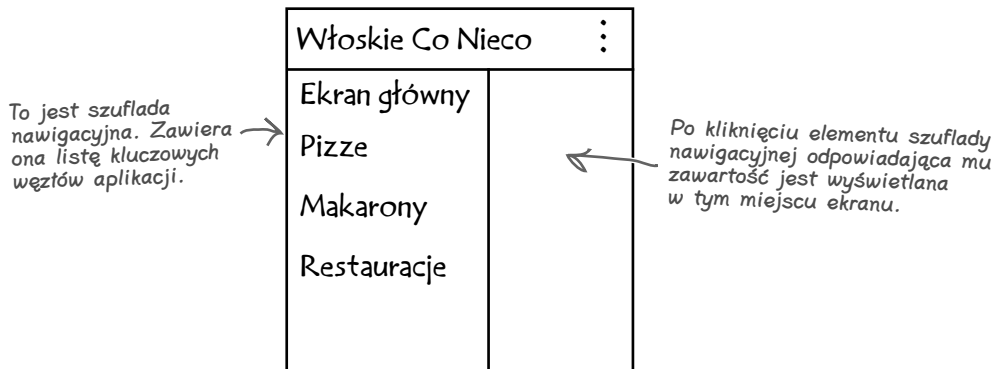
Zmiany w aplikacji dla restauracji Włoskie Co Nieco

W rozdziale 9, przedstawiliśmy szkic głównego ekranu najwyższego poziomu aplikacji dla Włoskiego Co Nieco. Zawierał on listę opcji umożliwiających użytkownikom przechodzenie w różne miejsca aplikacji. Pierwsze trzy opcje prowadziły do ekranów kategorii prezentujących odpowiednio pizze, dania z makaronów i restauracje, natomiast ostatnia opcja pozwalała przejść do ekranu szczegółów/edycji, na którym użytkownik mógł złożyć zamówienie.



Wiesz już, jak dodawać nowe elementy do paska akcji. To rozwiązanie najlepiej się nadaje do umieszczania opcji aktywnych, takich jak tworzenie zamówień. Co jednak można zrobić w przypadku odnośników do ekranów kategorii? Ponieważ są to opcje pasywne, służące za elementy nawigacyjne w obrębie całej aplikacji, musimy je potraktować nieco inaczej.

Pozostałe trzy opcje — *Pizze*, *Makarony* oraz *Restauracje* — umieścimy w **szufladzie nawigacyjnej** (ang. *navigation drawer*). Szuflada nawigacyjna to wysuwany panel zawierający odnośniki do głównych części aplikacji. Te główne części aplikacji są określane jako jej **kluczowe węzły** i zazwyczaj odgrywają najważniejszą rolę w poruszaniu się po niej; przeważnie są nimi ekrany głównego poziomu i ekrany kategorii.



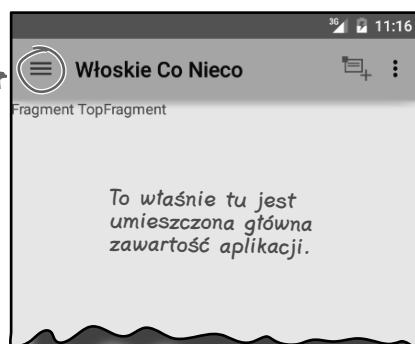
Szufłady nawigacyjne bez tajemnic

Do implementacji szuflad nawigacyjnych używany jest specjalny typ układów — **DrawerLayout**. Układ typu `DrawerLayout` zarządza dwoma widokami:

- ★ Widok zawartości głównej, którym zazwyczaj jest układ `FrameLayout`, dzięki czemu w prosty sposób można wyświetlać i zmieniać prezentowane fragmenty.
- ★ Widok szuflady nawigacyjnej, którym zazwyczaj jest `ListView`.

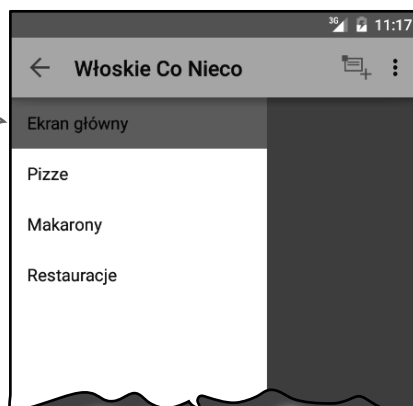
Domyślnie układ `DrawerLayout` wyświetla widok prezentujący główną zawartość aplikacji. Wygląda on bardzo podobnie do normalnej aktywności:

To jest ikona szuflady nawigacyjnej. Wystarczy ją kliknąć lub przeciągnąć palcem, by wyświetlić zawartość szuflady.



Kiedy klikniemy ikonę szuflady nawigacyjnej lub przeciągniemy palcem od krawędzi ekranu do jego środka, widok zawierający szufladę nawigacyjną zostanie wysunięty na ekran i częściowo przesłoni prezentowane na nim treści.

To jest szuflada nawigacyjna. Jak widać, zawiera ona listę opcji.



Szuflada jest nasuwana na prezentowaną zawartość główną.

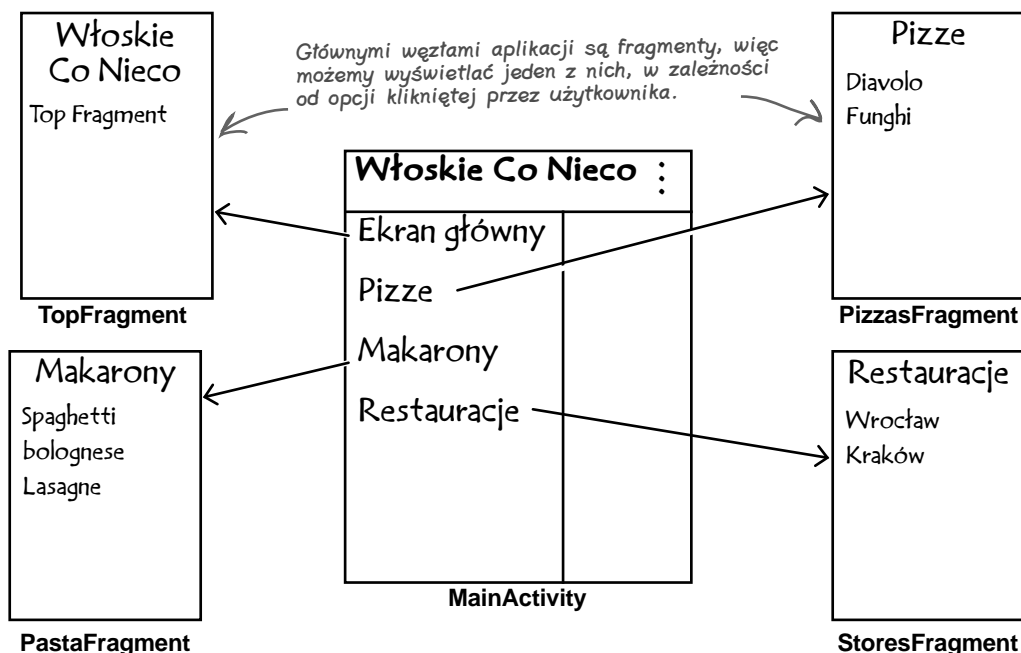
To właśnie zawartość szuflady umożliwia nam poruszanie się po aplikacji.

A jaki to ma wpływ na strukturę naszej aplikacji dla *Włoskiego Co Nieco*?

Struktura aplikacji dla restauracji Włoskie Co Nieco

Planujemy zmienić aktywność `MainActivity` w taki sposób, by korzystała z szuflady nawigacyjnej. Będzie ona zawierała układ `FrameLayout` do wyświetlania fragmentów i widok listy do prezentowania listy opcji.

Widok listy będzie zawierał opcje *Ekran główny*, *Pizze*, *Makarony* oraz *Restauracje*, dzięki którym użytkownik będzie mógł w prosty sposób poruszać się pomiędzy głównymi węzłami aplikacji. Dla każdej z tych opcji utworzymy następnie odrębny fragment. Oznacza to, że będziemy mogli podmieniać fragmenty w trakcie działania aplikacji, a użytkownik będzie w stanie uzyskać dostęp do szuflady nawigacyjnej z każdego ekranu aplikacji.



Oto czynności, które wykonamy w ramach wprowadzania tych modyfikacji:

- 1 **Utworzymy fragmenty dla poszczególnych głównych węzłów aplikacji.**
- 2 **Utworzymy i zainicjujemy szufladę nawigacyjną.**
Szuflada ta będzie zawierała widok `ListView` prezentujący listę opcji.
- 3 **Zapewnimy, by widok `ListView` reagował na kliknięcia elementów.**
Dzięki temu użytkownik będzie mógł przechodzić do głównych węzłów aplikacji.
- 4 **Dodamy `ActionBarDrawerToggle`.**
Dzięki temu elementowi użytkownik będzie mógł kontrolować szufladę przy użyciu paska akcji, a aktywność będzie mogła reagować na zdarzenia otwierania i zamykania szuflady.

Zacniemy od utworzenia fragmentów.



Spokojnie

Dodanie szuflady nawigacyjnej wymaga napisania sporej ilości kodu.

Na omówienie sposobu dodawania szuflady nawigacyjnej poświęcimy cały ten rozdział, a na samym jego końcu pokażemy kompletny kod aktywności `MainActivity`.

Utworzenie fragmentu TopFragment

Fragmentu TopFragment użyjemy do prezentowania zawartości najwyższego poziomu. Na razie ograniczymy się jednak do wyświetlenia tekstu „Fragment TopFragment”, dzięki czemu będziemy wiedzieli, który fragment jest aktualnie widoczny. A zatem utwórz pusty fragment, nadaj mu nazwę *TopFragment*, a używany przez niego plik układu nazwij *fragment_top*.

Oto zawartość pliku *TopFragment.java*:

```
package com.hfad.wloskieconieco;

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

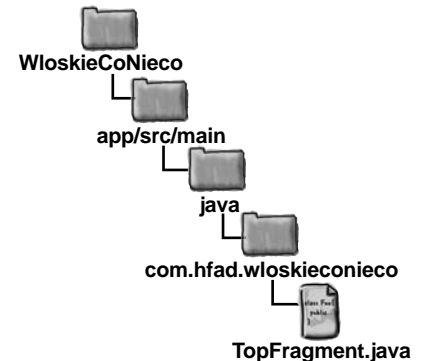
public class TopFragment extends Fragment {

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

Nasz TopFragment jest
zwyczajnym fragmentem.

- Dodanie fragmentów
- Utworzenie szufłady
- Obsługa kliknięcia ListView
- Dodanie ActionBarDrawerToggle

Wszystkie nasze fragmenty utworzymy na bazie pustych fragmentów, gdyż i tak w całości będziemy zastępować kod wygenerowany przez Android Studio.



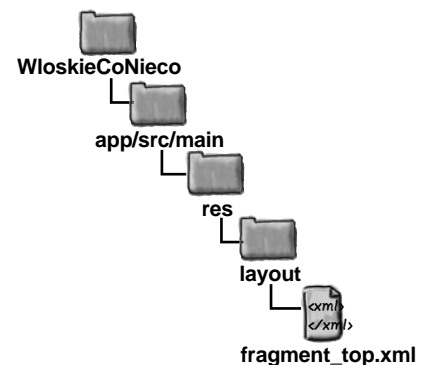
Oprócz tego dodaj do pliku *strings.xml* następujący zasób łańcuchowy (użyjemy go w układzie fragmentu):

```
<string name="title_top">Fragment TopFragment</string>
```

Dodaj ten łańcuch do pliku *strings.xml*. Dodamy go do układu, aby wiedzieć, kiedy będzie wyświetlany fragment TopFragment.

Oto kod pliku *fragment_top.xml*:

```
<RelativeLayout 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"
    tools:context=".MainActivity">
    <TextView
        android:text="@string/title_top"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```



Utworzenie fragmentu PizzaFragment

Utworzenie fragmentu PizzaFragment

Do wyświetlania listy pizz użyjemy fragmentu PizzaFragment typu ListFragment. A zatem utwórz pusty fragment o nazwie PizzaFragment, jednocześnie usuń zaznaczenie pola wyboru pozwalającego na utworzenie pliku układu. Rezygnujemy z tworzenia układu dlatego, że fragmenty typu ListFragment używają swoich własnych układów.

Następnie dodaj do pliku *strings.xml* nową tablicę łańcuchów o nazwie pizzas (zapiszemy w niej nazwy dostępnych pizz):

```
<string-array name="pizzas">
  <item>Diavolo</item>
  <item>Funghi</item>
</string-array>
```

← Dodajemy tablicę z nazwami pizz do pliku strings.xml.

Następnie zmodyfikuj kod w pliku *PizzaFragment.java* w taki sposób, by tworzony fragment dziedziczył po klasie ListFragment. Lista wyświetlana w tym fragmencie powinna zostać wypełniona nazwami pizz. Oto kod tego pliku:

```
package com.hfad.wloskieconieco;

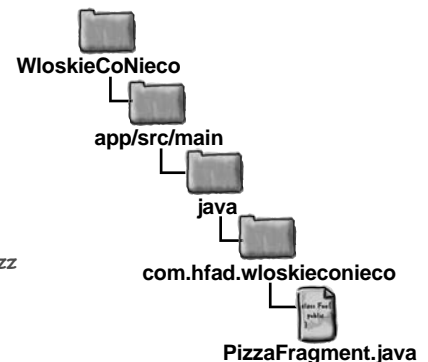
import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;

public class PizzaFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            inflater.getContext(),
            android.R.layout.simple_list_item_1,
            getResources().getStringArray(R.array.pizzas));
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

Do wyświetlenia listy pizz użyjemy fragmentu typu ListFragment.

- Dodanie fragmentów
- Utworzenie szuflady
- Obsługa kliknięcia ListView
- Dodanie ActionBarDrawerToggle



Utworzenie fragmentu PastaFragment

Fragmentu `PastaFragment` typu `ListFragment` użyjemy do wyświetlania listy dań z makronu. Aby przygotować ten fragment, utwórz pusty fragment i nadaj mu nazwę `PastaFragment`. Nie zapomnij także o usunięciu znaczników z pola wyboru pozwalającego na utworzenie pliku układu, gdyż fragmenty typu `ListFragment` dysponują własnym układem.

Następnie do pliku `strings.xml` dodaj tablicę łańcuchów o nazwie `pasta` (będzie ona zawierała nazwy dań z makaronu):

```
<string-array name="pasta">
    <item>Spaghetti bolognese</item>
    <item>Lasagne</item>
</string-array>
```

← Dodajemy do pliku `strings.xml` tablicę z nazwami dań z makaronu.

Następnie zmień kod w pliku `PastaFragment.java` w taki sposób, by klasa `PastaFragment` dziedziczyła po `ListFragment`. Widok `Listview` tego fragmentu powinien zostać wypełniony nazwami dań z makaronu. Oto kod tej klasy:

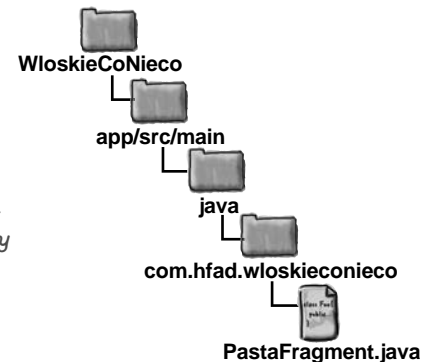
```
package com.hfad.wloskieconieco;

import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;

public class PastaFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            inflater.getContext(),
            android.R.layout.simple_list_item_1,
            getResources().getStringArray(R.array.pasta));
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

Fragmentu typu `ListFragment` użyjemy do wyświetlenia listy dań z makaronu.



Utworzenie fragmentu StoresFragment

Fragmentu StoresFragment typu ListFragment użyjemy do wyświetlania listy restauracji. Aby przygotować ten fragment, utwórz pusty fragment i nadaj mu nazwę StoresFragment. Nie zapomnij także o usunięciu znaczników z pola wyboru pozwalającego na utworzenie pliku układu, gdyż fragmenty typu ListFragment dysponują własnym układem.

Następnie do pliku *strings.xml* dodaj tablicę łańcuchów o nazwie stores (będzie ona zawierała nazwy miast, w których są restauracje Włoskie Co Nieco):

```
<string-array name="stores">
  <item>Wrocław</item>
  <item>Kraków</item>
</string-array>
```

← Dodajemy do pliku *strings.xml* tablicę z nazwami miast, w których są restauracje.

Następnie zmień kod w pliku *StoresFragment.java* w taki sposób, by klasa StoresFragment dziedziczyła po ListFragment. Widok ListView tego fragmentu powinien zostać wypełniony nazwami miast. Oto kod tej klasy:

```
package com.hfad.wloskieconieco;

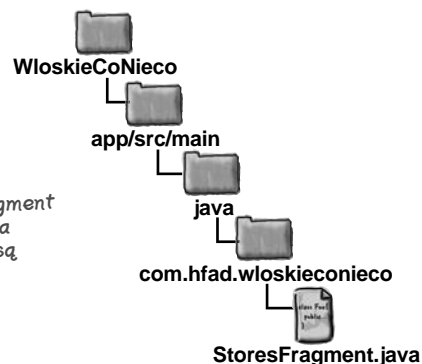
import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;

public class StoresFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            inflater.getContext(),
            android.R.layout.simple_list_item_1,
            getResources().getStringArray(R.array.stores));
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

Fragmentu typu ListFragment użyjemy do wyświetlenia listy miast, w których są restauracje.

- Dodanie fragmentów
- Utworzenie szuflady
- Obsługa kliknięcia ListView
- Dodanie ActionBarDrawerToggle



Dodanie układu DrawerLayout

Teraz zajmiemy się zmianą układu używanego przez aktywność główną, MainActivity, tak by używała ona układu typu DrawerLayout. Zgodnie z tym, o czym wspominaliśmy już wcześniej, będzie on zawierał układ FrameLayout, którego użyjemy do wyświetlania fragmentów, i widok ListView prezentujący zawartość szuflady nawigacyjnej.

Układ DrawerLayout można utworzyć, używając następującego kodu:

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        ... />

    <ListView
        android:layout_width="240dp"
        android:layout_height="match_parent"
        ... />

</android.support.v4.widget.DrawerLayout>
```

Nasz układ korzysta z układu DrawerLayout pochodzącego z biblioteki wsparcia v4. Biblioteka appcompat v7 zawiera bibliotekę wsparcia v4.

Układu FrameLayout będziemy używać do wyświetlania fragmentów.

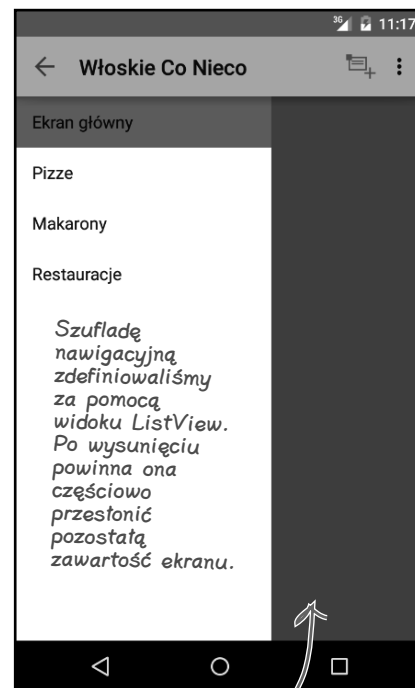
Element ListView opisuje szufladę nawigacyjną.

Elementem głównym naszego nowego układu jest DrawerLayout. To zrozumiałe, gdyż musi on kontrolować wszystko, co dzieje się na ekranie. Klasa DrawerLayout pochodzi z biblioteki Support Library v4, dlatego zastosujemy jej pełną nazwę: android.support.v4.widget.DrawerLayout.

Pierwszy element umieszczony w układzie DrawerLayout będzie służył do wyświetlania zawartości. W naszym przypadku będzie to układ FrameLayout, w którym będziemy wyświetlali poszczególne fragmenty. Chcemy, aby był on możliwie jak największy, dlatego obu jego atrybutom — layout_width i layout_height — przypisaliśmy wartość "match_parent".

Drugim elementem układu DrawerLayout jest sama szuflada nawigacyjna. Jeśli utworzymy ją za pomocą widoku ListView, to będzie ona zawierała listę opcji. Zazwyczaj będziemy chcieli, aby ta szuflada wysuwała się w poziomie i częściowo przykrywała dotychczasową zawartość ekranu. Dlatego atrybutowi layout_height tego widoku przypisaliśmy wartość "match_parent", natomiast w atrybucie layout_width podaliśmy stałą szerokość.

Kompletny kod pliku activity_main.xml przedstawiamy na następnej stronie.



Zawartość ekranu jest wyświetlana w układzie FrameLayout. Chcemy, by wypełniała ona cały dostępny obszar ekranu. W tym przykładzie jest ona częściowo przesłonięta przez szufladę nawigacyjną.

Kompletna zawartość pliku activity_main.xml

Oto kompletna zawartość pliku układu *activity_main.xml*:

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

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

    <ListView android:id="@+id/drawer"
        android:layout_width="240dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:choiceMode="singleChoice"
        android:divider="@android:color/transparent"
        android:dividerHeight="0dp"
        android:background="#ffffff" />
</android.support.v4.widget.DrawerLayout>
```

Szufladę
nawigacyjną
opisuje
widok
ListView.

Fragmenty będą
wyświetlane
w układzie
FrameLayout.

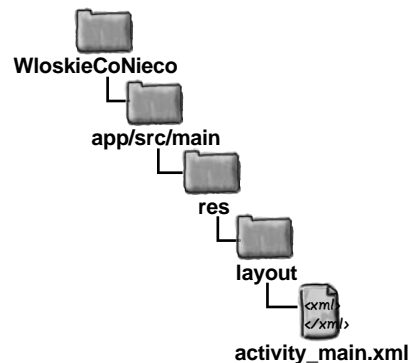
To jest szerokość szuflady.

Ten atrybut określa, gdzie należy umieścić szufladę.

Ten atrybut określa, że w danej chwili
można wybrać tylko jeden element listy.

W tych wierszach wyznaczamy linie oddzielające
poszczególne elementy listy i określamy kolor
tła szuflady.

- Dodanie fragmentów
- Utworzenie szuflady
- Obsługa kliknięcia ListView
- Dodanie ActionBarDrawerToggle



Zwróć baczność uwagę na ustawienia podane w elemencie `<ListView>`, gdyż najprawdopodobniej wszystkie szuflady nawigacyjne będą wyglądały podobnie.

Do określenia wymiarów szuflady używamy atrybutów `layout_width` i `layout_height`. Pierwszemu z nich przypisaliśmy szerokość "240dp", dzięki czemu po otwarciu szuflada będzie miała szerokość 240 dp.

Przypisanie atrybutowi `layout_gravity` wartości "start" sprawi, że w razie wyboru języka, w którym znaki są zapisywane od lewej do prawej, szuflada zostanie umieszczona po lewej stronie ekranu, a w przypadku języków, w których znaki są zapisywane od prawej do lewej, zostanie ona umieszczona po prawej stronie.

Atrybuty `divider`, `dividerHeight` oraz `background` zastosowaliśmy, aby ukryć linie oddzielające poszczególne elementy listy i określić kolor jej tła.

I wreszcie przypisanie atrybutowi `choiceMode` wartości "singleChoice" sprawi, że w danym momencie użytkownik będzie mógł wybrać tylko jeden element listy.



Obejrzyj to!

Jeśli Twój projekt nie zawiera zależności od biblioteki wsparcia appcompat v7, to przedstawiony w tym rozdziale kod szuflady nawigacyjnej nie będzie działał.

W Android Studio zależnościami możesz zarządzać, wybierając z menu opcje *File/Project Structure/App/Dependencies*.

Inicjalizacja listy szufłady nawigacyjnej

Skoro już dodaliśmy do pliku `activity_main.xml` układ `DrawerLayout`, musimy teraz określić jego zachowanie w pliku `MainActivity.java`. Pierwszą rzeczą, którą należy zrobić, jest dodanie opcji do widoku listy. W tym celu dodamy do pliku `strings.xml` kolejną tablicę łańcuchów. Następnie użyjemy adaptera tablicowego do określenia opcji listy.

Poniżej przedstawiliśmy tablicę łańcuchów znaków, którą musimy dodać do pliku zasobów `strings.xml` (każdy element tej tablicy określa fragment, który zostanie wyświetlony po kliknięciu danego elementu listy):

```
<string-array name="titles">
  <item>Ekran główny</item>
  <item>Pizze</item>
  <item>Makarony</item>
  <item>Resatuaracje</item>
</string-array>
```

To są opcje, które zostaną wyświetlone w szufładzie nawigacyjnej. Dodaj tę tablicę do pliku `strings.xml`.

Zawartość listy określimy w kodzie pliku `MainActivity.java`, a konkretnie w metodzie `onCreate()`. Tablicę łańcuchów znaków i widok listy zapiszemy w zmiennych prywatnych, gdyż będą nam one jeszcze potrzebne w innych miejscach kodu. Oto kod odpowiedzialny za wypełnienie listy:

```
...
import android.widget.AdapterView;
import android.widget.ListView;

public class MainActivity extends Activity {
  ...
  private String[] titles;
  private ListView drawerList;

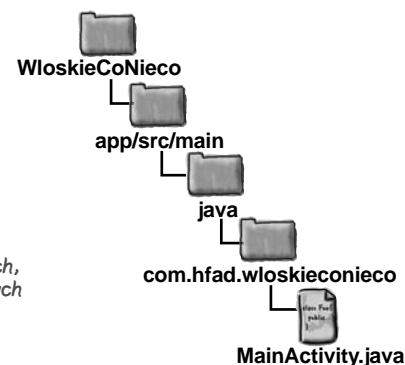
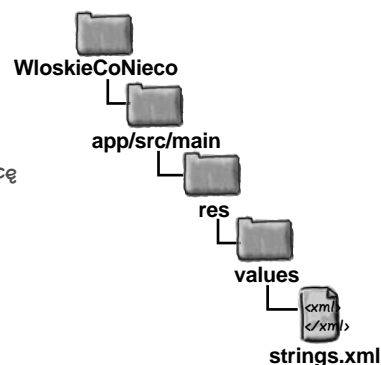
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    ...
    titles = getResources().getStringArray(R.array.titles);
    drawerList = (ListView)findViewById(R.id.drawer);
    drawerList.setAdapter(new ArrayAdapter<String>(this,
      android.R.layout.simple_list_item_activated_1, titles));
  }
  ...
}
```

Używamy tych klas, więc musimy je zaimportować.

Tych danych będziemy także potrzebowali w innych metodach, więc zapiszemy je w prywatnych zmiennych klasowych.

Do określenia zawartości widoku listy użyjemy adaptera `ArrayAdapter`.

Zastosowanie `simple_list_item_activated_1` oznacza, że element kliknięty przez użytkownika ma być wyróżniony.

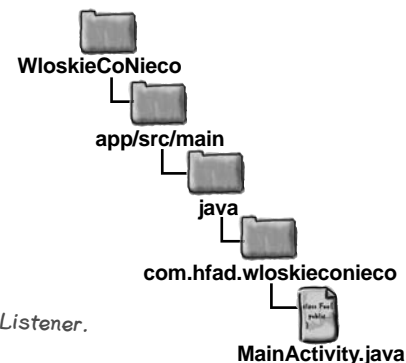


Skoro już określiliśmy opcje wyświetlane na liście szufłady nawigacyjnej, zadbajmy o to, by lista ta reagowała na kliknięcia.

Zastosowanie `OnItemClickListener`, aby zapewnić reagowanie na kliknięcia elementów listy

- Dodanie fragmentów
- Utworzenie szuflady
- Obsługa kliknięcia `ListView`
- Dodanie `ActionBarDrawerToggle`

Aby lista reagowała na kliknięcia, zastosujemy to samo rozwiązanie, którego użyliśmy już w rozdziale 6. — zaimplementujemy interfejs `OnItemClickListener`. Innymi słowy: utworzymy obiekt nasłuchujący, zaimplementujemy jego metodę `onItemClick()`, a następnie przypiszemy ten obiekt nasłuchujący do widoku listy. Oto kod, który realizuje te operacje:



```

...
import android.view.View;
import android.widget.AdapterView;

public class MainActivity extends Activity {
    ...

    private class DrawerItemClickListener implements ListView.OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id){
            // Kod, który należy wykonać po kliknięciu elementu listy
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        drawerList.setOnItemClickListener(new DrawerItemClickListener());
    }
};
  
```

Używamy tych klas, więc musimy je zaimportować.

Ta klasa opisuje nasz obiekt nasłuchujący typu `OnItemClickListener`.

Kiedy użytkownik kliknie któryś z elementów w szufladzie nawigacyjnej, zostanie wywołana metoda `onItemClick()`.

Do widoku `ListView` w szufladzie nawigacyjnej dodajemy nową instancję obiektu `OnItemClickListener`.

Metoda `onItemClick()` musi zawierać kod, który ma zostać wykonany, kiedy użytkownik kliknie któryś z elementów widoku listy. W naszym przypadku w odpowiedzi na kliknięcie wywołamy metodę `selectItem()`, przekazując do niej pozycję klikniętego elementu na liście. Już zaraz zajmiemy się zaimplementowaniem tej metody.

Metoda `selectItem()` będzie musiała wykonywać trzy operacje:

- ★ Podmienić fragment wyświetlany w układzie `FrameLayout`.
- ★ Zmienić tytuł wyświetlany na pasku akcji tak, by odpowiadał nowemu fragmentowi.
- ★ Zamknąć szufladę nawigacyjną.

Obecnie wiesz już wszystko, co jest niezbędne do zaimplementowania pierwszej z tych operacji, dlatego zajmiesz się tym samodzielnie w ramach ćwiczenia przedstawionego na następnej stronie.



Magnesiki z kodem

Kiedy użytkownik kliknie element wyświetlony na liście w szufladzie nawigacyjnej, wówczas w układzie `FrameLayout` o identyfikatorze `content_frame` musi zostać wyświetlony odpowiedni fragment. Przekonajmy się, czy potrafisz uzupełnić poniższy kod.

```
private void selectItem(int position) {
    Fragment fragment;

    switch(.....) {
        case 1:

            fragment = .....;
            break;
        case 2:

            fragment = .....;
            break;
        case 3:

            fragment = .....;
            break;
        default:

            fragment = .....;
    }

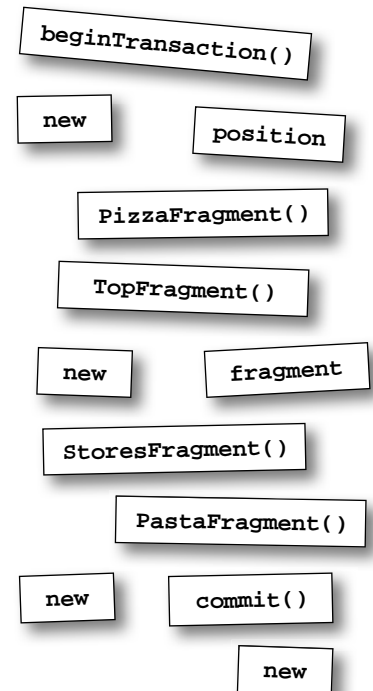
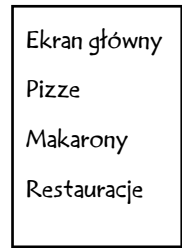
    FragmentTransaction ft = getFragmentManager(). .....;

    ft.replace(R.id.content_frame, .....);

    ft.addToBackStack(null);
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);

    ft. ....;
}
```

To jest widok `ListView` prezentujący listę elementów.





Magnesiki z kodem. Rozwiązanie

Kiedy użytkownik kliknie element wyświetlony na liście w szufladzie nawigacyjnej, wówczas w układzie FrameLayout o identyfikatorze content_frame musi zostać wyświetlony odpowiedni fragment. Przekonajmy się, czy potrafisz uzupełnić poniższy kod.

To jest widok ListView prezentujący listę elementów.

- Ekran główny
- Pizze
- Makarony
- Restauracje

```
private void selectItem(int position) {
```

```
    Fragment fragment;
```

```
    switch(... position ..) {
```

```
        case 1:
```

```
            fragment = new .. PizzaFragment() ..;
```

```
            break;
```

```
        case 2:
```

```
            fragment = new .. PastaFragment() ..;
```

```
            break;
```

```
        case 3:
```

```
            fragment = new .. StoresFragment() ..;
```

```
            break;
```

```
        default:
```

```
            fragment = new .. TopFragment() ..;
```

```
    }
```

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

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

```
    ft.addToBackStack(null);
```

```
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
```

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

```
}
```

Sprawdzamy pozycję klikniętego elementu na liście ListView szuflady nawigacyjnej.

Tworzymy fragment, którego typ jest zależny od pozycji klikniętego elementu. Na przykład jeśli użytkownik kliknął element Pizze, to tworzymy fragment klasy Pizzafragment.

Domyślnie tworzymy fragment klasy Topfragment.

Rozpoczynamy transakcję fragmentu, która podmieni aktualnie prezentowany fragment.

Zatwierdzamy transakcję.

Metoda selectItem() w obecnej postaci

Poniżej przedstawiliśmy zmodyfikowany kod pliku *MainActivity.java* (kiedy zostanie kliknięty jeden z elementów prezentowanych w szufładzie nawigacyjnej, wywoływana jest metoda `selectItem()`, która wyświetla odpowiedni fragment):

```

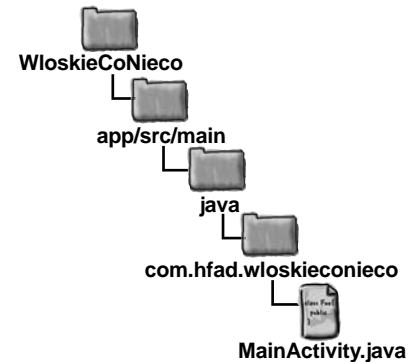
...
import android.app.Fragment;
import android.app.FragmentTransaction;

public class MainActivity extends Activity {
    ...
    private class DrawerItemClickListener implements ListView.OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id){
            selectItem(position); ← Po kliknięciu elementu wywołujemy
                                   metodę selectItem().
        }
    };

    private void selectItem(int position) { ← Sprawdzamy pozycję klikniętego elementu na liście.
        Fragment fragment;
        switch(position) {
            case 1:
                fragment = new PizzaFragment(); ← Na podstawie pozycji klikniętego elementu
                                                  na liście określamy typ tworzonego
                                                  fragmentu. Na przykład opcja „Pizze”
                                                  znajduje się na pozycji 1, więc
                                                  w przypadku jej kliknięcia utworzymy
                                                  fragment PizzaFragment.
                break;
            case 2:
                fragment = new PastaFragment();
                break;
            case 3:
                fragment = new StoresFragment();
                break;
            default:
                fragment = new TopFragment(); ← Domyślnie tworzymy fragment TopFragment.
        }
        FragmentTransaction ft = getFragmentManager().beginTransaction();
        ft.replace(R.id.content_frame, fragment);
        ft.addToBackStack(null);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
    }
}

```

↑
Używamy transakcji, by podmienić aktualnie wyświetlany fragment.



Skoro metoda `selectItem()` wyświetla już odpowiedni fragment, zajmijmy się zmianą tytułu paska akcji.

Zmiana tytułu paska akcji

Oprócz podmieniania wyświetlanego fragmentu musimy także zmieniać tytuł wyświetlany na pasku akcji, tak by odpowiadał on aktualnie prezentowanemu fragmentowi. Chcemy, by na pasku akcji domyślnie była wyświetlana nazwa aplikacji, jeśli jednak użytkownik kliknie na przykład opcję *Pizze*, to będziemy chcieli zmienić tytuł paska akcji na „Pizze”. Dzięki temu użytkownik będzie wiedział, w którym miejscu aplikacji aktualnie się znajduje.

Implementując to rozwiązanie, wykorzystamy pozycję klikniętego elementu listy i na jej podstawie określimy tekst, który zostanie wyświetlony na pasku akcji. Następnie zmienimy tytuł paska, używając metody `setTitle()` klasy `ActionBar`. Cały kod realizujący te operacje umieścimy w osobnej metodzie, gdyż będziemy jej potrzebowali także w innych miejscach aplikacji. Oto implementacja zmiany tytułu paska akcji:

```
private void selectItem(int position) {
    ...
    // Ustawiamy tytuł paska akcji
    setActionBarTitle(position);
}

private void setActionBarTitle(int position) {
    String title;
    if (position == 0){
        title = getResources().getString(R.string.app_name);
    } else {
        title = titles[position];
    }
    getActionBar().setTitle(title);
}
```

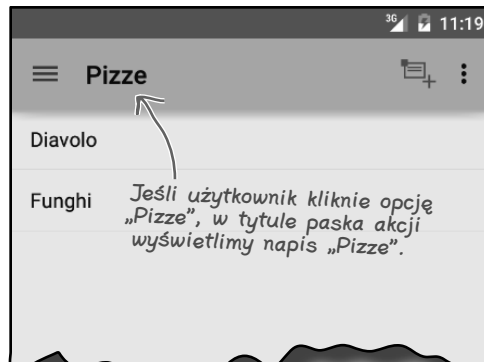
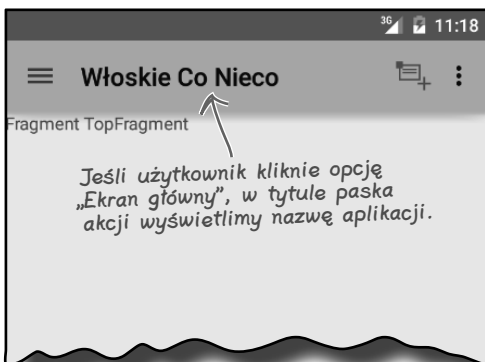
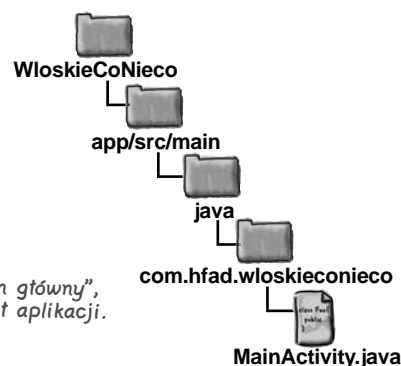
Wywołujemy metodę `setActionBarTitle()`, przekazując do niej pozycję klikniętego elementu listy.

Jeśli użytkownik kliknie opcję „Ekran główny”, to na pasku akcji wyświetlamy tytuł aplikacji.

W przeciwnym razie pobieramy tańcuch z tablicy `titles`, określony na podstawie pozycji klikniętego elementu listy.

To wywołanie wyświetla tańcuch znaków w tytule paska akcji.

- Dodanie fragmentów
- Utworzenie szuflady
- Obsługa kliknięcia `ListView`
- Dodanie `ActionBarDrawerToggle`



Zamykanie szufłady nawigacyjnej

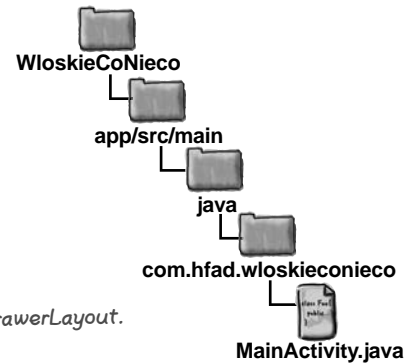
Ostatnią czynnością, którą wykonamy w kodzie metody `selectItem()`, będzie zamknięcie szufłady nawigacyjnej. Dzięki temu użytkownik nie będzie musiał robić tego samodzielnie.

Aby zamknąć szufładę, musimy pobrać referencję do układu `DrawerLayout` i wywołać jego metodę `closeDrawer()`. Ta metoda wymaga przekazania tylko jednego argumentu — obiektu `View` reprezentującego widok stanowiący szufładę nawigacyjną. W naszym przypadku jest to widok `ListView` prezentujący listę opcji:

```
private void selectItem(int position) {
    ...
    // Zamykamy szufładę nawigacyjną
    DrawerLayout drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawerLayout.closeDrawer(drawerList);
}
```

Pobieramy referencję do układu DrawerLayout.

drawerLayout to szufłada nawigacyjna układu DrawerLayout. To wywołanie nakazuje zamknięcie tej szufłady.



Układ DrawerLayout obejmuje cały ekran. Zawiera on układ FrameLayout używany do wyświetlania zawartości i widok ListView tworzący szufładę nawigacyjną.

Musimy nakazać układowi DrawerLayout zamknięcie szufłady nawigacyjnej.



Znasz już wszystkie elementy niezbędne do zaimplementowania metody `selectItem()`, zobaczmy więc, jak wygląda kompletna implementacja tego rozwiązania i jak jej użyć w kodzie aktywności `MainActivity`.

Zaktualizowany kod pliku MainActivity.java

Oto zaktualizowana zawartość pliku *MainActivity.java*:

```
package com.hfad.wloskieconieco;  
...
```

```
import android.support.v4.widget.DrawerLayout;
```

← *DrawerLayout należy do biblioteki Support Library v4.*

```
public class MainActivity extends Activity {
```

```
...
```

```
private DrawerLayout drawerLayout;
```

← *Dodajemy DrawerLayout jako zmienną prywatną, gdyż będziemy jej używać w wielu metodach.*

```
private class DrawerItemClickListener implements ListView.OnItemClickListener {
```

```
@Override
```

```
public void onItemClick(AdapterView<?> parent, View view, int position, long id){
```

```
    // Kod wykonywany po kliknięciu elementu szuflady nawigacyjnej
```

```
    selectItem(position);
```

← *Wywołujemy metodę selectItem().*

```
}
```

```
};
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_main);
```

```
    titles = getResources().getStringArray(R.array.titles);
```

```
    drawerList = (ListView)findViewById(R.id.drawer);
```

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

```
    // Określamy zawartość widoku ListView
```

```
    drawerList.setAdapter(new ArrayAdapter<String>(this,  
        android.R.layout.simple_list_item_activated_1, titles));
```

```
    drawerList.setOnItemClickListener(new DrawerItemClickListener());
```

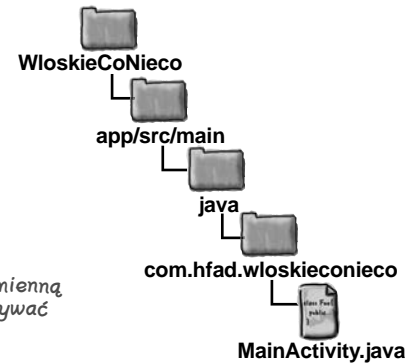
```
    if (savedInstanceState == null) {
```

```
        selectItem(0);
```

← *Jeśli aktywność MainActivity została właśnie utworzona, to używamy wywołania selectItem(), by wyświetlić w niej fragment Topfragment.*

```
}
```

- Dodanie fragmentów
- Utworzenie szuflady
- Obsługa kliknięcia ListView
- Dodanie ActionBarDrawerToggle



← *Pobieramy referencję do układu DrawerLayout.*

→ *Dalsza część kodu znajduje się na następnej stronie.*

Kod pliku MainActivity.java (ciąg dalszy)

```

private void selectItem(int position) {
    // Aktualizujemy główną zawartość aplikacji, podmieniając prezentowany fragment
    Fragment fragment;
    switch(position) {
        case 1:
            fragment = new PizzaFragment();
            break;
        case 2:
            fragment = new PastaFragment();
            break;
        case 3:
            fragment = new StoresFragment();
            break;
        default:
            fragment = new TopFragment();
    }
    FragmentTransaction ft = getFragmentManager().beginTransaction();
    ft.replace(R.id.content_frame, fragment);
    ft.addToBackStack(null);
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
    ft.commit();
    // Ustawiamy tytuł paska akcji
    setActionBarTitle(position);
    // Zamykamy szufladę nawigacyjną
    drawerLayout.closeDrawer(drawerList);
}

private void setActionBarTitle(int position) {
    String title;
    if (position == 0){
        title = getResources().getString(R.string.app_name);
    } else {
        title = titles[position];
    }
    getActionBar().setTitle(title);
}
...

```

Tworzymy odpowiedni fragment.

Wyświetlamy fragment, używając w tym celu transakcji fragmentu.

Określamy tytuł paska akcji.

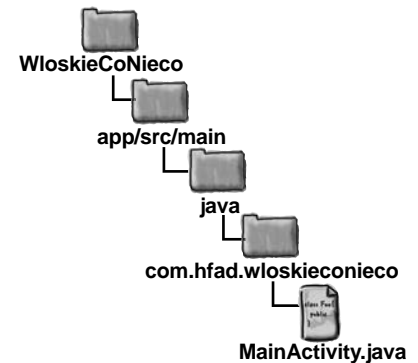
Zamykamy szufladę nawigacyjną.

Jeśli użytkownik kliknął opcję „Ekran główny”, to w tytule paska akcji wyświetlamy nazwę aplikacji.

W przeciwnym razie wyświetlamy tańcuch z tablicy titles, określony na podstawie pozycji klikniętego elementu listy.

Określamy tytuł paska akcji.

Pominęliśmy tutaj metody onCreateOptionsMenu() i onOptionsItemSelected() z wcześniejszego kodu aktywności MainActivity, gdyż nie wprowadzaliśmy w nich żadnych zmian.



Zamykanie i otwieranie szuflady

Do tej pory dodaliśmy szufladę nawigacyjną do aktywności `MainActivity`, dodaliśmy do jej listy główne węzły aplikacji oraz zadaliśmy o to, by szuflada reagowała na zdarzenia kliknięcia. Następną sprawą, którą się zajmiemy, będzie poznanie sposobów otwierania i zamykania szuflady oraz reagowania na jej stan.

Można wskazać kilka powodów, dla których moglibyśmy chcieć reagować na stan szuflady nawigacyjnej. Przede wszystkim możemy chcieć zmieniać tytuł wyświetlany na pasku akcji w momencie otwierania i zamykania szuflady. Na przykład kiedy szuflada będzie otworzona, możemy wyświetlać na pasku akcji nazwę aplikacji, a następnie po wyświetleniu wybranego fragmentu i zamknięciu szuflady — tytuł fragmentu.

Kolejny powód jest powiązany z elementami wyświetlanymi na pasku akcji. Możemy uznać za stosowne, by po otwarciu szuflady ukryć wszystkie lub niektóre z nich, tak by użytkownik mógł z nich korzystać wyłącznie wtedy, kiedy szuflada będzie zamknięta.

Na kilku następnych stronach pokażemy Ci, jak przygotować obiekt nasłuchujący `DrawerListener`, który pozwoli nasłuchiwać zdarzeń związanych z szufladą nawigacyjną i je obsługiwać. Zastosujemy je do ukrywania akcji `Udostępnij` na pasku akcji w momencie otwierania szuflady i do jej wyświetlania, gdy szuflada zostanie zamknięta.

- Dodanie fragmentów
- Utworzenie szuflady
- Obsługa kliknięcia `ListView`
- Dodanie `ActionBarDrawerToggle`

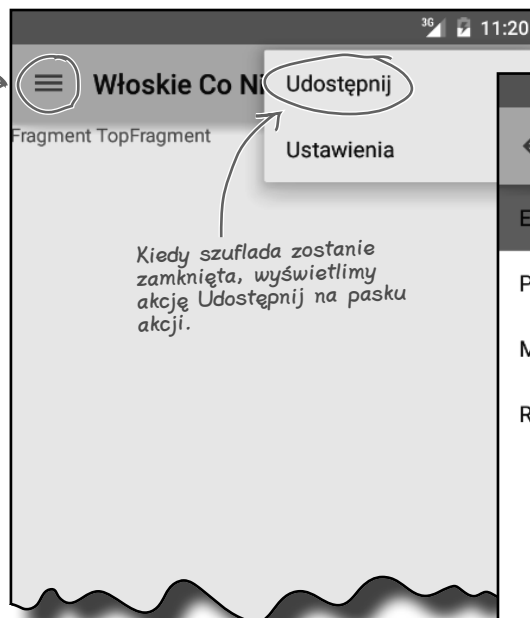


Spokojnie

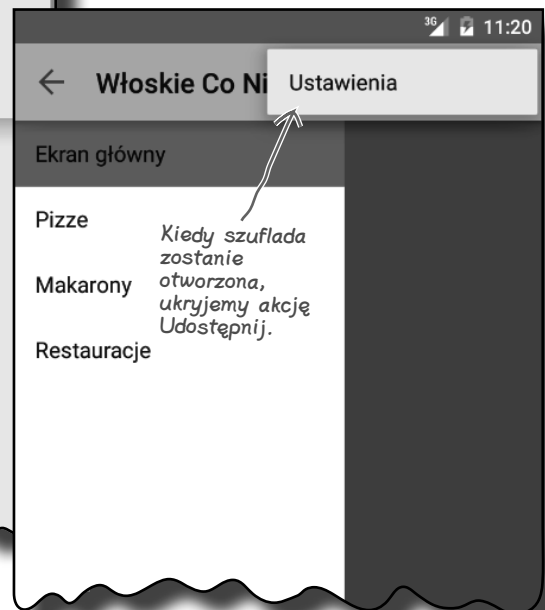
Nie sposób oprzeć się wrażeniu, że implementacja szuflady nawigacyjnej wymaga dużego nakładu pracy.

Choć kod tego rozwiązania wydaje się dosyć złożony, wystarczy go zastosować i wszystko będzie dobrze.

Szufladę otwieramy i zamykamy, używając przycisku umieszczonego na pasku akcji.



Kiedy szuflada zostanie zamknięta, wyświetlimy akcję `Udostępnij` na pasku akcji.



Kiedy szuflada zostanie otworzona, ukryjemy akcję `Udostępnij`.

Stosowanie ActionBarDrawerToggle

Najlepszym sposobem na utworzenie obiektu nasłuchującego DrawerListener jest skorzystanie z przycisku klasy **ActionBarDrawerToggle**. Jest to specjalny rodzaj komponentu implementującego interfejs DrawerListener przystosowany do wykorzystania na pasku akcji. Pozwala on nasłuchiwać zdarzeń generowanych przez układ DrawerLayout, jak robią to wszystkie obiekty nasłuchujące DrawerListener, oraz otwierać i zamykać szufladę poprzez klikanie ikony umieszczonej na pasku akcji.

Zacniemy od utworzenia w pliku *strings.xml* dwóch zasobów łańcuchowych opisujących akcje otwarcia i zamknięcia szuflady; będą nam one potrzebne ze względu na zapewnienie odpowiedniej dostępności aplikacji:

```
<string name="open_drawer">Otwórz szufladę</string>
<string name="close_drawer">Zamknij szufladę</string>
```

Dodaj te zasoby do pliku *strings.xml*. Będą one potrzebne dla przycisku *ActionBarDrawerToggle*.

Następnie utwórz obiekt *ActionBarDrawerToggle*. W tym celu należy wywołać jego konstruktor i przekazać do niego cztery argumenty: obiekt *Context* (zazwyczaj jest to referencja *this* reprezentująca bieżący kontekst), obiekt *DrawerLayout* oraz dwa zasoby łańcuchowe. Potem musisz przesłonić dwie metody klasy *ActionBarDrawerToggle*: **onDrawerClosed()** i **onDrawerOpened()**:

Tworzymy obiekt klasy *ActionBarDrawerToggle*.

```
ActionBarDrawerToggle drawerToggle = new ActionBarDrawerToggle(this, drawerLayout,
    R.string.open_drawer, R.string.close_drawer) {
```

```
    // Wywoływana, kiedy stan szuflady odpowiada jej całkowitemu zamknięciu
```

```
    @Override
```

```
    public void onDrawerClosed(View view) {
        super.onDrawerClosed(view);
```

Ta metoda jest wywoływana, kiedy szuflada zostanie zamknięta.

```
        // Kod, który ma zostać wywołany, gdy szuflada zostanie zamknięta
```

```
    }
```

Ta metoda jest wywoływana, kiedy szuflada zostanie otworzona.

```
    // Wywoływana, kiedy stan szuflady odpowiada jej całkowitemu otwarciu
```

```
    @Override
```

```
    public void onDrawerOpened(View drawerView) {
        super.onDrawerOpened(drawerView);
```

```
        // Kod, który ma zostać wywołany, gdy szuflada zostanie otworzona
```

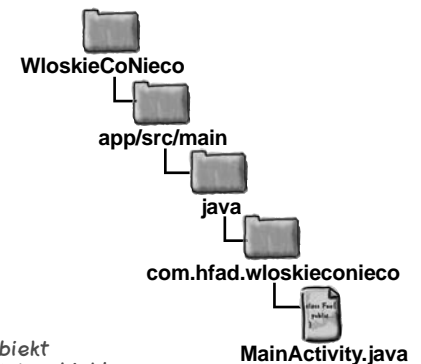
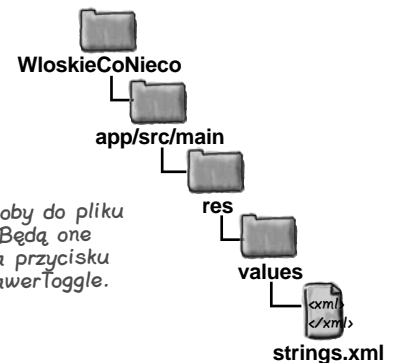
```
    }
```

```
};
```

Po utworzeniu obiektu *ActionBarDrawerToggle* należy przekazać go do układu *DrawerLayout*, wywołując jego metodę *setDrawerListener()*:

```
drawerLayout.setDrawerListener(drawerToggle);
```

To wywołanie ustawia obiekt *ActionBarDrawerToggle* jako obiekt nasłuchujący zdarzeń układu *DrawerLayout*.



Modyfikowanie elementów paska akcji w trakcie działania aplikacji

- Dodanie fragmentów
- Utworzenie szuflady
- Obsługa kliknięcia ListView
- Dodanie ActionBarDrawerToggle

Jeśli na pasku akcji znajdują się przyciski przeznaczone do użycia z konkretnym fragmentem, to możemy zdecydować się na ukrywanie ich w momencie otwierania szuflady i ponowne wyświetlanie po zamknięciu szuflady. Chcąc wprowadzać takie modyfikacje paska akcji, musimy wykonać dwie operacje.

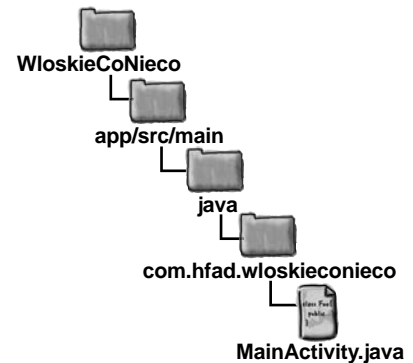
Pierwszą z nich jest wywołanie metody `invalidateOptionsMenu()`. W ten sposób informujemy system, że elementy, które mają się znajdować na pasku akcji, uległy zmianie i należy je odtworzyć.

W efekcie wywołania metody `invalidateOptionsMenu()` wywołana jest metoda `onPrepareOptionsMenu()` aktywności. Możemy ją przesłonić, by określić, jak mają się zmienić elementy paska akcji.

W naszej aplikacji chcemy zmienić widoczność akcji *Udostępnij* w zależności od stanu szuflady nawigacyjnej. Aby to zrobić, w metodach `onDrawerClosed()` i `onDrawerOpened()` obiektu `ActionBarDrawerToggle` musimy wywoływać metodę `invalidateOptionsMenu()`:

```
public void onDrawerClosed(View view) {  
    super.onDrawerClosed(view);  
    invalidateOptionsMenu();  
}  
  
public void onDrawerOpened(View drawerView) {  
    super.onDrawerOpened(drawerView);  
    invalidateOptionsMenu();  
}
```

Metoda `invalidateOptionsMenu()` informuje system o tym, że należy odtworzyć element menu. W naszym przypadku chcemy zmieniać widoczność opcji *Udostępnij* w zależności od tego, czy szuflada będzie otwarta czy zamknięta, dlatego wywołujemy ją w metodach `onDrawerOpened()` i `onDrawerClosed()`.



Następnie musimy zaimplementować metodę `onPrepareOptionsMenu()` i określić w niej widoczność akcji *Udostępnij*:

// Ta metoda jest wywoływana po każdym wywołaniu metody invalidateOptionsMenu()

@Override

```
public boolean onPrepareOptionsMenu(Menu menu) {  
    // Jeśli szuflada jest otworzona, ukrywamy elementy akcji związane  
    // z prezentowaną zawartością  
    boolean drawerOpen = drawerLayout.isDrawerOpen(drawerList);  
    menu.findItem(R.id.action_share).setVisible(!drawerOpen);  
    return super.onPrepareOptionsMenu(menu);  
}
```

Metoda `onPrepareOptionsMenu()` jest wywoływana za każdym razem, gdy zostanie wywołana metoda `invalidateOptionsMenu()`.

Akcję *Udostępnij* ukrywamy, jeśli szuflada jest widoczna (przekazując w wywołaniu metody `setVisible()` wartość `false`), i wyświetlamy, jeśli szuflada jest zamknięta (przekazując w wywołaniu wartość `true`).

Na następnej stronie przedstawimy cały kod związany z określaniem widoczności elementów paska akcji.

Zaktualizowany kod aktywności MainActivity

Oto zmodyfikowana zawartość pliku *MainActivity.java*:

```

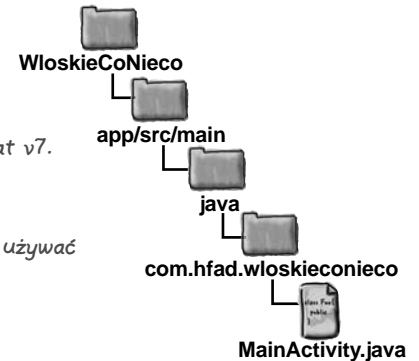
...
import android.support.v7.app.ActionBarDrawerToggle;

public class MainActivity extends Activity {
    ...
    private ActionBarDrawerToggle drawerToggle;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // Tworzymy obiekt ActionBarDrawerToggle
        drawerToggle = new ActionBarDrawerToggle(this, drawerLayout,
            R.string.open_drawer, R.string.close_drawer) {
            // Wywoływana, kiedy stan szuflady odpowiada jej całkowitemu zamknięciu
            @Override
            public void onDrawerClosed(View view) {
                super.onDrawerClosed(view);
                invalidateOptionsMenu();
            }
            // Wywoływana, kiedy stan szuflady odpowiada jej całkowitemu otwarciu
            @Override
            public void onDrawerOpened(View drawerView) {
                super.onDrawerOpened(drawerView);
                invalidateOptionsMenu();
            }
        };
        drawerLayout.setDrawerListener(drawerToggle);
    }

    // Ta metoda jest wywoływana po każdym wywołaniu metody invalidateOptionsMenu()
    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
        // Jeśli szuflada jest otworzona, ukrywamy elementy akcji związane
        // z prezentowaną zawartością
        boolean drawerOpen = drawerLayout.isDrawerOpen(drawerList);
        menu.findItem(R.id.action_share).setVisible(!drawerOpen);
        return super.onPrepareOptionsMenu(menu);
    }
    ...
}

```



Włączenie możliwości otwierania i zamykania szuflady nawigacyjnej

Dodaliśmy już szufladę nawigacyjną do aktywności `MainActivity`, dodaliśmy opcje do jej listy, zapewniliśmy, że aktywność reaguje na kliknięcia opcji, oraz dowiedzieliśmy się, jak ukrywać elementy wyświetlane na pasku akcji po otwarciu szuflady. Ostatnią rzeczą, jaką zrobimy, będzie zapewnienie użytkownikowi możliwości otwierania i zamykania szuflady nawigacyjnej za pomocą ikony umieszczonej na pasku akcji.

Jak już zaznaczyliśmy wcześniej, ta funkcjonalność jest jedną z zalet stosowania komponentu `ActionBarDrawerToggle`. Aby z niej skorzystać, musimy dodać do aktywności trochę nowego kodu. Najpierw pokażemy Ci po kolei poszczególne modyfikacje, które należy wprowadzić, a następnie, na samym końcu rozdziału, przedstawimy kompletny kod aktywności `MainActivity`.

W pierwszej kolejności zadamy o wyświetlenie na pasku akcji odpowiedniej ikony. W tym celu musimy dodać do kodu metody `onCreate()` dwa poniższe wywołania:

```
getActionBar().setDisplayHomeAsUpEnabled(true);  
getActionBar().setHomeButtonEnabled(true);
```

Włączamy przycisk W górę, by móc używać go do otwierania i zamykania szuflady.

Te dwa wywołania spowodują, że w aktywności będzie wyświetlany przycisk *W górę*. Ponieważ używamy przycisku `ActionBarDrawerToggle`, przycisk *W górę* zamiast nawigowania w górę hierarchii aplikacji będzie powodował otwieranie i zamykanie szuflady nawigacyjnej.

Następnie musimy zadbać o to, by przycisk `ActionBarDrawerToggle` reagował na kliknięcia. W tym celu w metodzie `onOptionsItemSelected()` aktywności musimy wywołać metodę `onOptionsItemSelected()` przycisku. Poniżej pokazaliśmy, jak należy to zrobić:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    if (drawerToggle.onOptionsItemSelected(item)) {  
        return true;  
    }  
    // Kod obsługujący pozostałe elementy paska akcji  
    ...  
}
```

Musimy dodać ten fragment kodu do metody `onOptionsItemSelected()`, aby przycisk `ActionBarDrawerToggle` zaczął reagować na kliknięcia.

Poniższe wywołanie:

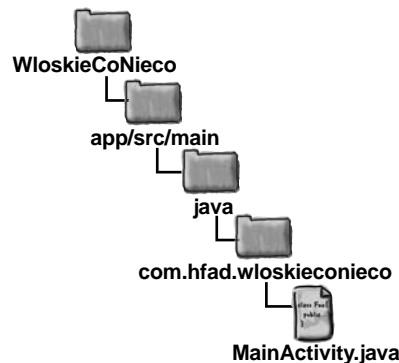
```
drawerToggle.onOptionsItemSelected(item)
```

zwraca wartość `true`, jeśli komponent `ActionBarDrawerToggle` obsłużył kliknięcie. Jeżeli wywołanie zwróci wartość `false`, będzie to oznaczać, że został kliknięty inny element umieszczony na pasku akcji, a zatem zostanie wykonany dalszy kod metody `onOptionsItemSelected()`.

420 Rozdział 10.

- Dodanie fragmentów
- Utworzenie szuflady
- Obsługa kliknięcia `ListView`
- Dodanie `ActionBarDrawerToggle`

`ActionBarDrawerToggle` pozwala używać przycisku *W górę* do otwierania i zamykania szuflady nawigacyjnej.



Synchronizacja stanu przycisku ActionBarDrawerToggle

Musimy jeszcze wprowadzić dwie modyfikacje, by zapewnić poprawne działanie przycisku ActionBarDrawerToggle.

Pierwszą z nich jest wywołanie metody `syncState()` przycisku ActionBarDrawerToggle w metodzie `postCreate()` aktywności. Metoda `syncState()` synchronizuje stan ikony szufłady ze stanem układu DrawerLayout.

Byłoby super, gdyby szuflada nawigacyjna mogła sama o to zadbać, ale nie może. Musimy to zrobić sami.

Synchronizacja stanu oznacza, że ikona szufłady ma inną postać, gdy szuflada jest otworzona, a inną, gdy jest zamknięta.



Metodę `syncState()` musimy wywołać w metodzie `onPostCreate()` aktywności, aby po utworzeniu aktywności przycisk ActionBarDrawerToggle był w odpowiednim stanie:

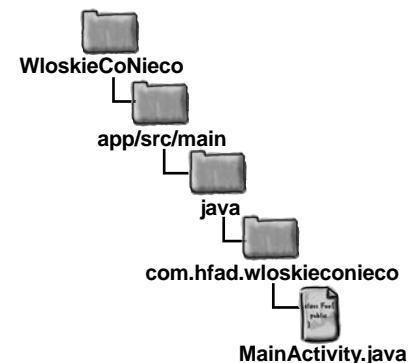
```
@Override
protected void onPostCreate(Bundle savedInstanceState) {
    super.onPostCreate(savedInstanceState);
    // Synchronizujemy stan przycisku przełącznika po wywołaniu metody onRestoreInstanceState
    drawerToggle.syncState();
}
```

Musimy dodać tę metodę do aktywności, aby stan przycisku ActionBarDrawerToggle był zsynchronizowany ze stanem szuflady nawigacyjnej.

I w końcu, jeśli konfiguracja urządzenia ulegnie zmianie, musimy przekazać klasie ActionBarDrawerToggle informacje na jej temat. Możemy to zrobić, wywołując w metodzie `onConfigurationChanged()` aktywności metodę `onConfigurationChanged()` klasy ActionBarDrawerToggle:

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    drawerToggle.onConfigurationChanged(newConfig);
}
```

Na kolejnej stronie pokażemy Ci, w którym miejscu pliku `MainActivity.java` należy wprowadzić ostatnie zmiany, a potem sprawdzimy, co się stanie po uruchomieniu aplikacji.

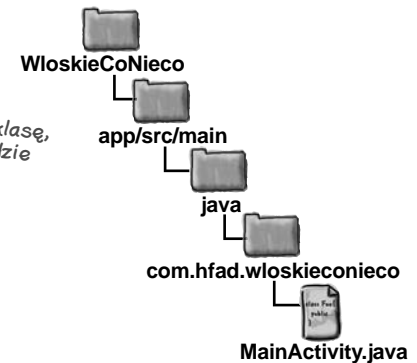


Tę metodę musisz dodać do aktywności, tak by wszystkie zmiany konfiguracji były przekazywane do klasy ActionBarDrawerToggle.

Zaktualizowany kod aktywności MainActivity

Oto zaktualizowana zawartość pliku *MainActivity.java*:

- Dodanie fragmentów
- Utworzenie szuflady
- Obsługa kliknięcia ListView
- Dodanie ActionBarDrawerToggle



```
...
import android.content.res.Configuration;

public class MainActivity extends Activity {
    ...
    private ActionBarDrawerToggle drawerToggle;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        getActionBar().setDisplayHomeAsUpEnabled(true);
        getActionBar().setHomeButtonEnabled(true);
    }

    @Override
    protected void onPostCreate(Bundle savedInstanceState) {
        super.onPostCreate(savedInstanceState);
        drawerToggle.syncState();
    }

    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
        drawerToggle.onConfigurationChanged(newConfig);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if (drawerToggle.onOptionsItemSelected(item)) {
            return true;
        }
        // Kod obsługujący pozostałe przyciski umieszczone na pasku akcji
        switch (item.getItemId()) {
            ...
        }
    }
}
```

Musimy zaimportować tę klasę, gdyż używamy jej w metodzie `onConfigurationChanged()`.

Włączamy przycisk W górę, dzięki czemu będzie on mógł być używany przez `ActionBarDrawerToggle`.

Synchronizujemy stan przycisku `ActionBarDrawerToggle` ze stanem szuflady nawigacyjnej.

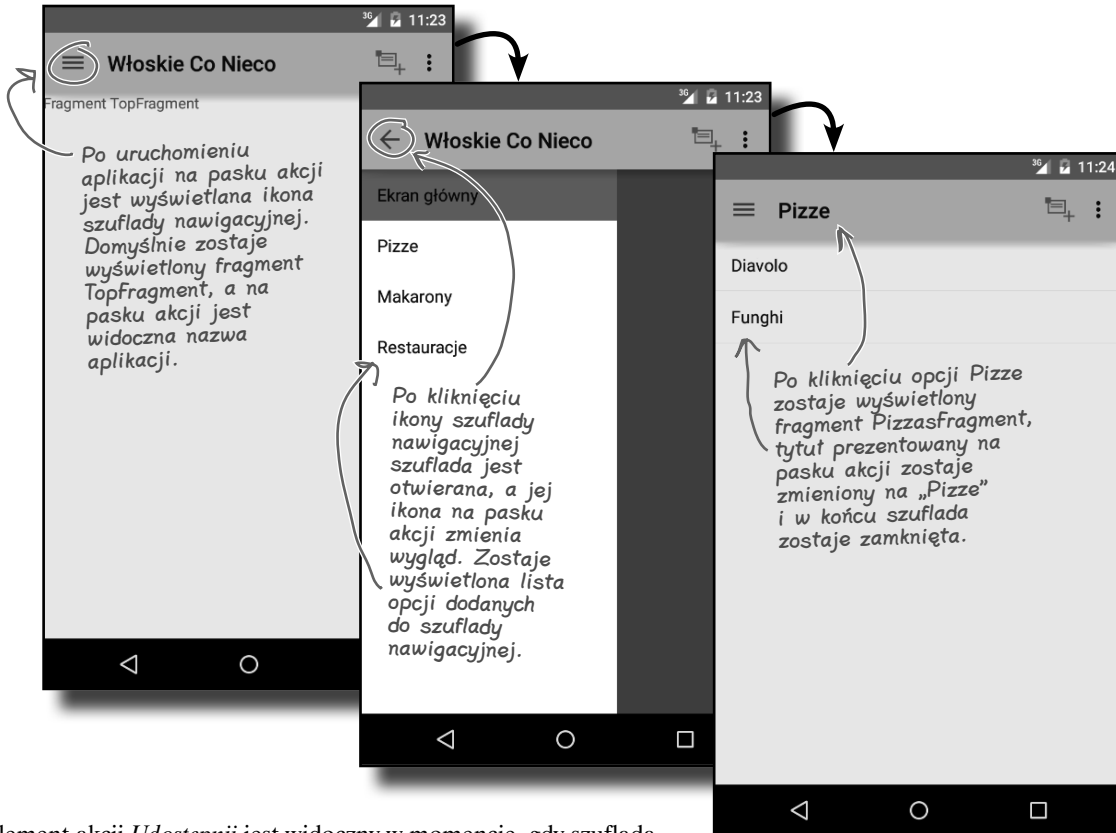
Ten parametr przekazuje wszelkie zmiany konfiguracji do przycisku `ActionBarDrawerToggle`.

Dzięki temu wywołaniu przycisk `ActionBarDrawerToggle` będzie mógł obsługiwać kliknięcia.

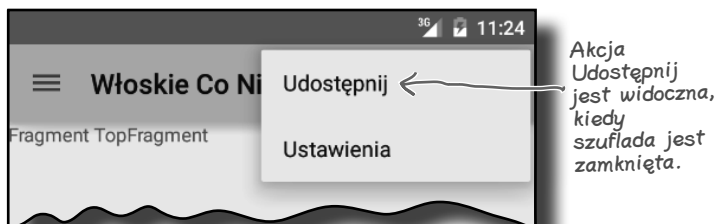


Jazda próbna aplikacji

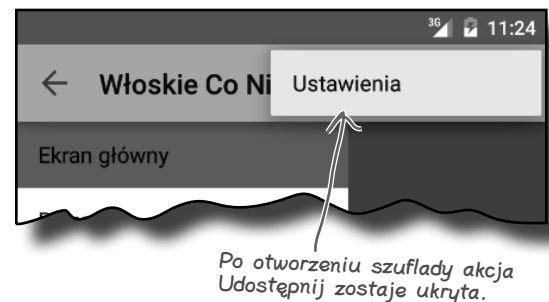
Po uruchomieniu aplikacji zostaje wyświetlona aktywność `MainActivity`. Jak widać, dysponuje już ona działającą szufladą nawigacyjną:



Element akcji `Udostępnij` jest widoczny w momencie, gdy szuflada nawigacyjna jest zamknięta, natomiast po jej otwarciu zostaje on ukryty:



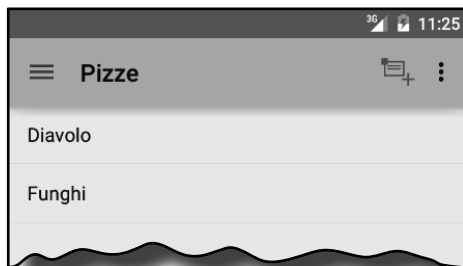
Jest jeszcze jednak rzecz, którą trzeba się zająć: musimy upewnić się, że po obróceniu urządzenia lub kliknięciu przycisku `Wstecz` na pasku akcji będzie wyświetlany odpowiedni tytuł. A jak to aktualnie wygląda?



Tytuł i fragment nie są zsynchronizowane

Kiedy klikniemy jedną z opcji dostępnych w szufladzie nawigacyjnej, tytuł wyświetlany na pasku akcji odpowiada aktualnie prezentowanemu fragmentowi. Na przykład jeśli klikniemy opcję *Pizze*, to na pasku akcji zostanie wyświetlony tytuł „Pizze”:

Kiedy klikniesz element w szufladzie nawigacyjnej, tytuł na pasku akcji zostanie prawidłowo zaktualizowany.



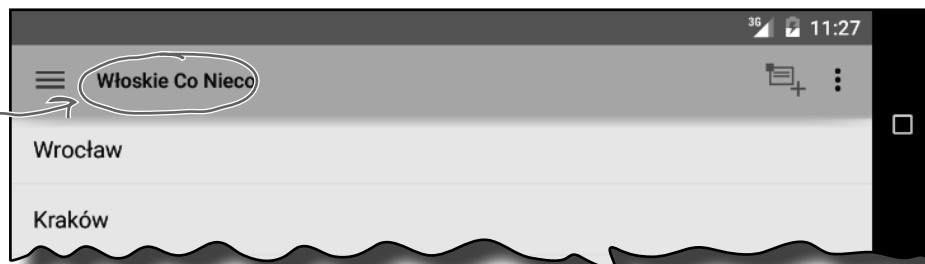
Kiedy jednak klikniemy przycisk *Wstecz*, tytuł na pasku akcji nie jest aktualizowany i przestaje odpowiadać wyświetlanemu fragmentowi. Na przykład założmy, że w szufladzie nawigacyjnej kliknęliśmy opcję *Restauracje*, a następnie opcję *Pizze*. W efekcie na ekranie zostanie wyświetlona lista pizz, a tytuł na pasku akcji będzie temu odpowiadał. Jeśli teraz klikniemy przycisk *Wstecz*, to ponownie zostanie wyświetlony fragment *StoresFragment*, ale tytuł na pasku akcji nie zmieni się i dalej będzie na nim widoczny tekst „Pizze”:

Kiedy klikniemy przycisk Wstecz, tytuł prezentowany na pasku akcji nie zmienia się. W tym przykładzie na pasku wciąż jest widoczny tytuł „Pizze”, choć poniżej została wyświetlona lista miast, w których znajdują się restauracje.



Z kolei jeśli obrócimy urządzenie, to na pasku akcji, niezależnie od aktualnie prezentowanego fragmentu, zostanie wyświetlona nazwa aplikacji, czyli „Włoskie Co Nieco”:

Po obróceniu urządzenia tytuł na pasku akcji zostaje przywrócony do stanu początkowego.



Spróbujmy teraz rozwiązać oba te problemy, zaczynając od zachowania synchronizacji paska akcji w przypadku zmiany orientacji urządzenia.

Obsługa zmian konfiguracji

Jak już wiesz, w przypadku obrócenia urządzenia aktualnie prezentowana aktywność jest usuwana i ponownie tworzona. Oznacza to, że wszelkie zmiany wprowadzone w interfejsie użytkownika zostają utracone; dotyczy to także zmian wprowadzanych na pasku akcji.

Aby rozwiązać ten problem, skorzystamy z rozwiązania, które stosowaliśmy już w poprzednich rozdziałach — użyjemy metody `onSaveInstanceState()` do zapisania pozycji aktualnie wybranego elementu szufłady nawigacyjnej. Następnie skorzystamy z tej informacji w metodzie `onCreate()`, by zaktualizować tytuł wyświetlany na pasku akcji.

Poniżej pokazaliśmy zmiany, które należy wprowadzić w kodzie aktywności:

```

...
public class MainActivity extends Activity {
    ...
    private int currentPosition = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // Wyświetlamy odpowiedni fragment.
        if (savedInstanceState != null) {
            currentPosition = savedInstanceState.getInt("position");
            setActionBarTitle(currentPosition);
        } else {
            selectItem(0);
        }
        ...
    }

    private void selectItem(int position) {
        currentPosition = position;
        ...
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putInt("position", currentPosition);
    }
    ...
}

```

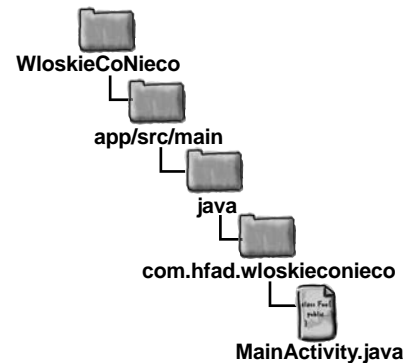
Zmiennej `currentPosition` przypisujemy domyślnie wartość 0.

Jeśli aktywność została utworzona po raz pierwszy, to wyświetlamy fragment `TopFragment`.

Jeśli aktywność została usunięta i ponownie odtworzona, to wartość zmiennej `currentPosition` określamy na podstawie zapisanego stanu aktywności, a następnie używamy jej do określenia tytułu na pasku akcji.

Aktualizujemy wartość zmiennej `currentPosition` po wybraniu elementu z szufłady nawigacyjnej.

Jeśli aktywność ma zostać usunięta, to zapisujemy stan zmiennej `currentPosition`.



Reagowanie na zmiany stosu cofnięć

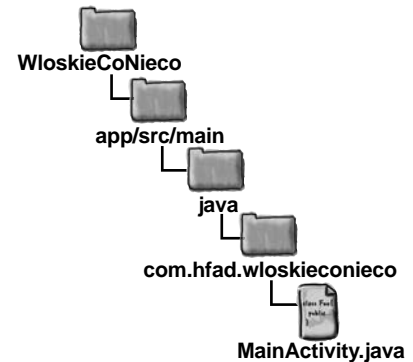
Ostatnią rzeczą, którą musimy się zająć, będzie aktualizowanie tytułu wyświetlanego na pasku akcji w przypadku, gdy użytkownik naciśnie klawisz *Wstecz*. Możemy to zrobić poprzez dodanie do menedżera fragmentów aktywności obiektu nasłuchującego typu **FragmentManager.OnBackStackChangedListener**.

Interfejs **FragmentManager.OnBackStackChangedListener** pozwala nasłuchiwać zmian zachodzących na stosie cofnięć. Dotyczy to takich zdarzeń jak dodanie na stos nowej transakcji fragmentu i naciśnięcie przycisku *Wstecz* w celu przywrócenia poprzedniego elementu zapisanego na stosie.

Obiekt nasłuchujący typu **OnBackStackChangedListener** można dodać do menedżera fragmentów aplikacji w poniższy sposób:

```
getFragmentManager().addOnBackStackChangeListener(  
    new FragmentManager.OnBackStackChangeListener() {  
        public void onBackStackChanged() {  
            // Kod wykonywany po zmianie stanu stosu cofnięć  
        }  
    }  
);
```

Musimy dodać nowy obiekt **FragmentManager.OnBackStackChangedListener** i zaimplementować jego metodę **onBackStackChanged()**. Ta metoda jest wywoływana za każdym razem, gdy zmienia się stan stosu cofnięć.



Kiedy zmieni się stan stosu cofnięć, zostaje wywołana metoda **onBackStackChanged()** obiektu **OnBackStackChangedListener**. To właśnie w niej należy umieścić kod, który chcemy wykonywać w przypadku, gdy użytkownik naciśnie przycisk *Wstecz*.

W naszej aplikacji po naciśnięciu przez użytkownika przycisku *Wstecz* chcemy wykonać trzy operacje:

- ★ Zaktualizować wartość zmiennej `currentPosition`, tak by odpowiadała pozycji aktualnie wyświetlonego fragmentu na liście opcji szuflady nawigacyjnej.
- ★ Wywołać metodę `setActionBarTitle()`, przekazując przy tym do niej wartość zmiennej `currentPosition`.
- ★ Upewnić się, że na liście opcji szuflady nawigacyjnej zostanie zaznaczona odpowiednia opcja, wywołując w tym celu metodę `setItemChecked()`.

Wykonanie każdej z tych czynności wymaga znajomości pozycji, którą na liście opcji szuflady nawigacyjnej zajmuje aktualnie wyświetlony fragment. A jak możemy tę pozycję określić?

Dodawanie znaczników do fragmentów

Aby się dowiedzieć, jaką wartość powinna przyjąć zmienna `currentPosition`, sprawdzimy typ aktualnie prezentowanego fragmentu. Na przykład jeśli tym fragmentem jest obiekt klasy `PizzaFragment`, to zmiennej `currentPosition` przypiszemy wartość 1.

Referencję do fragmentu, który aktualnie jest dołączony do aktywności, pobierzemy, dodając do każdego fragmentu łańcuchowy znacznik. Następnie użyjemy metody `findFragmentByTag()` menedżera fragmentów, by pobrać ten fragment.

Znacznik można dodawać do fragmentu w ramach transakcji fragmentu. Poniżej przedstawiliśmy aktualny kod realizujący transakcję fragmentu, który jest umieszczony w metodzie `selectItem()` i który zmienia aktualnie wyświetlony fragment:

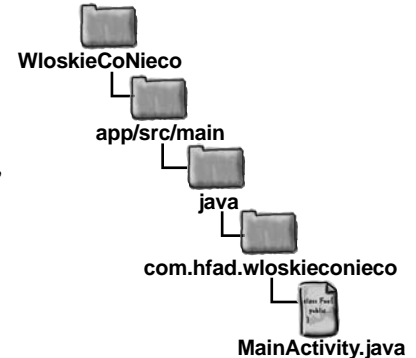
```
FragmentTransaction ft = getFragmentManager().beginTransaction();
ft.replace(R.id.content_frame, fragment);
ft.addToBackStack(null);
ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
ft.commit();
```

Aby dodać znacznik, musimy umieścić w wywołaniu metody `replace()` dodatkowy argument typu `String`:

```
FragmentTransaction ft = getFragmentManager().beginTransaction();
ft.replace(R.id.content_frame, fragment, "visible_fragment");
ft.addToBackStack(null);
ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
ft.commit();
```

W powyższym kodzie dodaliśmy do wywołania metody `replace()` znacznik `"visible_fragment"`. Znacznikiem tym zostanie opatrzony każdy fragment wyświetlany w aktywności `MainActivity`.

Teraz zastosujemy metodę `findFragmentByTag()` menedżera fragmentów, by pobrać referencję do fragmentu, który jest aktualnie dołączony do aktywności.



Ten dodatkowy argument powoduje dodanie znacznika „visible_fragment” do fragmentu, który jest umieszczany na stosie cofnięć.

Odnajdywanie fragmentu na podstawie znacznika

Aby pobrać fragment, który w danym momencie jest dołączony do aktywności, użyjemy metody `findFragmentByTag()`, przekazując w jej wywołaniu znacznik, którego użyliśmy podczas wykonywania transakcji fragmentu:

```
FragmentManager fragMan = getFragmentManager();
Fragment fragment = fragMan.findFragmentByTag("visible_fragment");
```

Poszukujemy fragmentu ze znacznikiem „visible_fragment”.

Metoda `findFragmentByTag()` rozpoczyna działanie od przeszukania wszystkich fragmentów dołączonych do aktywności. Jeśli nie uda się jej znaleźć fragmentu opatrzonego podanym znacznikiem, to metoda przeszuka fragmenty umieszczone na stosie cofnięć. Dzięki dodaniu tego samego znacznika "visible_fragment" do wszystkich fragmentów powyższy kod zwróci referencję do fragmentu, który jest aktualnie dołączony do aktywności.

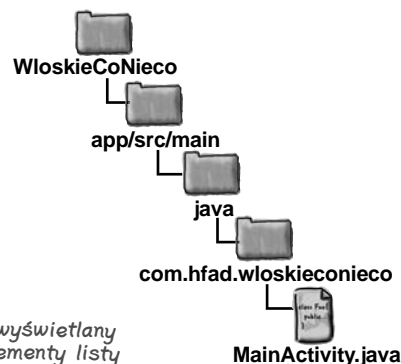
Poniżej przedstawiliśmy pełny kod naszej implementacji obiektu `OnBackStackChangedListener`. Najpierw, przy użyciu metody `findFragmentByTag()`, pobieramy referencję do fragmentu aktualnie dołączonego do aktywności. Potem sprawdzamy typ pobranej instancji fragmentu i na jego podstawie określamy wartość zmiennej `currentPosition`:

```
getFragmentManager().addOnBackStackChangedListener(
    new FragmentManager.OnBackStackChangedListener() {
        public void onBackStackChanged() {
            FragmentManager fragMan = getFragmentManager();
            Fragment fragment = fragMan.findFragmentByTag("visible_fragment");
            if (fragment instanceof TopFragment) {
                currentPosition = 0;
            }
            if (fragment instanceof PizzaFragment) {
                currentPosition = 1;
            }
            if (fragment instanceof PastaFragment) {
                currentPosition = 2;
            }
            if (fragment instanceof StoresFragment) {
                currentPosition = 3;
            }
            setActionBarTitle(currentPosition);
            drawerList.setItemChecked(currentPosition, true);
        }
    }
);
```

To wywołanie zwraca fragment, który w danej chwili jest dołączony do aktywności.

Te instrukcje sprawdzają typ fragmentu i na jego podstawie określają wartość zmiennej `currentPosition`.

Te dwa ostatnie wywołania określają tytuł wyświetlany na pasku akcji i wyróżniają odpowiednie elementy listy w szufladzie nawigacyjnej.



To już cały kod niezbędny do zapewnienia prawidłowej synchronizacji tytułu wyświetlanego na pasku akcji z aktualnie prezentowanym fragmentem w przypadku naciskania przycisku *Wstecz*. Zanim sprawdzimy, jak działa nasza aplikacja po wprowadzeniu tych zmian, pokażemy kompletny kod aktywności `MainActivity`.

Kompletny kod aktywności MainActivity

Oto kompletna zawartość pliku *MainActivity.java*:

```
package com.hfad.wloskieconieco;

import android.app.Activity;
import android.app.Fragment;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.content.Intent;
import android.content.res.Configuration;
import android.os.Bundle;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.ActionBarDrawerToggle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.ShareActionProvider;
```

Używamy klasy *FragmentManager*,
zatem musimy ją zaimportować.



To są wszystkie klasy używane
w kodzie aktywności.



```
public class MainActivity extends Activity {
```

```
    private ShareActionProvider shareActionProvider;
    private String[] titles;
    private ListView drawerList;
    private DrawerLayout drawerLayout;
    private ActionBarDrawerToggle drawerToggle;
private int currentPosition = 0;
```

Używamy tych wszystkich
zmiennych prywatnych.



Metoda *onItemClick()* interfejsu
OnItemClickListener jest wywoływana, gdy
użytkownik kliknie elementy listy *ListView*
umieszczonej w szufładzie nawigacyjnej.

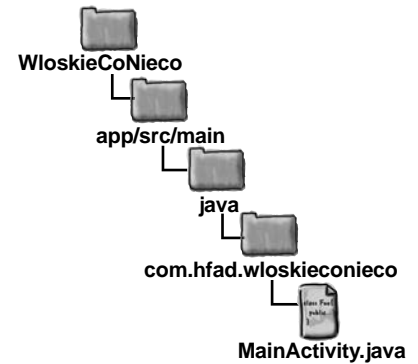


```
    private class DrawerItemClickListener implements ListView.OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id){
            // Kod wykonywany po kliknięciu elementu w szufładzie nawigacyjnej
            selectItem(position);
        }
    };
```

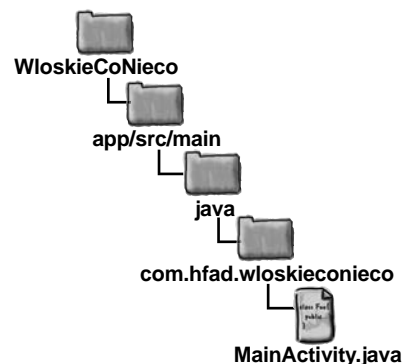
Kiedy element listy w szufładzie
nawigacyjnej zostanie kliknięty,
wywołujemy metodę *selectItem()*.



Dalszy ciąg kodu
znajduje się na
następnej stronie.



Plik MainActivity.java (ciąg dalszy)



```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    titles = getResources().getStringArray(R.array.titles);
    drawerList = (ListView) findViewById(R.id.drawer);
    drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
    // Określamy zawartość widoku ListView
    drawerList.setAdapter(new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_activated_1, titles));
    drawerList.setOnItemClickListener(new DrawerItemClickListener());
    // Wyświetlamy odpowiedni fragment
    if (savedInstanceState != null) {
        currentPosition = savedInstanceState.getInt("position");
        setActionBarTitle(currentPosition);
    } else {
        selectItem(0);
    }
    // Tworzymy obiekt ActionBarDrawerToggle
    drawerToggle = new ActionBarDrawerToggle(this, drawerLayout,
        R.string.open_drawer, R.string.close_drawer) {
        // Wywoływana, kiedy stan szuflady odpowiada jej całkowitemu zamknięciu
        @Override
        public void onDrawerClosed(View view) {
            super.onDrawerClosed(view);
            invalidateOptionsMenu();
        }
        // Wywoływana, kiedy stan szuflady odpowiada jej całkowitemu otwarciu
        @Override
        public void onDrawerOpened(View drawerView) {
            super.onDrawerOpened(drawerView);
            invalidateOptionsMenu();
        }
    };
}

```

Wypełniamy elementy listy ListView w szufladzie nawigacyjnej i zapewniamy, aby reagowała na kliknięcia.

Jeśli aktywność została usunięta i ponownie odtworzona, to ustawiamy prawidłowy tytuł na pasku akcji.

Domyślnie jest wyświetlany fragment Topfragment.

Podczas otwierania i zamykania szuflady nawigacyjnej wywołujemy metodę `invalidateOptionsMenu()`, gdyż chcemy zmieniać elementy wyświetlane na pasku akcji.

Dalszy ciąg kodu znajduje się na następnej stronie.

Plik MainActivity.java (ciąg dalszy)

Ten kod stanowi dalszą część metody onCreate().

```
drawerLayout.setDrawerListener(drawerToggle);
getActionBar().setDisplayHomeAsUpEnabled(true);
getActionBar().setHomeButtonEnabled(true);
```

← To wywołanie włącza przycisk W górę na pasku akcji, dzięki czemu będziemy mogli go używać do wyświetlania szufłady nawigacyjnej.

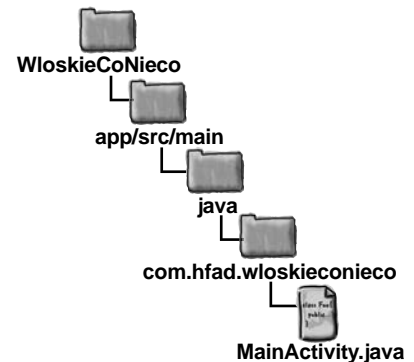
```
getFragmentManager().addOnBackStackChangeListener(
    new FragmentManager.OnBackStackChangeListener() {
```

Ta metoda jest wywoływana w przypadku zmiany stanu stosu cofnięć.

Te instrukcje warunkowe sprawdzają klasę fragmentu aktualnie dołączonego do aktywności i w zależności od niej odpowiednio ustawiają wartość zmiennej currentPosition.

```
public void onBackStackChanged() {
    FragmentManager fragMan = getFragmentManager();
    Fragment fragment = fragMan.findFragmentByTag("visible_fragment");
    if (fragment instanceof TopFragment) {
        currentPosition = 0;
    }
    if (fragment instanceof PizzaFragment) {
        currentPosition = 1;
    }
    if (fragment instanceof PastaFragment) {
        currentPosition = 2;
    }
    if (fragment instanceof StoresFragment) {
        currentPosition = 3;
    }
    setActionBarTitle(currentPosition);
    drawerList.setItemChecked(currentPosition, true);
}
}
```

↑ Te dwa wywołania ustawiają tytuł wyświetlany na pasku akcji i zaznaczają odpowiedni element listy w szufładzie nawigacyjnej.



Dalszy ciąg kodu znajduje się na następnej stronie. →

Plik MainActivity.java (ciąg dalszy)

Metoda `selectItem()` jest wywoływana, kiedy użytkownik kliknie jeden z elementów wyświetlonej w szufladzie nawigacyjnej.

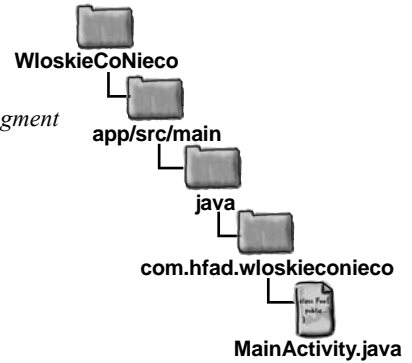
```
private void selectItem(int position) {  
    // Aktualizujemy główną zawartość aplikacji, podmieniając prezentowany fragment  
    currentPosition = position;  
    Fragment fragment;  
    switch(position) {  
        case 1:  
            fragment = new PizzaFragment();  
            break;  
        case 2:  
            fragment = new PastaFragment();  
            break;  
        case 3:  
            fragment = new StoresFragment();  
            break;  
        default:  
            fragment = new TopFragment();  
    }  
    FragmentTransaction ft = getFragmentManager().beginTransaction();  
    ft.replace(R.id.content_frame, fragment, "visible_fragment");  
    ft.addToBackStack(null);  
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);  
    ft.commit();  
    // Ustawiamy tytuł paska akcji  
    setActionBarTitle(position);  
    // Zamykamy szufladę nawigacyjną  
    drawerLayout.closeDrawer(drawerList);  
}
```

Na podstawie pozycji elementu zaznaczonego przez użytkownika w szufladzie nawigacyjnej określamy, który fragment należy wyświetlić.

Wyświetlamy fragment.

Wyświetlamy odpowiedni tytuł na pasku akcji.

Zamykamy szufladę nawigacyjną.



Dalszy ciąg kodu znajduje się na następnej stronie. →

Plik MainActivity.java (ciąg dalszy)

```

@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    // Jeśli szuflada jest otworzona, ukrywamy elementy akcji związane
    // z prezentowaną zawartością
    boolean drawerOpen = drawerLayout.isDrawerOpen(drawerList);
    menu.findItem(R.id.action_share).setVisible(!drawerOpen);
    return super.onPrepareOptionsMenu(menu);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Synchronizujemy stan przycisku przełącznika po wywołaniu
    // metody onRestoreInstanceState
    drawerToggle.syncState();
}

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    drawerToggle.onConfigurationChanged(newConfig);
}

@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putInt("position", currentPosition);
}

private void setActionBarTitle(int position) {
    String title;
    if (position == 0) {
        title = getResources().getString(R.string.app_name);
    } else {
        title = titles[position];
    }
    getActionBar().setTitle(title);
}

```

Akcję Udostępnij wyświetlamy, kiedy szuflada jest zamknięta, i chowamy, gdy jest otwarta.

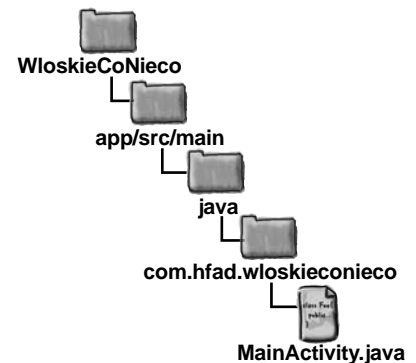
Synchronizujemy stan przycisku ActionBarDrawerToggle ze stanem szuflady nawigacyjnej.

Wszelkie szczegółowe informacje o zmianach konfiguracji przekazujemy do przycisku ActionBarDrawerToggle.

Jeśli aktywność ma zostać usunięta, to zapisujemy bieżący stan zmiennej currentPosition.

Ustawiamy tytuł wyświetlany na pasku akcji, tak by odpowiadał on aktualnie prezentowanemu fragmentowi.

Dalszy ciąg kodu znajduje się na następnej stronie.



Plik MainActivity.java (ciąg dalszy)

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Przygotowujemy menu; to wywołanie dodaje elementy do paska akcji, jeśli jest używany
    getMenuInflater().inflate(R.menu.menu_main, menu);
    MenuItem menuItem = menu.findItem(R.id.action_share);
    shareActionProvider = (ShareActionProvider) menuItem.getActionProvider();
    setIntent("To jest przykładowy tekst.");
    return super.onCreateOptionsMenu(menu);
}
```

← Ta metoda dodaje do paska akcji zawartość pliku zasobów menu.

```
private void setIntent(String text) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_TEXT, text);
    shareActionProvider.setShareIntent(intent);
}
```

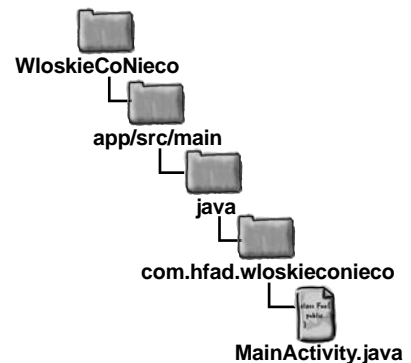
← Ta metoda tworzy intencję na potrzeby akcji Udostępnij.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (drawerToggle.onOptionsItemSelected(item)) {
        return true;
    }
    switch (item.getItemId()) {
        case R.id.action_create_order:
            // Kod wykonywany po kliknięciu przycisku Złóż zamówienie
            Intent intent = new Intent(this, OrderActivity.class);
            startActivity(intent);
            return true;
        case R.id.action_settings:
            // Kod wykonywany po kliknięciu przycisku Ustawienia
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

← Ta metoda jest wywoływana, kiedy użytkownik kliknie jakiś element paska akcji.

← Jeśli został kliknięty przycisk ActionBarDrawerToggle, pozwalamy mu wykonać to, co do niego należy.

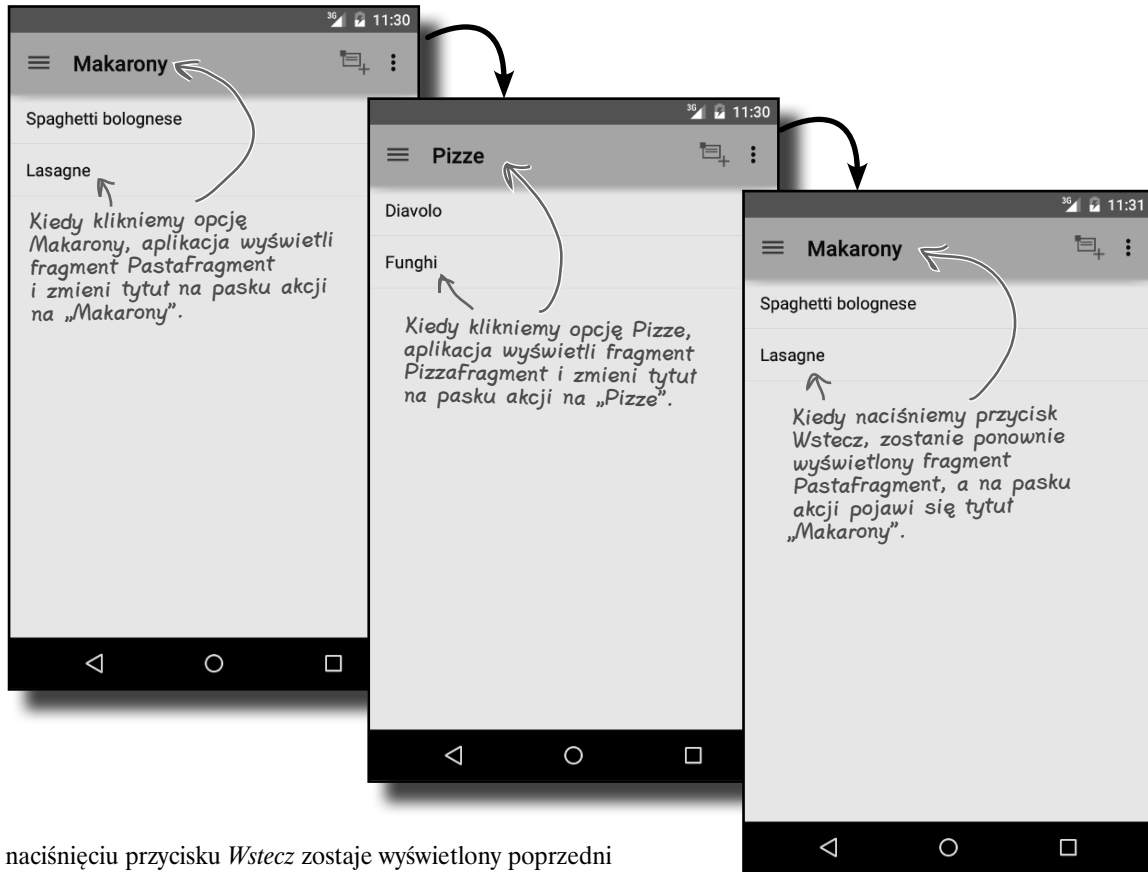
← W razie kliknięcia przycisku Złóż zamówienie uruchamiamy aktywność OrderActivity.



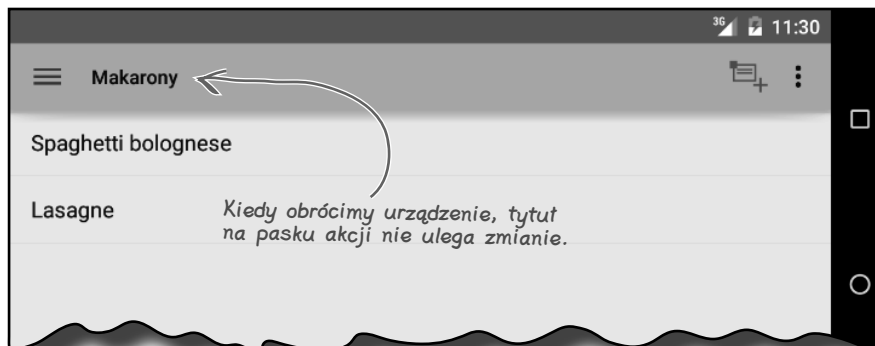


Jazda testowa aplikacji

Zobaczymy, co się stanie po uruchomieniu aplikacji.



Po naciśnięciu przycisku *Wstecz* zostaje wyświetlony poprzedni fragment, a tytuł na pasku akcji także jest odpowiednio zmieniany. Tytuł jest również odpowiednio synchronizowany z prezentowanym fragmentem w przypadku zmiany orientacji urządzenia.





Twój przybornik do Androida

Opanowałeś już rozdział 10. i dodałeś do swojego przybornika z narzędziami układy `DrawerLayout`.

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



CELNE SPOSTRZEŻENIA

- Użyj układu `DrawerLayout`, aby stworzyć aktywność z szufladą nawigacyjną. Zastosuj tę szufladę, by przechodzić do głównych węzłów aplikacji.
- Jeśli używasz paska akcji, to zastosuj przycisk typu `ActionBarDrawerToggle` jako obiekt nasłuchujący typu `DrawerToggle`. Dzięki temu będziesz mógł reagować na otwieranie i zamykanie szuflady nawigacyjnej i w odpowiedzi na te zdarzenia odpowiednio modyfikować elementy widoczne na pasku akcji.
- Do zmieniania elementów paska akcji w trakcie działania aplikacji musisz zastosować metodę `invalidateOptionsMenu()` i wprowadzić odpowiednie zmiany w metodzie `onPrepareOptionsMenu()` aktywności.
- Aby reagować na zmiany zachodzące na stosie cofnięć, należy zaimplementować interfejs `FragmentManager.OnBackStackChangedListener`.
- Metoda `findFragmentByTag()` menedżera fragmentów pozwala odszukać fragment na podstawie znacznika.

Skorowidz

- A**
- activity, *Patrz:* aktywność
 - adapter, 250, 632
 - ArrayAdapter, 251, 252, 291, 494
 - CursorAdapter, 494, 495, 497, 523
 - tworzenie, 498
 - RecyclerView.Adapter, 605
 - separacja od interfejsu, 633
 - ADB, 654, 656
 - akcja, 12, 96, 97, 113, 367
 - dodawanie elementu, 379
 - dostawca, *Patrz:* dostawca akcji
 - pasek, *Patrz:* pasek akcji
 - aktywność, 2, 12, 31, 74, 116
 - bieżąca, 347
 - Blank Activity, 13, 42
 - blokowanie ponownego uruchomienia, 136
 - CreateMessageActivity, 75
 - cykl życia, 133, 134, 135, 142, 148, 156, 327
 - na pierwszym planie, 151
 - widzialny, 143
 - deklarowanie, 81
 - domyślna, 107, 108
 - interakcja z bazą danych, 472, 475
 - interaktywna, 40
 - kategoria, *Patrz:* kategoria aktywność, kod, *Patrz:* plik MainActivity.java
 - listy, 247, 248, 257
 - tworzenie, 249
 - nadrzędna, 392
 - nazwa, 14
 - poziomu głównego, 229, 230, 232, 238, 293, 392, 507, 639
 - pusta, *Patrz:* aktywność Blank Activity stan, 133, 282
 - wstrzymana, 150
 - szczegółów/edycji, 229, 230, 259, 295
 - tworzenie, 13, 14, 40, 42, 78, 79
 - uruchamianie, 82, 84
 - wstrzymana, 150, 522
 - zapisywanie stanu bieżącego, 137
 - zastępca, 151, 152
 - Android Debug Bridge, *Patrz:* ADB
 - Android SDK, 5, 654
 - Android Studio, 4, 5, 7
 - błąd, 345
 - instalacja, 6
 - Android Virtual Device Manager, 24
 - Android wersja, 11
 - animacja, 598, 667
 - widoków, 667
 - ANT, 7
 - API, 3
 - poziom, 10, 368, 369
 - aplikacja, 2, 3, 31, 95, 96, 116, 129, 228
 - API, *Patrz:* API
 - bezpieczeństwo, 652
 - etykieta, *Patrz:* aplikacja nazwa ikona, *Patrz:* ikona
 - katalog, *Patrz:* katalog kompilowanie, 27
 - lokalizacja, 37
 - nawigowanie, 233, 367, 400
 - nazwa, 9, 385
 - nazwa firmowej domeny, 9
 - pakiet, *Patrz:* plik APK
 - rozpowszechnianie, 664
 - spakowanie, 27
 - stoper, *Patrz:* stoper
 - tworzenie, 8, 9, 10, 12, 13
 - udostępnianie, 664
 - uruchamianie
 - na fizycznym urządzeniu, 105, 113
 - w emulatorze, 23, 27, 113
 - wdrożenie, 27
 - węzeł kluczowy, 398
 - widżet, *Patrz:* widżet aplikacji
 - wydajność, 652
 - wykonanie, 27
 - źródło danych, 438
 - ART, 650
 - atrybut
 - android:columnCount, 189
 - android:entries, 250
 - android:exported, 548
 - android:gravity, 181, 182, 183, 184, 193
 - android:icon, 377
 - android:id, 45
 - android:inputType, 206
 - android:label, 372
 - android:layout_above, 171
 - android:layout_alignBottom, 171
 - android:layout_alignLeft, 171
 - android:layout_alignParentBottom, 169
 - android:layout_alignParentLeft, 169
 - android:layout_alignParentRight, 169
 - android:layout_alignParentTop, 169
 - android:layout_alignRight, 171, 183
 - android:layout_alignTop, 171
 - android:layout_below, 171
 - android:layout_centerHorizontal, 169
 - android:layout_centerInParent, 169
 - android:layout_centerVertical, 169
 - android:layout_column, 193
 - android:layout_columnSpan, 194
 - android:layout_gravity, 193
 - android:layout_height, 45, 166, 177
 - android:layout_marginBottom, 172
 - android:layout_marginLeft, 172
 - android:layout_marginRight, 172
 - android:layout_marginTop, 172
 - android:layout_row, 193
 - android:layout_toLeftOf, 171
 - android:layout_toRightOf, 171
 - android:layout_weight, 179, 180
 - android:layout_width, 45, 166, 177
 - android:name, 548
 - android:onClick, 244, 347, 359
 - android:orderInCategory, 377
 - android:parentActivityName, 392
 - android:rowCount, 189
 - android:text, 45, 50
 - android:theme, 372
 - android:title, 377
 - choiceMode, 406
 - divider, 406
 - dividerHeight, 406
 - layout_gravity, 406
 - layout_height, 406
 - layout_width, 406
 - padding, 167
 - parent, 373
 - showAsAction, 378
 - xmlns:android, 166
 - AVD, 23, 30
 - konfiguracja, 26
 - tworzenie, 24, 25

B

back stack, *Patrz:* stos cofnąć
 baza danych, 438, 601
 aktualizacja, 449, 455, 457, 459, 460, 461, 462, 465, 510
 JDBC, *Patrz:* JDBC
 kopiowanie, 658
 modyfikacja struktury, 465
 nazwa, 466
 numer wersji, 456, 459
 odtworzenie, 658
 rekord
 aktualizacja, 448, 449, 465
 usuwanie, 450, 465
 schemat zmiana, 456
 SQLite, *Patrz:* SQLite
 tworzenie, 443, 445, 447
 wydajność, 525, 526, 539
 zamykanie, 489, 499
 biblioteka
 AWT, 4
 Support Libraries, 368, 369, 370, 406, 602
 Swing, 4
 wsparcia, *Patrz:* biblioteka Support Libraries
 błąd, 345

C

card view, *Patrz:* widok karty
 cień, 598, 599

D

Dalvik, 650
 dostawca
 akcji, 386, 387
 treści, 665
 dp, *Patrz:* piksel niezależny od gęstości
 DrawerLayout, 399, 405
 dziennik, 543, 545
 logcat, 657

E

Eclipse, 7
 edytor
 kodu, 18, 32
 projektu, 18, 32, 43, 44, 49
 tekstów, 7
 efekt 3D, 598
 ekran, 270, 306, 307, 308
 gęstość, 215, 309
 kategorii, 366
 orientacja, 309, *Patrz też:*
 urządzenie obracanie

 poziomu głównego, 366
 proporcje, 309
 szczegółów/edycji, 366
 wielkość, 309, 359
 wygląd, 12
 element
 Button, 48
 ContentValues, 449
 FragmentManager, 301
 FrameLayout, 300
 HorizontalScrollView, 219
 ImageView, 215, 216
 ListView, *Patrz:* widok listy
 meta-data, 392
 RelativeLayout, 47
 ScrollView, 219
 service, 548
 style, 373
 TextView, 48, 193, 205
 definiowanie, 205
 emulator, 4, 23, 660
 zrzut stanu, 661
 event listener, *Patrz:* obiekt nasłuchujący

F

filtr intencji, *Patrz:* intencja filtr
 fragment, 271, 272, 273, 287, 359
 cykl życia, 283, 327
 dodawanie
 do projektu, 276, 401
 do układu aktywności, 279
 interfejs, 295, 296, 298
 listy, 288, 289, 297
 zagnieżdżanie, 326, 337, 341, 342
 znacznik, *Patrz:* znacznik
 FrameLayout, *Patrz:* układ ramki

G

Google Play płatności, 5
 GPS, 579, 582
 gradle, 7
 grafika 9-patch, 672
 graphical user interface, *Patrz:* GUI
 GridLayout, *Patrz:* układ siatki
 Groovy, 7
 GUI, 2, 164, 201, 545, 552

I

ikona
 ic_action_new_event, 379
 zestaw systemowy, 379
 Instrumentation Testing, *Patrz:* test
 oprzyrządowania
 IntelliJ IDEA, 5

intencja, 82, 83, 87, 88, 117, 386, 388, 545
 filtr, 101, 102, 113
 jawna, 113, 559
 niejawna, 98, 113
 oczekująca, 559, 560
 tworzenie, 96, 97
 uruchamianie przez powiadomienie, 559, 560
 wyznaczanie, 101
 intent filter, *Patrz:* intencja filtr
 intent resolution, *Patrz:* intencja
 wyznaczanie
 interfejs, 296, 298
 DrawerListener, 417
 Listener, 634, 636
 OnClickListener, 347, 349, 351, 359
 OnItemClickListener, 408
 programowania aplikacji, *Patrz:* API
 użytkownika
 graficzny, *Patrz:* GUI
 wygląd, 12
 iPhone Simulator, *Patrz:* symulator
 iPhone'a

J

Java, 2, 4, 5, 16
 instalacja, 6
 klasa własna, *Patrz:* klasa własna
 Javy
 Java Virtual Machine, *Patrz:* JVM
 język
 Groovy, *Patrz:* Groovy
 Java, *Patrz:* Java
 SQL, *Patrz:* SQL
 XML, *Patrz:* XML
 journal file, *Patrz:* plik magazynu
 JVM, 650, 652

K

catalog
 aplikacji, 16
 app, 17
 build, 17
 com.hfad., 17
 debug, 17
 drawable, 215, 237
 drawable-hdpi, 308, 379
 generated, 17
 java, 17
 layout, 17, 308, 320
 layout-large, 308, 320
 main, 17
 opcje, 309
 platform-tools, 654
 r, 17

Skorowidz

- katalog
 - res, 17, 132
 - source, 17
 - src, 17
 - tworzenie, 215
 - values, 17, 373
 - kategoria, 101, 113, 229, 230, 232, 247, 308
 - ekran, *Patrz:* ekran kategorii
 - klasa
 - abstrakcyjna, 135, 497
 - ActionBar, 412
 - ActionBarActivity, 370, 385
 - ActionBarDrawerToggle, 417, 420, 421
 - Activity, 57, 135
 - Android.util.Log, 546
 - ApplicationTestCase, 673
 - AppWidgetProvider, 671
 - AppWidgetProviderInfo, 671
 - AsyncTask, 530, 535, 539
 - Bundle, 137, 138, 141
 - Button, 244
 - CaptionedImagesAdapter, 632, 633, 635
 - Context, 135
 - ContextThemeWrapper, 135
 - ContextWrapper, 135
 - Cursor, 440
 - CursorAdapter, *Patrz:* adapter
 - CursorAdapter
 - CursorLoader, 669
 - DrawerLayout, 369
 - EditText, 91
 - Handler, 124
 - Handler, 552
 - implementacja, 66
 - IntentService, 544, 545, 571
 - java.util.Map, 141
 - ListActivity, *Patrz:* aktywność listy
 - ListFragment, 404
 - List View, *Patrz:* widok listy
 - Location, 576
 - OnItemClickListener, 631
 - R, 59
 - ReceiveMessageActivity, 92
 - RecyclerView.Adapter, 605
 - Service, 544, 571, 575
 - SQLiteDatabase, 440, 448, 449, 450
 - SQLiteOpenHelper, 440, 443, 444
 - testowa, 673
 - testowanie, 66
 - Theme.Holo, 368
 - View, 45, 202, 328
 - ViewGroup, 202
 - WebView, 666
 - własna Javy, 66
 - Workout, 275
 - klucz RSA, 105
 - kod
 - aktywności, *Patrz:* plik MainActivity.
 - java
 - bajtowy, 650
 - źródłowy, *Patrz:* plik źródłowy
 - komponent
 - Spinner, 49, 53, 54, 60
 - TextView, 45, 60
 - komunikat, 220
 - na ekranie, 552
 - odbiorca, 670
 - w dzienniku systemowym, 545, 551
 - w górnej części ekranu, *Patrz:* powiadomienie
 - konsola, 28
 - konstruktor bezargumentowy, 278
 - kursor, 475, 486, 488, 496
 - pobieranie wartości, 489
 - tworzenie, 497, 509
 - zamykanie, 489, 499
- ## L
- layout, *Patrz:* układ
 - LinearLayout, *Patrz:* układ liniowy
 - lista, 229, 288, 295, 599
 - opcji, 399
 - rozwijana, 214, 251
 - widok, *Patrz:* widok listy
- ## Ł
- łańcuch znaków, *Patrz:* plik wartości łańcuchowych
- ## M
- manifest, 17, 80, 81, 136, 249, 287, 372, 548, 582
 - mapy, 668, 669
 - Material Design, 597, 599, 600
 - Maven, 7
 - menu, 377
 - metoda
 - add, 301
 - addToBackStack, 301
 - bindService, 588
 - changeCursor, 523
 - chroniona, 327
 - closeDrawer, 413
 - commit, 301
 - createChooser, 108, 109, 111, 112, 113
 - cursor.close, 489
 - db.close, 489
 - delete, 450, 510
 - distanceTo, 576
 - doInBackground, 530, 532, 533, 535
 - execSQL, 466
 - execute, 536
 - findFragmentById, 287
 - findFragmentByTag, 427
 - findViewById, 59, 60, 135, 203, 285, 287, 328
 - getApplicationContext, 555
 - getChildFragmentManager, 341
 - getFragmentManager, 339, 340
 - getHeight, 203
 - getId, 203
 - getInt, 489
 - getIntent, 88, 260
 - getReadableDatabase, 444, 484, 485
 - getString, 489
 - getStringExtra, 92
 - getSystemService, 561
 - getView, 284, 285, 287
 - getWidth, 203
 - getWritableDatabase, 444, 484, 485
 - insert, 448, 510
 - invalidateOptionsMenu, 418
 - isClickable, 203
 - isFocused, 203
 - Log.d, 546
 - Log.e, 546
 - Log.i, 546
 - Log.v, 546
 - Log.w, 546
 - Log.wtf, 546
 - moveToFirst, 488
 - moveToLast, 488
 - moveToNext, 488
 - moveToPrevious, 488
 - onActivityCreated, 283, 327
 - onAttach, 283, 327
 - onBackStackChanged, 426
 - onBind, 572, 573
 - onClick, 56, 58, 59, 61, 67, 350
 - onCreate, 57, 129, 133, 134, 135, 137, 141, 142, 152, 161, 283, 327, 424, 575
 - onCreateOptionsMenu, 135, 380
 - onCreateView, 278, 304, 327, 342, 356, 357
 - onDestroy, 133, 134, 135, 142, 152, 161, 283, 327, 575
 - onDestroyView, 283, 327
 - onDetach, 283, 327
 - onDowngrade, 444, 455, 456, 459, 461
 - onHandleIntent, 544, 545, 552
 - onItemClick, 241, 408, 631
 - onListItemClick, 257
 - onLocationChanged, 576

onOpen, 444
 onOptionsItemSelected, 381
 onPause, 135, 150, 151, 152, 153, 154, 156, 161, 283, 327
 onPostExecute, 530, 534, 535
 onPreExecute, 530, 531, 535
 onPrepareOptionsMenu, 418
 onProgressUpdate, 533, 535
 onRestart, 135, 142, 143, 149, 151, 161, 327, 523
 onResume, 135, 150, 151, 152, 153, 154, 161, 283, 327, 327
 onSaveInstanceState, 135, 142, 304, 355, 424
 onSendMessage, 93
 onServiceConnect, 587
 onServiceDisconnect, 587
 onStart, 135, 142, 143, 144, 149, 152, 154, 161, 283, 285, 327
 przesłanianie, 144
 onStartCommand, 553, 575
 onStop, 135, 142, 143, 144, 154, 156, 161, 283, 327, 588
 przesłanianie, 144
 onUpgrade, 444, 455, 456, 459, 460
 post, 124, 552
 postDelayed, 124
 publiczna, 58, 327
 publishProgress, 533
 put, 448
 putExtra, 88
 query, 477, 478, 479, 484, 486, 487
 remove, 301
 replace, 301
 requestFocus, 203
 requestLocationUpdates, 579
 setActionBarTitle, 426
 setContentView, 57, 133, 135, 278, 281, 356
 setListAdapter, 291
 setOnClickListener, 351
 setShareIntent, 388
 setTitle, 412
 setVisibility, 203, 418
 startActivity, 82, 99, 117, 135
 toString, 251
 unbindService, 588
 update, 449, 450, 510

motyw, 371, 373, 385
 domyślny, 374
 Theme.Holo, 368, 371, 385
 Theme.Material, 368, 374

N

navigation drawer, *Patrz:* szuflada nawigacyjna

NinePatch, *Patrz:* grafika 9-patch
 null, 138

O

obiekt
 Binder, 572, 573, 592
 LocationListener, 576, 577, 578, 579
 LocationManager, 579
 nasłuchujący, 203, 241, 242, 244, 248, 296, 408
 DrawerListener, 416
 OnBackStackChangedListener, 426
 OnClickListener, 351
 ServiceConnection, 572, 573, 587
 TaskStackBuilder, 559, 560
 obraz, 215, 216
 obsługa zdarzeń, 203
 obszar powiadomień, 543

P

pasek
 akcji, 368, 375, 385
 dodawanie elementów, 376, 379
 dostawca akcji, *Patrz:* dostawca akcji
 modyfikowanie elementów, 418
 przycisk W górę, 391
 tytuł, 412, 416
 przewijania, 219
 piksel niezależny od gęstości, 166
 platforma SDK, *Patrz:* SDK platforma plik
 .apk, 27, 30, 651
 .class, 5, 650
 .dex, 650, 652
 .jar, 5, 650
 activity_main.xml, 17, 19, 32, 33
 AndroidManifest.xml, *Patrz:* manifest APK, 651
 bazy danych, 439
 classes.dex, 650, 651
 graficzny, 237
 graficzny ikon, 16
 klasowy, 650, 652
 konfiguracyjny, 16
 magazynu, 439
 MainActivity.java, 17, 21, 31, 57
 R.java, 17, 59, 65
 strings.xml, 17, 34, 35, 36, 40, 50, 53, 65, 77, 120
 styles.xml, 373
 stylów, 16
 wartości łańcuchowych, 16, 17, 34, 36
 zasobów
 menu, 377

stylów, 372, 373
 źródłowy, 16, 18, 19
 pole
 tekstowe, 204, 206
 wyboru, 210
 polecenie
 adb devices, 654
 adb logcat, 657
 adb pull, 657
 adb push, 657
 adb shell, 656
 powiadomienie, 556
 ikona, 556, 561
 szuflada, 556, 561
 tworzenie, 558
 wysyłanie za pomocą usługi systemowej, 561
 powłoka systemowa, 656
 proces, 116, 129
 procesor, 270
 wirtualny Dalvik, *Patrz:* Dalvik projekt, 75, 235
 przeglądarka WWW, 666
 przełącznik, 209
 przycisk, 45, 55, 56, 58, 122, 204, 549
 ActionBarDrawerToggle, 417, 420, 421
 definiowanie, 207
 etykieta, 120
 opcji, 212, 238, 239
 przełącznika, 208
 W górę, 391, 392, 393, 420
 Wstecz, 391, 424, 426
 z obrazem, 216, 217
 z tekstem, 216

R

RelativeLayout, *Patrz:* układ względny
 Robotium, 673

S

SDK, 5, 23
 platforma, 5
 wersja minimalna, 10, 16, 25
 service, *Patrz:* usługa
 spinner, *Patrz:* komponent Spinner
 SQL, 447
 klauzula
 GROUP BY, 482
 HAVING, 482
 SELECT, 492
 w zapytaniach, 481
 SQLite, 439, 446
 pomocnik, 440, 442, 452
 tworzenie, 444

Skorowidz

stoper, 118, 121, 123, 142, 326
stos cofnięć, 299, 300, 301, 340, 424
styl, 373

Material Design, 597, 598
symulator iPhone'a, 660
szuflada nawigacyjna, 369, 398, 399, 507
inicjalizacja listy, 407
otwieranie, 416, 417, 420
tworzenie, 400
zamykanie, 413, 416, 417, 420

Ś

środowisko
programistyczne IDE, 4, 5
uruchomieniowe, *Patrz:* ART

T

tablet, 42, 270, 271, 288, 306, 307, 314, 359
klawiatura, 206
wirtualny, 30
tablica, 250, 251
telefon, 23, 24, 42, 270, 271, 288, 306,
307, 315, 316, 359
klawiatura, 206
numer, 97, 206
połączenie, 117, 142
test oprzyrządowania, 673
testowanie, 673
tost, 220, 543, 556
transakcja, 299, 301, 340
zagnieżdżanie, 341
typ Runnable, 124

U

układ, 2, 12, 31, 32, 40, 48, 164, 201, 204
activity_create_message.xml, 75
DrawerLayout, *Patrz:* DrawerLayout
liniowy, 165, 174, 175, 183, 186
definiowanie, 174
identyfikator widoku, 175
modyfikowanie, 176
marginesy, 172, 173
menedżer, 615
GridLayoutManager, 615, 616
LinearLayoutManager, 615, 616
StaggeredGridLayoutManager, 615

nazwa, 14
ramki, 300, 342
siatki, 165, 189, 197, 369
definiowanie, 190, 191
szerokość, 166
waga, 178, 180
wysokość, 166
względny, 47, 164, 165, 166, 173, 204
zależny od wielkości ekranu, 308
zmiana, 51, 76
urządzenie
ekran, *Patrz:* ekran
konfiguracja, 132, 136
obracanie, 130, 131, 132, 152, 303,
304, 354, 355, 424, 425, 522
procesor, *Patrz:* procesor
tablet, *Patrz:* tablet
telefon, *Patrz:* telefon
ustawienia lokalne, 132
wirtualne z Androidem, *Patrz:* AVD
usługa, 542
deklarowanie, 548
IntentService, 553
lokalizacyjna, 576
nazwa, 548
powiadomień, 556, 557
powiązana, 542, 569, 572
systemowa, 561
uruchamianie, 550
uruchomiona, 542, 544

W

wartość
fill_parent, 166
łańcuchowa, *Patrz:* plik wartości
łańcuchowych
match_parent, 166
null, *Patrz:* null
void, 58
wątek, 526
główny, 526, 533, 552
w tle, 526, 529, 533, 539, 552
wyświetlania, 526
wiadomość, 94
widok, 203
aktualizowanie, 300
CardView, *Patrz:* widok karty

grupa, 202
hierarchia, 204
karty, 599, 609
dodawanie danych, 610
tworzenie, 603
listy, 231, 241, 242, 244, 251, 289,
400, 494, 495
źródło danych, 250
obrazów, 215
przewijany, 219
RecyclerView, 599, 600, 605, 606,
613
menedżer układu, *Patrz:* układ
menedżer
obsługa kliknięć, 631, 632
rozmieszczenie
względem innych widoków, 170
względem układu nadrzędnego, 168
tekstowy, 205
tło, *Patrz:* grafika 9-patch
zastępowanie, 300
widżet
aplikacji, 671
CardView, 369
RecyclerView, 369
wiersz poleceń, 7, 654, 656
wirtualna maszyna Javy, *Patrz:* JVM
właściwość, *Patrz:* atrybut
wyjątek, 546
SQLiteException, 485, 492

X

XML, 2, 16, 44, 48, 164

Z

zadanie asynchroniczne, 536, 537, 539
zapytanie, 476
budowniczy, 477
w tle, 669
warunki, 476, 478, 479
wyniki
ograniczanie, 476, 478, 479
porządkowanie, 480
zdarzenie, 203, 631, 670
generowane przez widok, 632
znacznik, 427, 546

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

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

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

GRUPA WYDAWNICZA

 **Helion SA**



Rusz głową!

Android

Programowanie aplikacji

Android jest niezwykłym systemem. Dynamikę jego rozwoju i ekspansji można określić jednym słowem: *oszałamiająca!* Programiści, producenci urządzeń mobilnych, a przede wszystkim użytkownicy wysoko sobie cenią jego zalety, takie jak modułowa architektura, wysoka elastyczność czy otwarty charakter systemu. Liczbę urządzeń pracujących pod kontrolą Androida podaje się w miliardach, a przewiduje się, że będzie ich o wiele, wiele więcej. Umiejętność efektywnego pisania świetnych, atrakcyjnych aplikacji dla Androida staje się niezawodną receptą na sukces.

Książka, którą trzymasz w rękach, to podręcznik niezwykły, gdyż uwzględnia specyfikę funkcjonowania ludzkiego mózgu i sposób, w jaki najszybciej się uczy. Dzięki nowatorskiemu podejściu autorów nauka pisania aplikacji nie jest nudna: niepostrzeżenie będziesz nabierać coraz większej wprawy. Już w trakcie zapoznawania się z podstawowymi koncepcjami rozpoczniesz pracę w IDE Android Studio. Dowiesz się, jak zaprojektować strukturę aplikacji i jak zbudować idealny interfejs. Będziesz swobodnie posługiwać się aktywnościami, intencjami, usługami. Poznasz interfejs Material Design firmy Google, dowiesz się, jak wykorzystywać bazy danych SQLite. A to dopiero początek...

W tej książce znajdziesz między innymi:

- omówienie zasad tworzenia aplikacji interaktywnych — odpowiadających na działania użytkownika
- przedstawienie świetnego narzędzia dla programistów — środowiska Android Studio
- opis koncepcji istotnych dla Androida: aktywności i ich cyklu życia, intencji, układów i fragmentów, usług i wielu innych
- wskazówki dotyczące tworzenia wyrafinowanych systemów nawigowania w aplikacji, korzystania z szuflad nawigacyjnych oraz z pasków akcji
- przedstawienie interfejsu Material Design
- omówienie ART — środowiska uruchomieniowego Androida i ADB, programu narzędziowego umożliwiającego rozwiązanie niektórych problemów podczas programowania i testowania aplikacji

Rusz głową i zacznij pisać świetne aplikacje dla Androida!

Dawn Griffiths — z wykształcenia matematyk, zawodowo pracuje jako programistka, ma już ponad 20-letnie doświadczenie. Napisała kilka książek z serii *Rusz głową!*

David Griffiths — programuje od 12. roku życia. Pracował jako instruktor zwinnych metod programowania. Zna już ponad 10 języków. Podobnie jak jego żona, Dawn Griffiths, jest matematykiem i autorem kilku książek z serii *Rusz głową!*

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

Helion

43408 numer katalogowy
księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

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

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

ISBN 978-83-283-2063-5



9 788328 320635

Informatyka w najlepszym wydaniu

cena: 99,00 zł