

Android Studio — prawdziwe atelier sztuki programowania!



Android Studio

Wygodne i efektywne tworzenie aplikacji

Adam Gerber • Clifton Craig

Apress® **Helion** 



Tytuł oryginału: Learn Android Studio: Build Android Apps Quickly and Effectively

Tłumaczenie: Rafał Jońca

ISBN: 978-83-283-2009-3

Original edition copyright © 2015 by Adam Gerber and Clifton Craig.
All rights reserved.

Polish edition copyright © 2016 by HELION SA.
All rights reserved.

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

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

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

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

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/answyg.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/answyg>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

O autorach	13
O redaktorze technicznym	15
Podziękowania	17
Wprowadzenie	19
Rozdział 1. Wprowadzenie do Android Studio	23
Instalacja JDK w systemie Windows	23
Pobieranie JDK w systemie Windows	23
Uruchomienie instalacji w systemie Windows	24
Konfiguracja zmiennych środowiskowych	24
Instalacja JDK w systemie Mac OS	29
Pobieranie JDK w systemie Mac OS	29
Uruchomienie instalacji w systemie Mac OS	30
Konfiguracja JDK w systemie Mac OS	32
Instalacja Android Studio	32
Tworzenie pierwszego projektu — HelloWorld	35
Użycie Android Virtual Device Manager	37
Uruchomienie aplikacji HelloWorld w AVD	39
Uruchamianie aplikacji HelloWorld na urządzeniu	41
Podsumowanie	43
Rozdział 2. Poruszanie się po Android Studio	45
Edytor	46
Zakładki edytora	46
Margines	47
Pasek znaczników	48
Przyciski narzędzi	48
Wygląd domyślny	48
Okna narzędzi nawigacyjnych	49
Okno narzędzia Project	49
Okno narzędzia Structure	49

Okno narzędzia Favorites	50
Okno narzędzia TODO	51
Pasek głównego menu	51
Pasek narzędziowy	52
Pasek nawigacyjny	52
Pasek statusu	52
Typowe operacje	53
Zaznaczanie tekstu	53
Polecenia cofania i przywracania	53
Znajdowanie ostatnio otwieranych plików	54
Przemieszczanie się po ostatnich operacjach nawigacyjnych	54
Kopiowanie, wycinanie i wklejanie	54
Menu kontekstowe	55
Uzyskiwanie pomocy	55
Poruszanie się po IDE przy użyciu klawiatury	56
Polecenie Select In	56
Polecenie Class	56
Polecenie File	57
Polecenie Line	57
Polecenie Related File	57
Polecenie Last Edit Location	57
Polecenie Type Hierarchy	57
Polecenie Declaration	57
Znajdowanie i zastępowanie tekstu	58
Polecenie Find	58
Polecenie Find in Path	58
Polecenie Replace	58
Polecenie Replace in Path	58
Podsumowanie	59
Rozdział 3. Programowanie w Android Studio	61
Zawijanie kodu	61
Uzupełnianie kodu	63
Komentowanie kodu	66
Wykorzystanie generowania kodu	66
Konstruktor	66
Metody ustawiające i pobierające	68
Przesłanie metod	68
Metoda toString()	70
Delegowanie metod	70
Wstawianie szablonów działających na żywo	70
Przenoszenie kodu	73
Odpowiedni styl kodu	74
Polecenie Auto-Indent Lines	74
Polecenie Rearrange Code	75
Polecenie Reformat Code	76
Polecenie Surround With	76
Podsumowanie	77

Rozdział 4. Refaktoryzacja kodu	79
Operacja Rename	80
Operacja Change Signature	81
Operacja Type Migration	82
Operacja Move	82
Operacja Copy	84
Operacja Safe Delete	84
Operacje wydobywania	84
Operacja Extract Variable	85
Operacja Extract Constant	85
Operacja Extract Field	86
Operacja Extract Parameter	86
Operacja Extract Method	87
Zaawansowana refaktoryzacja	89
Operacje Push Members Down i Pull Members Up	89
Operacja Replace Inheritance with Delegation	89
Operacja Encapsulate Fields	91
Operacja Wrap Method Return Value	92
Operacja Replace Constructor with Factory Method	92
Operacja Convert Anonymous to Inner	93
Podsumowanie	94
Rozdział 5. Laboratorium — przypomnienia, część I	95
Tworzenie nowego projektu	96
Inicjalizacja repozytorium Git	97
Tworzenie interfejsu użytkownika	101
Korzystanie z edytora wizualnego	101
Edycja pliku XML zawierającego układ graficzny	102
Dodanie uprawnień wizualnych	106
Dodanie elementów do ListView	108
Utworzenie menu paska akcji	109
Zapewnienie trwałości przypomnień	110
Model danych	111
Niskopoziomowe API SQLite	112
Podsumowanie	117
Rozdział 6. Laboratorium — przypomnienia, część II	119
Dodawanie i usuwanie przypomnień	119
Reagowanie na interakcję użytkownika	121
Własne okna dialogowe	122
Utworzenie menu kontekstowego z wyborem wielu elementów	123
Obsługa wcześniejszych wersji API	125
Dodanie kontekstowego trybu akcji	126
Implementacja dodawania, edycji i usuwania przypomnień	128
Planowanie własnego okna dialogowego	129
Od planów do kodu	130

	Utworzenie własnego okna dialogowego	132
	Dodanie własnej ikony	134
	Podsumowanie	135
Rozdział 7.	Wprowadzenie do narzędzia Git	139
	Instalacja Git	139
	Ignorowanie plików	141
	Dodawanie plików	142
	Klonowanie przykładowej aplikacji Przypomnienia	142
	Klonowanie i rozwidlanie	143
	Przeglądanie historii zmian repozytorium	145
	Odgałęzienia	146
	Programowanie w osobnej gałęzi	146
	Tworzenie rewizji i rozgałęzień w Git	152
	Gdzie jest polecenie cofnięcia?	153
	Łączenie gałęzi	157
	Zmiana historii poleceniem Reset	159
	Operacja Git Rebase	163
	Odłączenie od gałęzi	163
	Referencje względne	166
	Rozwiązywanie konfliktów w trakcie operacji zmiany bazy	167
	Zdalne repozytoria Git	172
	Podsumowanie	173
Rozdział 8.	Projektowanie układu komponentów aplikacji	175
	Klasa Activity	175
	Klasy View i ViewGroup	176
	Okno podglądu	177
	Wysokość i szerokość	178
	Tryb projektowania	180
	Komponent FrameLayout	181
	Komponent LinearLayout	184
	Komponent RelativeLayout	185
	Zagnieżdżanie komponentów	188
	Komponent ListView	192
	Wskazówki dotyczące projektowania układu graficznego aplikacji	198
	Obsługa różnych wielkości ekranu	199
	Łączymy wszystko razem	199
	Fragmety	208
	Podsumowanie	215
Rozdział 9.	Laboratorium — waluty, część I	217
	Specyfikacja aplikacji	217
	Inicjalizacja repozytorium Git	221
	Modyfikacja układu aktywności MainActivity	223
	Definiowanie kolorów	226
	Zastosowanie kolorów w układzie graficznym	227
	Tworzenie i stosowanie stylów	228

Utworzenie klasy JSONParser	231
Utworzenie aktywności ekranu powitalnego	232
Pobieranie kodów walut w formacie JSON	234
Uruchamianie MainActivity	236
Podsumowanie	238
Rozdział 10. Laboratorium — waluty, część II	239
Definicja składowych klasy MainActivity	239
Rozpakowanie kodów walut z paczki	240
Utworzenie menu opcji	241
Implementacja zachowania dla elementów menu	242
Utworzenie układu spinner_closed	243
Powiązanie mCurrencies z listami	244
Przeniesienie zachowania list rozwijanych do MainActivity	244
Utworzenie menedżera ustawień	246
Znalezienie pozycji na podstawie kodu	247
Wydobycie kodu waluty z całego tekstu	247
Implementacja współdzielonych ustawień	249
Obsługa naciśnięcia przycisku	250
Przechowywanie klucza API	251
Pobranie klucza API	252
Klasa CurrencyConverterTask	252
Metoda onPreExecute()	256
Metoda doInBackground()	256
Metoda onPostExecute()	256
Selektor przycisku	257
Ikona aplikacji i jej tytuł	258
Podsumowanie	259
Rozdział 11. Testowanie i analiza	261
Utworzenie nowego testu instrumentacyjnego	262
Definicja metod setUp() i tearDown()	262
Definicja wywołania zwrotnego w MainActivity	264
Definicja metod testujących	265
Uruchomienie testu instrumentacji	266
Naprawa błędu	267
Użycie narzędzia Monkey	268
Korzystanie z narzędzi analitycznych	269
Inspekcja kodu	269
Analiza zależności	270
Analiza stosu wywołań	270
Podsumowanie	273
Rozdział 12. Debugowanie	275
Dziennik zdarzeń	275
Użycie narzędzia logcat	276
Zapis danych w dzienniku	278

Polowanie na błędy!	278
Korzystanie z debugera interaktywnego	282
Obliczanie wyrażenia	284
Wykorzystanie stosu wywołań	286
Elementy narzędzia interaktywnego debugera	289
Przeglądarka punktów wstrzymania	289
Warunkowe punkty wstrzymania	292
Podsumowanie	295
Rozdział 13. Narzędzie Gradle	297
Składnia Gradle	298
System budowania aplikacji IntelliJ	299
Podstawowe zagadnienia systemu budowania Gradle	299
Struktura skryptu Gradle dla Androida	300
Zależności projektu	301
Laboratorium — projekt prognozy pogody	302
Zależności od bibliotek systemu Android	307
Zależność od bibliotek Javy	312
Biblioteki zewnętrzne	320
Otwieranie starszych projektów	322
Podsumowanie	323
Rozdział 14. Dodatkowe narzędzia Android SDK	325
Android Device Monitor	325
Zakładka Threads	325
Zakładka Heap	326
Zakładka Allocation Tracker	328
Zakładka Network Statistics	329
Przeglądarka hierarchii	329
Integracja monitora Androida	332
Zakładka Memory	332
Narzędzie Method Trace	333
Narzędzie Allocation Tracker	334
Wykonywanie zrzutów ekranu	335
Edytor nawigacji	336
Projektowanie interfejsu użytkownika	337
Pierwsze kroki w edytorze nawigacji	337
Łączenie aktywności	337
Terminal	340
Uzyskanie listy urządzeń	340
Instalacja APK	340
Pobranie pliku	340
Wysłanie pliku	340
Przekazywanie portów	340
Narzędzia chmury Google	341
Utworzenie części klienckiej	342
Utworzenie modułu serwerowego	343

Połączenie wszystkich elementów	343
Wdrożenie na serwerach Google App Engine	349
Podsumowanie	352
Rozdział 15. Laboratorium — Android Wear	353
Konfiguracja środowiska	353
Instalacja sterowników urządzenia	353
Konfiguracja narzędzi SDK	354
Konfiguracja urządzenia wirtualnego Wear	354
Konfiguracja sprzętu Android Wear	355
Tworzenie projektu MegaDroid	357
Optymalizacja związana z technologią ekranu	358
Tworzenie usługi WatchFace	360
Inicjalizacja zasobów i stylów	362
Zarządzanie aktualizacjami zegara	362
Rysowanie tarczy i wskazówek	366
Podsumowanie	369
Rozdział 16. Dostosowywanie Android Studio do własnych potrzeb	371
Style kodu	371
Wygląd, kolory i czcionki	374
Skróty klawiaturowe	376
Makra	376
Szablony kodu i plików	377
Menu i paski narzędziowe	377
Moduły dodatkowe	380
Podsumowanie	381
Skorowidz	383



Laboratorium — Android Wear

Android Wear, jedna z najnowszych technologii firmy Google, daje szansę na bardziej osobisty i intymny kontakt z urządzeniem. Obecnie nie ma jeszcze zbyt wielu urządzeń obsługujących Android Wear, ale ich lista stale rośnie. W zasadzie dostępne urządzenia to głównie zegarki, ale gdy technologia bardziej się rozwinie, w zasadzie urządzeniem obsługującym Android Wear będzie mogło być cokolwiek, na przykład naszyjnik lub ubranie. Obecnie zegarki z systemem Android są produkowane przez trzy firmy znane z produkcji urządzeń z systemem Android: Samsung, Motorola i Sony. W tym rozdziale wykonamy aplikację, która może działać zarówno przewodowo, jak i bezprzewodowo.

-
- **Uwaga** Zachęcamy do sklonowania tego projektu za pomocą narzędzia Git, abyś mógł przyrzeć się całemu przykładowi. Z drugiej strony w poszczególnych podrozdziałach będziemy krok po kroku tworzyć ten projekt od podstaw. Jeśli narzędzie Git nie jest zainstalowane na Twoim komputerze, zajrzyj do rozdziału 7. Otwórz sesję Git-bash w systemie Windows (lub terminal w systemie Mac albo Linux) i przejdź do folderu C:\androidBook\reference (utwórz folder, jeśli nie istnieje). W systemach Linux i Mac folder może znajdować się w innym miejscu. Wykonaj następujące polecenie: `git clone https://bitbucket.org/gluwer/megadroid.git MegaDroid`.
-

Konfiguracja środowiska

Przed rozpoczęciem prac nad aplikacją musimy poświęcić nieco czasu na konfigurację środowiska. Choć można wykonać całą pracę, posiłkując się tylko emulatorem, zawsze najlepiej mieć rzeczywiste urządzenie Wear. Upewnij się, że Twoje urządzenie korzysta z najnowszej wersji systemu operacyjnego. Pobierz i zainstaluj wszystkie wymagane sterowniki. Podłącz urządzenie do komputera i poszukaj go na liście urządzeń okna narzędzia *Android Monitor*.

Instalacja sterowników urządzenia

W systemie Windows konieczna może okazać się instalacja sterowników niezbędna do uzyskania komunikacji za pomocą USB. Pamiętaj o tym, aby instalować sterownik tylko wtedy, gdy urządzenie nie jest rozpoznane. Punkt ten możesz pominąć, jeśli planujesz przesyłanie aplikacji za pomocą połączenia Bluetooth. Przy pierwszej próbie podłączenia urządzenia system Windows automatycznie rozpocznie wyszukiwanie sterowników.

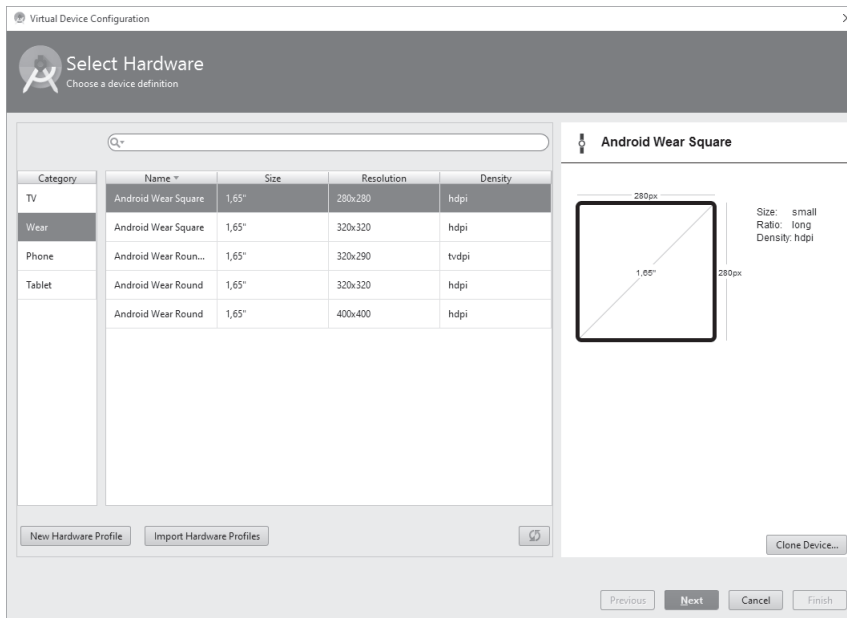
Otwórz *Menedżer urządzeń* i znajdź urządzenie na liście wewnątrz grupy *Inne urządzenia*. Kliknij je prawym klawiszem myszy i wybierz polecenie *Aktualizuj oprogramowanie sterownika*. W oknie, które się pojawi, wybierz drugą opcję (*Przełączaj mój komputer w poszukiwaniu oprogramowania sterownika*), a następnie opcję *Pozwól mi wybrać z listy sterowników urządzeń moim komputerze*. Z listy wybierz *Android Device*. Jeśli na następnej liście nie znajdziesz sterownika firmy produkującej zegarek, wybierz ogólny sterownik *ADB Interface*, który również zadziała prawidłowo.

Konfiguracja narzędzi SDK

Przed rozpoczęciem prac programistycznych pobierz i zainstaluj SDK w wersji przynajmniej 5.0.1 lub nowszej i uaktualnij narzędzia SDK do wersji 24.0.2 lub nowszej. Obsługa Android Wear pojawiła się w wersji 4.4W.2 i narzędziach SDK 23.0, ale przykłady użyte w dalszej części wymagają nowszego SDK. Warto również zainstalować przykłady z SDK oraz biblioteki Google znajdujące się w pakiecie *Extras*.

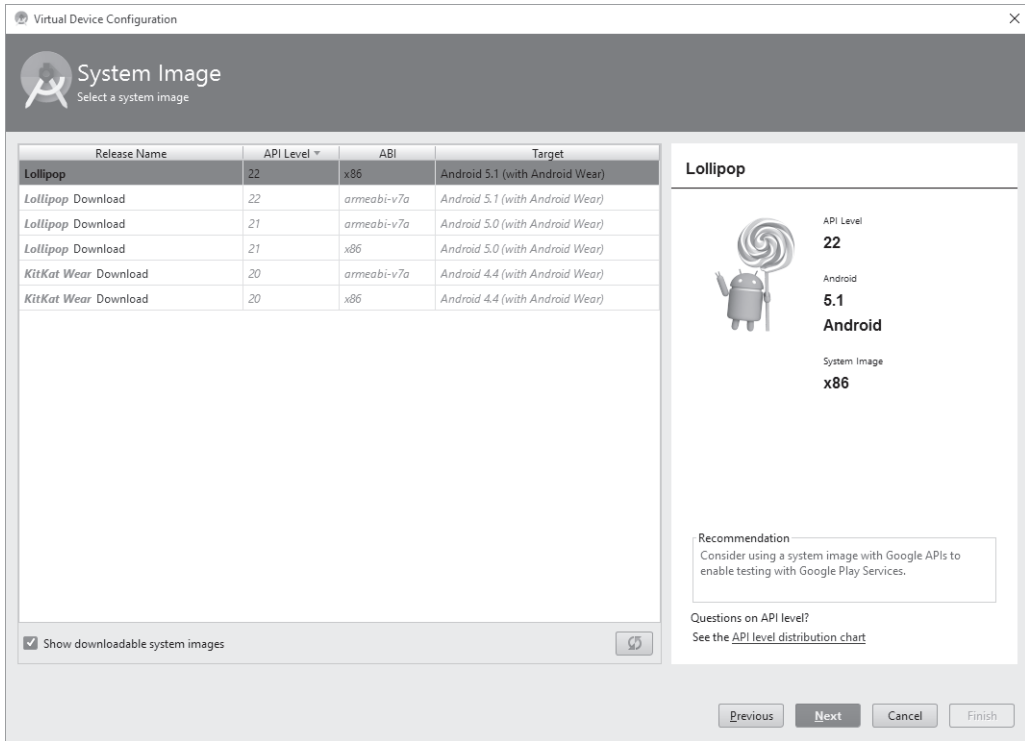
Konfiguracja urządzenia wirtualnego Wear

Uruchom narzędzie *AVD Manager* z poziomu paska narzędziowego lub poleceniem *Tools/Android/AVD Manager*. Kliknij przycisk *Create Virtual Device*. Z listy kategorii po lewej wybierz *Wear*. Jako profile sprzętowe po prawej stronie pojawiają się profile typu *Android Wear Square* i *Android Wear Round* (patrz rysunek 15.1). Wybierz API 5.0.1 lub nowsze, ponieważ prezentowane przykłady wymagają funkcji dostępnych w wersji 5.0.1. W zależności od szybkości komputera można wybrać obraz systemu w wersji x86. Tego rodzaju obrazy działają szybciej, ponieważ nie muszą emulować procesora. Z drugiej strony ich dostępność uzależniona jest od systemu HAXM (*Hardware Accelerated Execution Manager*) firmy Intel. HAXM również instaluje się z poziomu narzędzia *AVD Manager*. Do działania wymaga on dostępności technologii wirtualizacji (VT) firmy Intel, która nie jest dostępna na każdym sprzęcie. Kliknij przycisk *Next*, aby pozostawić wartości domyślne, i przejdź do następnego ekranu kreatora. By uzyskać optymalną szybkość działania, włącz opcję *Use Host CPU*.



Rysunek 15.1. Wybór kategorii Wear

Wybierz najnowszy dostępny poziom API (obecnie jest to Lollipop). Następnie kliknij przycisk *Next* (patrz rysunek 15.2).



Rysunek 15.2. Wybór obrazu systemu w wersji Lollipop

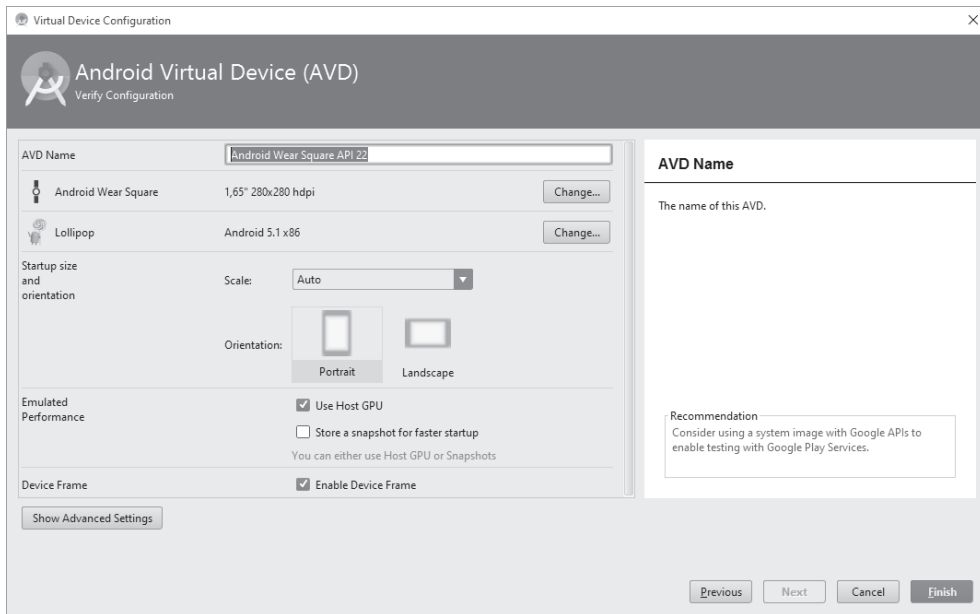
Nadaj AVD nazwę Android Wear Square API 22 na ekranie przedstawionym na rysunku 15.3.

Kliknij przycisk *Finish*, aby utworzyć AVD. Następnie kliknij strzałkę w dół na końcu wpisu dotyczącego Wear AVD i wybierz polecenie *Duplicate* (patrz rysunek 15.4). Jeśli pierwszy utworzony AVD był typu *Square*, zmień go na wersję *Round*, i odwrotnie. Warto utworzyć dwa obrazy, aby móc testować aplikację w obu wariantach.

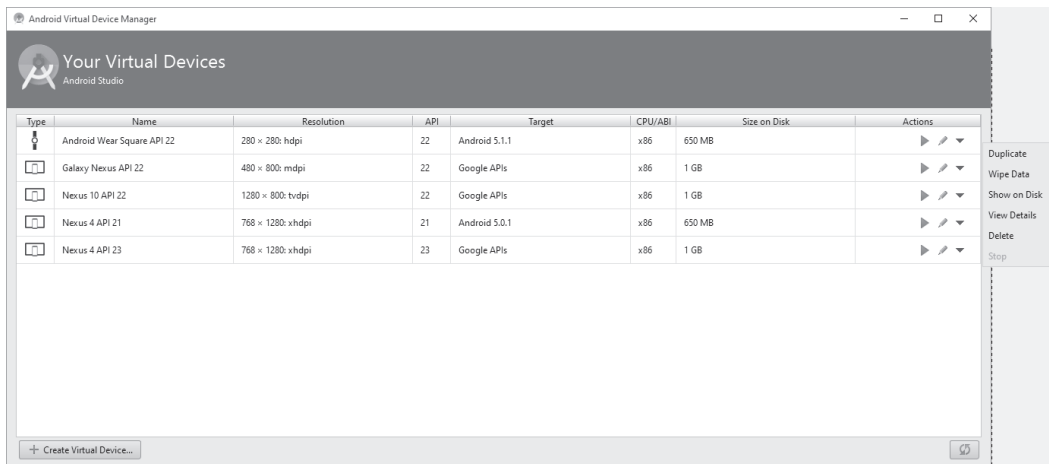
Konfiguracja sprzętu Android Wear

Osoby posiadające urządzenie Android Wear muszą je odpowiednio skonfigurować, aby móc go używać do testowania aplikacji. Aplikację przekazuje się na urządzenia Wear za pomocą telefonu komórkowego lub tabletu, więc one również będą potrzebne. Na komórce z systemem Android zainstaluj aplikację Android Wear ze sklepu Google Play. Uruchom aplikację i sparuj ją z urządzeniem Wear (na przykład zegarkiem).

Istnieją dwa sposoby wdrażania aplikacji na zegarku — przewodowy i bezprzewodowy (Bluetooth). Rozwiązanie przewodowe jest prostsze, ale rozwiązanie Bluetooth jest wygodniejsze, szczególnie jeśli portów USB w komputerze jest mało i nie chcesz zajmować się sterownikami. Brak portów USB jest szczególnie uciążliwy, gdy jednocześnie testuje się aplikację na smartfonie, tablecie i zegarku.



Rysunek 15.3. Nadaj obrazowi nazwę Android Wear Square API 22



Rysunek 15.4. Zduplicuj AVD Android Wear Square API 22

Włączenie trybu programisty

Jeśli nigdy wcześniej nie był włączony tryb programisty (lub urządzenie jest całkowicie nowe), wykonaj poniższe kroki, aby móc włączyć na urządzeniu tryb pozwalający na ciągłą pracę urządzenia, testowanie poprzez Bluetooth, sprawdzanie układu aplikacji itp.

1. Otwórz aplikację *ustawienia* na urządzeniu, naciskając i przytrzymując przez dwie sekundy przycisk na boku.
2. Przejdź na sam dół listy i wybierz ostatni element (informacje o urządzeniu).

3. Wewnątrz nowego ekranu kliknij siedmiokrotnie element *numer kompilacji*. Po powrocie do głównej listy ustawień dostępny będzie nowy element z opcjami programisty.
4. Wejść do nowego elementu i włączyć opcję debugowania.

Włączenie debugowania Bluetooth

Na ekranie opcji programisty włącz debugowanie Bluetooth, aby móc korzystać z bezprzewodowego testowania aplikacji. Następnie otwórz narzędzie *Terminal* i wpisz następujące polecenia ADB:

```
adb forward tcp:4444 localabstract:/adb-hub
adb connect localhost:4444
```

Sprawdź status aplikacji Android Wear na urządzeniu mobilnym. Powinny pojawić się wartości:

```
Host: podłączone
Cel: podłączone
```

Od tego momentu aplikacja i zegarek są gotowe do instalacji aplikacji.

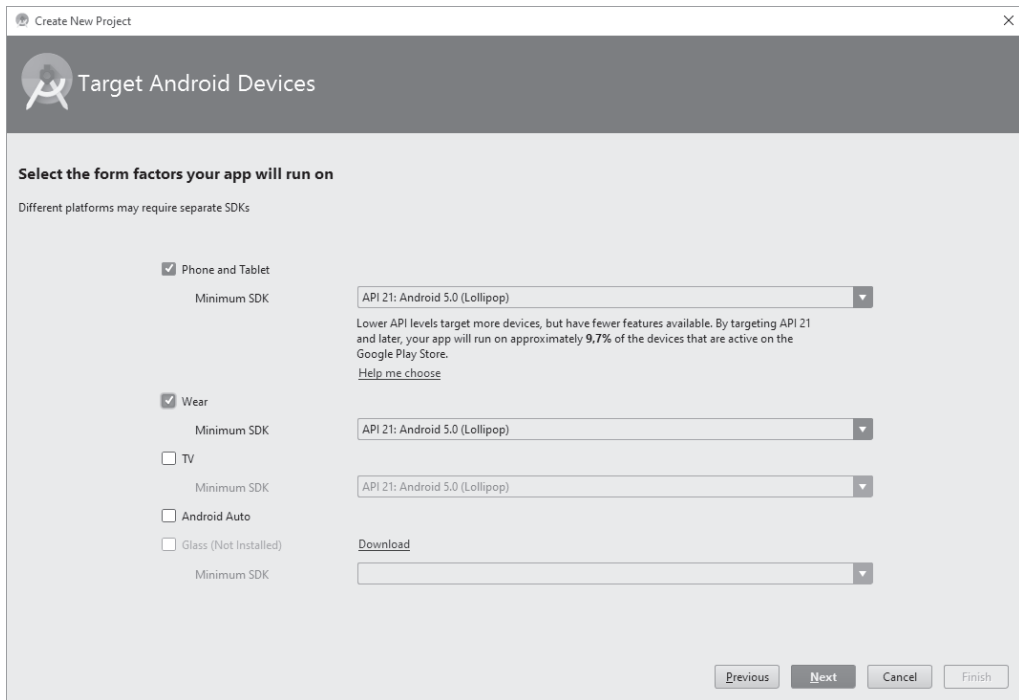
Tworzenie projektu MegaDroid

W tym podrozdziale wykonamy aplikację tarczy zegarka z fikcyjną postacią o nazwie MegaDroid. MegaDroid to połączenie postaci z dwóch gier wideo z lat 80. (nie będziemy podawali ich nazw). Tło zegarka będzie stanowiła postać wojownika, który walczy z wrogami za pomocą dwóch mieczy. Aplikacja może stanowić dodatek do prawdziwej gry. Rysunek 15.5 przedstawia finalny wynik. Przykład można wykorzystać jako podstawę do napisania własnej aplikacji zegara z własną marką. Tworzenie własnych aplikacji zegara to nowość wprowadzona w wersji Lollipop. Tego rodzaju rozwiązanie działa jako rzeczywista tarcza zegarka (urządzenia), więc otwiera przed programistą nowe możliwości. Aplikacja może wyświetlać informacje z różnych źródeł, na przykład internetu, GPS, kalendarza powiązanego urządzenia itp. Ponieważ aplikacja zegara działa cały czas, może posłużyć jako rozszerzenie aplikacji. W przykładzie omówimy tylko najważniejsze aspekty związane z interfejsem użytkownika i jego aktualizacją wraz z upływem czasu.



Rysunek 15.5. Finalna wersja zegarka z postacią MegaDroid

Użyj kreatora nowego projektu opisanego w rozdziale 1., aby utworzyć projekt Android Wear. Na drugim ekranie kreatora wybierz opcję *Wear* i SDK w wersji przynajmniej 5.0 (patrz rysunek 15.6). Na pozostałych ekranach pozostaw wartości domyślne. Na ekranie wyboru szablonów wybierz *Blank Activity* dla aplikacji mobilnej i *Blank Wear Activity* dla komponentu Wear. Zakończ konfigurację projektu przyciskiem *Finish*. Zbuduj i uruchom projekt, aby sprawdzić go na rzeczywistym urządzeniu lub emulatorze.



Rysunek 15.6. Włączenie opcji *Wear* w kreatorze nowego projektu

Wykonanie jak najlepszego projektu układu aplikacji to jedno z najważniejszych zadań stojących przed twórcami aplikacji typu Wear. Aby produkt stał się optymalny, na początku większość czasu należy poświęcić na dopracowywanie układu elementów, ich kolorów, wyrazistości, czytelności itp. Każde tło zegarka jest unikatowe, więc dostosowanie się do wymogów urządzenia bywa trudne. Oczywiście znacznie łatwiej wykonać proste tło z cyfrowym zegarem (bo są to tylko cyfry) niż z wersją imitującą zegar analogowy. Dokumentacja dostępna na stronach internetowych projektu Android może onieśmielić wiele osób niezaznajomionych ze sztuką projektowania. Na stronie zamieszczono sugestię, by zastanowić się nad wykonaniem zarówno wersji kwadratowej, jak i okrągłej, a także nad dodatkowymi danymi, elementami interfejsu użytkownika i obsługą różnych rozdzielczości. Można także zastanowić się nad ekranem konfiguracji. Ponieważ prezentowany przykład ma przedstawić jedynie podstawy, wiele z wymienionych aspektów zostało pominiętych.

Optymalizacja związana z technologią ekranu

System wykonawczy Wear uruchamia aplikację w dwóch trybach wyświetlania: w trybie tła i w trybie interaktywnym. Zegarek przełącza się między tymi trybami w zależności od tego, czy aplikacja jest przeglądana, czy używana. Tryb tła ma za zadanie oszczędzać baterię. Aplikacja powinna taki tryb wykryć i na przykład przyciemnić ekran. Ponieważ aktualizacje w tym trybie są wysyłane raz na minutę, warto zmniejszyć częstotliwość odświeżania ekranu. Tworzona aplikacja usunie w tym trybie wygładzanie jednej ze wskazówek i zmieni częstotliwość odświeżania ekranu z jednej sekundy na jedną minutę.

Niektóre urządzenia w trybie tła obsługują jedynie niższą rozdzielczość, a samo urządzenie stosuje ograniczoną paletę kolorów. To redukuje zużycie energii i zapobiega wypaleniu ekranu. Wejście w tryb ograniczonych kolorów można wykryć i ograniczyć wykorzystane barwy tylko do czerni, bieli, zielonego, niebieskiego, czerwonego, karmazynowego, żółtego i cyjanowego. Warto czasem zastosować zamiast całego

obrazu jedynie jego otoczkę. W trybie ograniczonych kolorów tło powinno pozostawać prawie czarne. Piksele w kolorze innym niż czerń nie powinny zajmować więcej niż 10% ekranu, a intensywne kolory więcej niż 5%. W tym trybie warto również wyłączyć korzystanie z wygładzania krawędzi. Wygładzanie krawędzi to technika, po której zastosowaniu krawędzie rysunku wydają się mniej poszarpane (porównaj rysunku 15.7 i 15.8).



Rysunek 15.7. Obraz bez wygładzania



Rysunek 15.8. Obraz z wygładzaniem

Aplikacja w trybie działania w tle wykorzysta wersję grafiki w odcieniach szarości (patrz rysunek 15.9). Skopiuj wszystkie obrazy z folderu gotowego projektu do folderu nowego projektu. W systemie Windows przeciągnij prawym klawiszem myszy folder *res* z folderu

`C:\androidBook\reference\MegaDroid\wear\src\main` do folderu `C:\androidBook\MegaDroid\wear\src\main` i wybierz z menu kontekstowego polecenie *Kopiuj tutaj*. W systemie Mac OS lub Linux z poziomu konsoli wykonaj polecenie:

```
cp -R ~/androidBook/reference/MegaDroid/wear/src/main/res
~/androidBook/MegaDroid/wear/src/main/
```



Rysunek 15.9. Grafika w odcieniach szarości

Tworzenie usługi WatchFace

Usługa tarczy zegarka odpowiada za utworzenie WatchFaceService.Engine, czyli głównego mechanizmu obsługującego tarczę zegarka. Klasa WatchFaceService.Engine odpowiada na wywołania zwrotne systemu, aktualizuje czas i rysuje obraz tła (tarczy zegarka). Utwórz nową klasę MegaDroidWatchFaceService dziedziczącą po klasie CanvasWatchFaceService. Umieść w klasie kod z listingu 15.1.

Listing 15.1. Klasa *MegaDroidWatchFaceService*

```
public class MegaDroidWatchFaceService extends CanvasWatchFaceService {
    private static final String TAG = "MegaDroidWatchSvc";
    @Override
    public Engine onCreateEngine() {
        // utworzenie i zwrócenie klasy Engine dla obrazu tła
        return new MegaDroidEngine(this);
    }

    /* implementacja metod wywołań zwrotnych */
    private class MegaDroidEngine extends CanvasWatchFaceService.Engine {
        private final Service service;
        public MegaDroidEngine(Service service) {
            this.service = service;
        }

        /**
         * inicjalizacja tła zegarka
         */
        @Override
        public void onCreate(SurfaceHolder holder) {
            super.onCreate(holder);
        }

        /**
         * wywoływana w momencie zmiany właściwości,
         * co pozwala włączyć tryb szarości
         */
        @Override
        public void onPropertiesChanged(Bundle properties) {
            super.onPropertiesChanged(properties);
        }

        /**
         * wywoływana przez system wykonawczy co minutę
         */
        @Override
        public void onTimeTick() {
            super.onTimeTick();
        }

        /**
         * wywoływana w trybie wejścia do trybu tła i wyjścia z tego trybu
         */
        @Override
        public void onAmbientModeChanged(boolean inAmbientMode) {
            super.onAmbientModeChanged(inAmbientMode);
        }
    }
}
```

```

    }

    @Override
    public void onDraw(Canvas canvas, Rect bounds) {
        // rysowanie tła
    }

    /**
     * wywoływana, gdy tło staje się widoczne lub niewidoczne
     */
    @Override
    public void onVisibilityChanged(boolean visible) {
        super.onVisibilityChanged(visible);
    }
}
}

```

Rejestracja usługi

Dodaj poniższy fragment do pliku manifestu aplikacji Android tuż przed znacznikiem zamykającym application:

```

<service
    android:name=".MegaDroidWatchFaceService"
    android:label="@string/mega_droid_service_name"
    android:allowEmbedded="true"
    android:taskAffinity=""
    android:permission="android.permission.BIND_WALLPAPER" >
    <meta-data
        android:name="android.service.wallpaper"
        android:resource="@xml/watch_face" />
    <meta-data
        android:name="com.google.android.wearable.watchface.preview"
        android:resource="@drawable/preview_analog" />
    <meta-data
        android:name="com.google.android.wearable.watchface.preview_circular"
        android:resource="@drawable/preview_analog_circular" />
    <intent-filter>
        <action android:name="android.service.wallpaper.WallpaperService" />
        <category android:name=
            "com.google.android.wearable.watchface.category.WATCH_FACE" />
    </intent-filter>
</service>

```

Po umieszczeniu kodu w manifeście pojawi się kilka błędów. Użyj klawisza *F2* i kombinacji *Alt+Enter*, aby wyświetlić sugestie i rozwiązać problemy. Pierwsza sugestia dotyczy wygenerowania nazwy dla nowej usługi, która pojawi się na liście zawierającej nazwy i rodzaje tła. Następna sugestia dotyczy kodu XML deskryptora określającego tapetę. Utwórz plik *xml/watch_face.xml* i umieść w nim poniższy kod:

```

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

```

Dwa następne błędy nie mogą być łatwo naprawione na poziomie sugestii Intellij. Stanowią referencję do zasobów rysowania, które odpowiadają za podgląd tarczy na ekranie wyboru tła. Można niezbędne pliki utworzyć w dowolnym edytorze graficznym. Alternatywą jest wykonanie zrzutu ekranu działającej aplikacji, ale przypomina to stawianie wszystkiego na głowie. Jeśli nie chcesz zajmować się edytorem grafiki rastrowej,

pobierz dwa dowolne obrazy, nadaj im odpowiednie nazwy i wstaw do projektu. Gdy aplikacja będzie gotowa, wykonaj zrzuty ekranu i wstaw je w miejsce wcześniejszych, tymczasowych obrazów.

Inicjalizacja zasobów i stylów

W metodzie onCreate() dodaj następujący kod, który zainicjalizuje style i zasoby rysowania:

```
public void onCreate(SurfaceHolder holder) {
    super.onCreate(holder);
    setWatchFaceStyle(new WatchFaceStyle.Builder(service)
        .setCardPeekMode(WatchFaceStyle.PEEK_MODE_SHORT)
        .setStatusBarGravity(Gravity.RIGHT | Gravity.TOP)
        .setHotwordIndicatorGravity(Gravity.LEFT | Gravity.TOP)
        .setBackgroundVisibility(WatchFaceStyle.BACKGROUND_VISIBILITY_INTERRUPTIVE)
        .setShowSystemUiTime(false)
        .build());

    Resources resources = service.getResources();
    Drawable backgroundDrawable = resources.getDrawable(R.drawable.bg);
    this.backgroundBitmap = ((BitmapDrawable) backgroundDrawable).getBitmap();
    this.character = ((BitmapDrawable) resources.getDrawable(
        R.drawable.character_standing)).getBitmap();
    this.logo = ((BitmapDrawable) resources.getDrawable(
        R.drawable.megadroid_logo)).getBitmap();
    this.minuteHand = ((BitmapDrawable) resources.getDrawable(
        R.drawable.minute_hand)).getBitmap();
    this.hourHand = ((BitmapDrawable) resources.getDrawable(
        R.drawable.hour_hand)).getBitmap();

    this.secondPaint = new Paint();
    secondPaint.setARGB(255, 255, 0, 0);
    secondPaint.setStrokeWidth(2.f);
    secondPaint.setAntiAlias(true);
    secondPaint.setStrokeCap(Paint.Cap.ROUND);

    this.time = new Time();
}
```

Zarządzanie aktualizacjami zegara

Dodaj dwa poniższe pola do głównej klasy:

```
private static final long INTERACTIVE_UPDATE_RATE_MS =
    TimeUnit.SECONDS.toMillis(1);
private static final int MSG_UPDATE_TIME = 0;
```

Utworzymy procedurę aktualizacji, która wymusi aktualizację zegara na podstawie zdefiniowanej wcześniej stałej INTERACTIVE_UPDATE_RATE_MS:

```
/** aktualizacja czasu co sekundę w trybie interaktywnym */
final Handler mUpdateTimeHandler = new Handler() {
    @Override
    public void handleMessage(Message message) {
        switch (message.what) {
            case MSG_UPDATE_TIME:
```

```

        if (Log.isLoggable(TAG, Log.VERBOSE)) {
            Log.v(TAG, "aktualizacja czasu");
        }
        invalidate();
        if (shouldTimerBeRunning()) {
            long timeMs = System.currentTimeMillis();
            long delayMs = INTERACTIVE_UPDATE_RATE_MS
                - (timeMs % INTERACTIVE_UPDATE_RATE_MS);
            mUpdateTimeHandler.sendMessageDelayed(MSG_UPDATE_TIME, delayMs);
        }
        break;
    }
}
};

private boolean shouldTimerBeRunning() {
    return isVisible() && !isInAmbientMode();
}

```

Kod procedury aktualizacyjnej ponownie umieści się na liście zadań do wykonania (z odpowiednim opóźnieniem) po pierwszym uruchomieniu. Kod niczego nie rysuje — wywołuje tylko unieważnienie zawartości ekranu, co wymusza niejawnie wywołanie metody `onDraw()`. Przed następnym uruchomieniem z zadaniem opóźnieniem sprawdza jeszcze, czy zegarek nie znajduje się w trybie tła i czy samo tło jest w ogóle widoczne. Dodaj metodę `onDestroy()`, aby zwolnić zasoby, gdy usługa będzie czyszczona przez system:

```

@Override
public void onDestroy() {
    mUpdateTimeHandler.removeMessages(MSG_UPDATE_TIME);
    super.onDestroy();
}

```

Zaimplementuj metodę `onPropertiesChanged()`, aby śledzić włączenie trybu `lowBitAmbient`. Dodaj składową typu `boolean`, by zapamiętać aktualny tryb działania. W ten sposób będziemy decydować, czy ograniczyć szybkość wyświetlania. Umieść w metodzie poniższy kod:

```

public void onPropertiesChanged(Bundle properties) {
    super.onPropertiesChanged(properties);
    this.lowBitAmbient = properties.getBoolean(PROPERTY_LOW_BIT_AMBIENT, false);
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "onPropertiesChanged: w tle z szarością = " + lowBitAmbient);
    }
}

```

Składowa `lowBitAmbient` powinna mieć postać:

```

private boolean lowBitAmbient;

```

Dodaj w metodzie `onTimeTick()` wywołanie metody `invalidate()`, co spowoduje wymuszanie ponownego rysowania ekranu:

```

@Override
public void onTimeTick() {
    super.onTimeTick();
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "onTimeTick: tryb tła = " + isInAmbientMode());
    }
    invalidate();
}

```

Następnie zaimplementuj wywołanie zwrótnie `onAmbientModeChanged()`. W zależności od trybu użyjemy grafiki w kolorach szarości lub grafiki kolorowej i dla jednej ze wskazówek wyłączymy wygładzanie (patrz wcześniejszy opis).

```
public void onAmbientModeChanged(boolean inAmbientMode) {
    super.onAmbientModeChanged(inAmbientMode);
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "onAmbientModeChanged: " + inAmbientMode);
    }
    if(inAmbientMode) {
        character = ((BitmapDrawable) service.getResources().getDrawable(
            R.drawable.character_standing_greyscale)).getBitmap();
        logo = ((BitmapDrawable) service.getResources().getDrawable(
            R.drawable.megadroid_logo_bw)).getBitmap();
        hourHand = ((BitmapDrawable) service.getResources().getDrawable(
            R.drawable.hour_hand_bw)).getBitmap();
        minuteHand = ((BitmapDrawable) service.getResources()
            .getDrawable(R.drawable.minute_hand_bw)).getBitmap();
    } else {
        character = ((BitmapDrawable) service.getResources()
            .getDrawable(R.drawable.character_standing)).getBitmap();
        logo = ((BitmapDrawable) service.getResources()
            .getDrawable(R.drawable.megadroid_logo)).getBitmap();
        hourHand = ((BitmapDrawable) service.getResources()
            .getDrawable(R.drawable.hour_hand)).getBitmap();
        minuteHand = ((BitmapDrawable) service.getResources()
            .getDrawable(R.drawable.minute_hand)).getBitmap();
    }
    if (lowBitAmbient) {
        boolean antiAlias = !inAmbientMode;
        secondPaint.setAntiAlias(antiAlias);
    }
    invalidate();

    // To, czy kod ponawiający odświeżanie powinien działać, zależy od
    // trybu działania (i tego, czy tło jest widoczne), więc
    // kod odpowiedzialny za ponowienie odświeżenia wywołujemy
    // za każdym razem.
    updateTimer();
}
```

Kod wywołuje metodę `updateTimer()`, którą dopiero musimy zdefiniować. Metoda ta wysyła do `mUpdateTimeHandler` pusty komunikat aktualizacji. Z drugiej strony komunikat ten wysyłamy tylko w sytuacji, gdy tło jest widoczne i znajduje się w trybie interaktywnym. Utwórz metodę o następującej treści:

```
private void updateTimer() {
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "updateTimer");
    }
    mUpdateTimeHandler.removeMessages(MSG_UPDATE_TIME);
    if (shouldTimerBeRunning()) {
        mUpdateTimeHandler.sendMessage(MSG_UPDATE_TIME);
    }
}
```

Użyj kodu z listingu 15.2 do zdefiniowania odbiornika komunikatu zmiany strefy czasowej. Metody rejestracji i usuwania rejestracji pozwolą odpowiednio zareagować w sytuacji zmiany strefy czasowej.

Listing 15.2. Klasa `BroadcastReceiver` dotycząca zmian strefy czasowej

```

final BroadcastReceiver mTimeZoneReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        time.clear(intent.getStringExtra("time-zone"));
        time.setToNow();
    }
};
boolean mRegisteredTimeZoneReceiver = false;

private void registerReceiver() {
    if (mRegisteredTimeZoneReceiver) {
        return;
    }
    mRegisteredTimeZoneReceiver = true;
    IntentFilter filter = new IntentFilter(Intent.ACTION_TIMEZONE_CHANGED);
    service.registerReceiver(mTimeZoneReceiver, filter);
}

private void unregisterReceiver() {
    if (!mRegisteredTimeZoneReceiver) {
        return;
    }
    mRegisteredTimeZoneReceiver = false;
    service.unregisterReceiver(mTimeZoneReceiver);
}

```

Kod spowoduje uaktualnienie czasu za każdym razem, gdy otrzyma komunikat zmiany strefy czasowej. Metoda `registerReceiver()` rejestruje `IntentFilter` powiązany z akcją `ACTION_TIMEZONE_CHANGED`. Klasa `IntentFilter` to programowy sposób na powiązanie aktywności lub klasy `BroadcastReceiver` z konkretną akcją.

Na końcu zmodyfikuj metodę wywołania zwrótnego `onVisibilityChanged()` odpowiedzialną za uruchamianie lub włączanie aktualizacji tła:

```

@Override
public void onVisibilityChanged(boolean visible) {
    super.onVisibilityChanged(visible);
    if (Log.isLoggable(TAG, Log.DEBUG)) {
        Log.d(TAG, "onVisibilityChanged: " + visible);
    }

    if (visible) {
        registerReceiver();
        // Aktualizacja strefy czasowej w sytuacji, gdy nie byliśmy widoczni.
        time.clear(TimeZone.getDefault().getID());
        time.setToNow();
    } else {
        unregisterReceiver();
    }

    // To, czy kod ponawiający odświeżanie powinien działać, zależy od
    // trybu działania (i tego, czy tło jest widoczne), więc
    // kod odpowiedzialny za ponowienie odświeżenia wywołujemy
    // za każdym razem.
    updateTimer();
}

```

Rysowanie tarczy i wskazówek

Rysowanie tarczy zegara i wskazówek jako tła urządzenia to najbardziej złożony fragment całej aplikacji. Wywołanie metody `invalidate()` powoduje wywołanie metody `onDraw()`. Wewnątrz metody na podstawie grafik utworzymy finalną wersję tarczy i wskazówek zegara, która będzie aktualizowana po upływie godzin, minut i sekund. Zmień kod metody `onDraw()` na przedstawiony na listingu 15.3.

Listing 15.3. Pełna implementacja metody `onDraw()`

```
public void onDraw(Canvas canvas, Rect bounds) {
    time.setToNow();

    int width = bounds.width();
    int height = bounds.height();

    // Rysowanie tła po odpowiednim przeskalowaniu.
    if (backgroundScaledBitmap == null
        || backgroundScaledBitmap.getWidth() != width
        || backgroundScaledBitmap.getHeight() != height) {
        backgroundScaledBitmap = Bitmap.createScaledBitmap(backgroundBitmap,
            width, height, true /*filtr*/);
    }
    canvas.drawBitmap(backgroundScaledBitmap, 0, 0, null);

    canvas.drawBitmap(character, (width- character.getWidth())/2,
        (height- character.getHeight())/2+ 20, null);
    canvas.drawBitmap(logo, (width- logo.getWidth())/2, (logo.getHeight()*2), null);

    float secRot = time.second / 30f * (float) Math.PI;
    int minutes = time.minute;
    float minRot = minutes / 30f * (float) Math.PI;
    float hrRot = ((time.hour + (minutes / 60f)) / 6f ) * (float) Math.PI;

    // Znajdowanie środka. Ignorowanie ramki, aby w zegarku z ekranem kołowym
    // uzyskać rzeczywisty środek, a nie środek logiczny nieuwzględniający marginesów.
    float centerX = width / 2f;
    float centerY = height / 2f;

    Matrix matrix = new Matrix();
    int minuteHandX = ((width - minuteHand.getWidth()) / 2) - (minuteHand.getWidth() / 2);
    int minuteHandY = (height - minuteHand.getHeight()) / 2;
    matrix.setTranslate(minuteHandX-20, minuteHandY);
    float degrees = minRot * (float) (180.0 / Math.PI);
    matrix.postRotate(degrees+90, centerX, centerY);
    canvas.drawBitmap(minuteHand, matrix, null);

    matrix = new Matrix();
    int rightArmX = ((width - hourHand.getWidth()) / 2) + (hourHand.getWidth() / 2);
    int rightArmY = (height - hourHand.getHeight()) / 2;
    matrix.setTranslate(rightArmX + 20, rightArmY);
    degrees = hrRot * (float) (180.0 / Math.PI);
    matrix.postRotate(degrees-90, centerX, centerY);
    canvas.drawBitmap(hourHand, matrix, null);

    float secLength = centerX - 20;
```



```

if (!isInAmbientMode()) {
    float secX = (float) Math.sin(secRot) * secLength;
    float secY = (float) -Math.cos(secRot) * secLength;
    canvas.drawLine(centerX, centerY, centerX + secX, centerY + secY, secondPaint);
}
}

```

Analiza kodu fragment po fragmencie pozwoli lepiej zrozumieć jego działanie. Zaczniemy od pobrania wysokości i szerokości obiektu `bounds` przekazanego do metody:

```

int width = bounds.width();
int height = bounds.height();

```

Następnie sprawdzamy przeskalowaną wersję tła i rysujemy ją na ekranie. Tło musi zostać przeskalowane na określoną szerokość i wysokość, ale operację wystarczy wykonać tylko raz, bo wymiary ekranu nie zmieniają się.

// Rysowanie tła po odpowiednim przeskalowaniu.

```

if (backgroundScaledBitmap == null
    || backgroundScaledBitmap.getWidth() != width
    || backgroundScaledBitmap.getHeight() != height) {
    backgroundScaledBitmap = Bitmap.createScaledBitmap(backgroundBitmap,
        width, height, true /*filtr*/);
}
canvas.drawBitmap(backgroundScaledBitmap, 0, 0, null);

```

Najpierw rysujemy tło z postacią, a następnie logo. Odrobina matematyki pomoże w wyśrodkowaniu postaci, ale i przesunięciu jej o 20 pikseli w pionie od pełnego środka. Środkowanie wymaga uzyskania różnicy między szerokością przestrzeni rysowania i szerokością postaci i podzielenia całej wartości przez dwa. Dla wysokości wykonujemy te same obliczenia oraz dodajemy 20 pikseli przesunięcia:

```

canvas.drawBitmap(character, (width - character.getWidth()) / 2,
    (height - character.getHeight()) / 2 + 20, null);
canvas.drawBitmap(logo, (width - logo.getWidth()) / 2,
    (logo.getHeight() * 2), null);

```

Następna część to ćwiczenie z geometrii, bo musimy znaleźć kąt obrotu wskazówek. We wzorze dzielimy sekundy i minuty przez 30π . Dla godzin wzór jest bardziej złożony, bo godzinę dzielimy na 6 części po uwzględnieniu przesunięcia minutowego. Uzyskany w ten sposób wynik mnożymy przez π . Wskazówkę godzinową przesuwamy lekko na podstawie liczby minut w godzinie (więc wartość dzielimy przez 60). Oczywiście takie przesunięcie jest opcjonalne i wskazówka godzinowa mogłaby się poruszać skokowo.

```

float secRot = time.second / 30f * (float) Math.PI;
int minutes = time.minute;
float minRot = minutes / 30f * (float) Math.PI;
float hrRot = ((time.hour + (minutes / 60f)) / 6f) * (float) Math.PI;

```

Następnie znajdujemy środek ekranu, aby właśnie od niego rozpocząć rysowanie.

```

float centerX = width / 2f;
float centerY = height / 2f;

```

Na podstawie środka ekranu dokonujemy przesunięcia i obrotu wskazówki dotyczącej minut. Kąt obrotu wyliczamy, mnożąc minuty przez $180/\pi$. Ponieważ grafika pokazuje domyślnie godzinę 9, musimy dodać dodatkowe przesunięcie o 90 stopni:

```

Matrix matrix = new Matrix();
int minuteHandX = ((width - minuteHand.getWidth()) / 2) - (minuteHand.getWidth() / 2);
int minuteHandY = (height - minuteHand.getHeight()) / 2;

```

```
matrix.setTranslate(minuteHandX-20, minuteHandY);
float degrees = minRot * (float) (180.0 / Math.PI);
```

```
matrix.postRotate(degrees+90, centerX, centerY);
canvas.drawBitmap(minuteHand, matrix, null);
```

Wskaźówka godzinowa wymaga podobnych działań, ale dla grafiki wskazującej domyślnie na godzinę 3 musimy odjąć od kąta obrotu 90 stopni:

```
matrix = new Matrix();
int rightArmX = ((width - hourHand.getWidth()) / 2) + (hourHand.getWidth() / 2);
int rightArmY = (height - hourHand.getHeight()) / 2;
matrix.setTranslate(rightArmX + 20, rightArmY);
degrees = hrRot * (float) (180.0 / Math.PI);
```

```
matrix.postRotate(degrees-90, centerX, centerY);
canvas.drawBitmap(hourHand, matrix, null);
```

Wskaźówka sekund to jedynie czerwona linia od środka ekranu. Jej długość wyliczamy, odejmując 20 pikseli od środka. Logikę rysowania umieszczamy w bloku warunkowym, aby nie rysować sekundnika w trybie tła. Współrzędną X końca linii obliczamy na podstawie sinusa kąta i długości linii. Współrzędną Y wyliczamy jako minus cosinus kąta obrotu pomnożony przez długość linii. Wyliczone wartości wykorzystujemy w metodzie drawLine(), aby narysować linię od środka do wyliczonego punktu. Sposób rysowania jest określony przez utworzony wcześniej w metodzie onCreate() obiekt secondPaint.

```
float secLength = centerX - 20;
```

```
if (!isInAmbientMode()) {
    float secX = (float) Math.sin(secRot) * secLength;
    float secY = (float) -Math.cos(secRot) * secLength;
    canvas.drawLine(centerX, centerY, centerX + secX, centerY + secY, secondPaint);
}
```

Zbuduj i uruchom aplikację tła zegarka. Zainstaluj ją na urządzeniu. Rysunek 15.10 przedstawia ją w akcji na urządzeniu Galaxy Gear Live.



Rysunek 15.10. Aplikacja MegaDroid działająca na urządzeniu

Podsumowanie

W tym rozdziale nauczyłeś się tworzyć aplikację tła zegarka Android Wear. Zapoznałeś się ze sposobami wdrażania aplikacji za pomocą USB i Bluetooth. Dowiedziałeś się, w jaki sposób reagować na tryb działania w tle, aby ograniczyć zużycie energii. Omówiliśmy różne komponenty niezbędne do realizacji zadania tła zegarka, na przykład usługę tła, własną funkcję działającą z opóźnieniem i rysowanie na zadanym obszarze. Rozdział stanowi jedynie wprowadzenie do systemu Android Wear, ale możliwości tego systemu są spore. Aplikacja tła ma pełny dostęp do usług systemowych i może pobierać wpisy kalendarza, wyświetlać dane o poziomie baterii lub kontakty z książki adresowej.

Skorowidz

A

- adnotacja @Override, 254
- ADT, 22
- aktualizacja, 322
- aktualizacje zegara, 362
- aktywność
 - aplikacji, 175
 - MainActivity, 223, 236
 - SplashActivity, 234, 263
- analiza, 261
 - stosu wywołań, 270
 - zależności, 270
- Android
 - Device Monitor, 42, 325
 - SDK, 325
 - Studio, 21
 - Virtual Device Manager, 37
 - Wear, 353
- API, 125
- API SQLite, 112
- aplikacja
 - DebugMe, 278
 - MegaDroid, 368
 - Przypomnienia, 142
- automatyczne zarządzanie pamięcią, 21
- AVD, Android Virtual Device, 23

B

- baza danych SQLite, 95
- biblioteka, 301
 - kXML, 314

biblioteki

- Javy, 312
- systemu Android, 307
- zewnętrzne, 320

Bitcoin, 217

blok

- if-else, 249
- if-else if, 288
- switch-case, 292

bloki

- konfiguracyjne, 298
- zadań, 298

błędy, 267, 278, 316, 322

bufor cykliczny, 275

C

- chmura Google, 341
- CRUD, Create, Read, Update, Delete, 113
- czcionki, 374

D

- debuger interaktywny, 282
- debugowanie, 275
 - Bluetooth, 357
- definicja
 - interfejsu, 264
 - metod testujących, 265
 - wywołania zwrotnego, 264
- definiowanie
 - kolorów, 226
 - metod, 70

dodawanie

- elementów do ListView, 108
- klasy nadrzędnej, 69
- kontekstowego trybu akcji, 126
- marginesu, 189
- plików, 142
- przypomnień, 119, 128
- usprawnień wizualnych, 106
- własnej ikony, 134

domknięcie, closure, 298

dostosowywanie Android Studio, 371

dpi, dots per inch, 199

dziennik zdarzeń, 275

E

Eclipse, 22

edycja

- pliku XML, 102
- przypomnień, 128

edytor, 46

- nawigacji, 336–339
- wizualny, 102
- złączenia, 169

ekran, 358

- logowania, 339
- powitalny, 217, 232
- profilu, 205

element

- ListView, 103, 107
- receiver, 150

elementy

- menu, 109, 242
- okna podglądu, 178

emulacja, 37

F

folder

- .gradle, 300
- .idea, 300
- app, 300
- drawable, 199
- gradle, 300
- layout-large, 211

format

- ARR, 311
- JSON, 234

formatowanie wyników, 252

fragmenty, 208

G

generowanie

- koðu, 66
- metody, 70, 71
- pola stałej, 150

gęstość pikseli, 199

Google App Engine, 349

Gradle, 297

struktura skryptu, 300

H

harmonogram, 147

historia

- repozytorium, 160, 167, 171
- repozytorium Git, 155, 160
- rewizji, 159
- zmian repozytorium, 145

I

IDE, Integrated Development Environment, 45

ignorowanie plików, 141

ikona aplikacji, 258

implementacja

- współdzielonych ustawień, 249
- zachowania, 242

import, 62

informacje o aplikacji, 162

inicjalizacja repozytorium Git, 97, 221

inspekcja kodu, 269

instalacja

- Android Studio, 32
- APK, 340
- bibliotek klienckich, 346
- Git, 139
- JDK w systemie Mac OS, 29
- JDK w systemie Windows, 23
- sterowników urządzenia, 353

instrukcja

- Reset, 157
- Revert, 157

integracja monitora Androida, 332

IntelliJ IDEA, 371

interakcja użytkownika, 121

interfejs, 264

- OnItemSelectedListener, 245
- ukończonej aplikacji, 96
- użytkownika, 101, 337

J

Java, 21
 JDK, Java Development Kit, 23

K

klasa

- Activity, 175
- BroadcastReceiver, 149, 365
- BuddyDetailFragment, 209
- BuddyListFragment, 208
- CurrencyConverterTask, 252, 253
- DatabaseHelpers, 112
- Fragment, 209
- JSONParser, 231
- ListActivity, 193
- MainActivity, 193, 204, 239, 242
- MainActivityTest, 262
- MegaDroidWatchFaceService, 360
- NationalWeatherRequest, 316
- Patio, 91
- Person, 206
- PersonActivity, 207
- PersonAdapter, 196
- PrefsMgr, 247
- ProfileActivity, 203
- Reminder, 111
- RemindersActivity, 116
- RemindersDbAdapter, 112
- RemindersSimpleCursorAdapter, 115
- RemoteCloudBeanAsyncTask, 347
- Toast, 122
- View, 176
- ViewGroup, 176

klasy nadrzędne, 69

klonowanie

- aplikacji, 142
- repozytorium, 145

klucz API, 251, 252

kod XML, 105

kody walut, 234, 240

kolor, 226, 227, 374

komenda

- pull, 340
- push, 340

komentowanie kodu, 66

komponent

- FrameLayout, 181, 182
- LinearLayout, 184

- ListView, 192, 194
- RelativeLayout, 185, 186

komponenty

- aplikacji, 175
- układu aplikacji, 179

konfiguracja

- JDK, 32
- narzędzi SDK, 354
- sprzętu Android Wear, 355
- środowiska, 353
- urządzenia wirtualnego, 354
- zmiennych środowiskowych, 24

konflikt, 167

konsola deweloperska Google, 350

konstrukcja

- if-else, 249
- if-else if, 288
- switch-case, 292

konstruktor, 66

- ArrayAdapter, 244

kontener, 176

kopiowanie, 54

kreator

- modułu, 344
- nowego projektu, 35, 96, 220

L

lista

- rozwijana, 244, 247
- urządzeń, 340
- zmian, 152

log, 275

Ł

łączenie

- aktywności, 337
- gałęzi, 157

M

makieta, 129

makra, 376

marginies, 47, 189

menedżer ustawień, 246

menu, 377

- akcji, 96

- kontekstowe, 55, 123

- kontekstowe zakładek, 47

menu

- opcji, 241
- paska akcji, 109
- Refactor This, 80

metoda

- add(), 71
- calculateAnswer(), 281
- checkanswer(), 281
- CurrencyConverterTask, 255
- doInBackground(), 235, 254, 256
- eventuallyHideAnswer(), 281
- extractCodeFromCurrency(), 248
- findPositionGivenCode(), 247
- getCount(), 197
- getIdFromPosition(), 127
- getItemId(), 197
- getKey(), 252
- hasAlpha(), 70
- insertSomeReminders(), 120
- Integer.parseInt(), 288
- isNumeric(), 288
- isOnline(), 242
- onClick(), 267
- onCreate(), 176, 195, 213, 244, 252
- onDestroy(), 176, 363
- onDraw(), 366
- onItemClick(), 122
- onItemSelected(), 245
- onListItemClick(), 215
- onListItemSelected(), 214
- onOptionsItemSelected(), 110, 134, 241
- onPause(), 176
- onPostExecute(), 235, 254
- onPreExecute(), 256
- onRestart(), 176
- onResume(), 176
- onStart(), 176
- onStop(), 176
- setString(), 246
- setUp(), 262, 263
- substring(), 247
- tearDown(), 262
- toString(), 70
- updateTimer(), 364

metody

- cyklu życia aktywności, 176
- pobierające, 68
- ustawiające, 68

model

- danych, 111
- pobierania, 173
- wysyłania, 173

moduł

- App Engine Java Endpoints, 344
- Bitbucket, 381
- serwerowy, 343

moduły dodatkowe, 380

modyfikacja

- klasy MainActivity, 204
- klasy Person, 206
- metody onOptionsItemSelected(), 241
- układu aktywności, 223

monitor Androida, 332

motyw Darcula, 375

N

narzędzia, 21

- analityczne, 269
- Android SDK, 325
- chmury Google, 341
- nawigacyjne, 49
- SDK, 354

narzędzie

- Allocation Tracker, 334
- Android Monitor, 335
- AVD Manager, 354
- Debugger, 289
- Favorites, 50
- Git, 139
- Gradle, 297
- logcat, 276
- Method Trace, 333
- Monkey, 268
- Project, 49
- Structure, 49
- TODO, 51

nawias klamrowy, 64

nawigacja, 54

O

obiekt

- Date, 148
- RemoteCloudBeanAsyncTask, 348

obliczanie wyrażenia, 284

- obsługa
 - API, 125
 - naciśnięcia przycisku, 250
 - odgałęzienia, 146
 - odłączenie od gałęzi, 163, 165
 - odtworzenie nagrania, 336
 - okna
 - narzędzi nawigacyjnych, 49
 - narzędziowe, 46
 - okno
 - Analyze Stacktrace, 272
 - Android Device Monitor, 326
 - Breakpoints, 289
 - Choose Device, 42
 - Choose Fields to Initialize by Constructor, 93
 - Code Style Schemes, 374
 - Commit Changes, 151
 - Convert Anonymous to Inner, 94
 - Create New Class, 64
 - Create New Logcat Filter, 277
 - Dependency Viewer, 271
 - Edycja przypomnienia, 98
 - edycji przypomnienia, 136
 - edytora, 46
 - edytora nawigacji, 338
 - Evaluate Expression, 285
 - Extract Method, 88, 120, 248
 - Files Merged with Conflicts, 169
 - logcat, 272
 - narzędzia Favorities, 50
 - narzędzia Project, 49
 - narzędzia Structure, 49
 - narzędzia TODO, 51
 - New Changelist, 157
 - New Resource File, 104
 - podglądu, 177, 181
 - Process Duplicates, 88
 - Pull Members Up, 90
 - Rebase branch, 163
 - Rebasing Commits, 164
 - Reformat File, 372
 - Replace Constructor With Factory Method, 93
 - Replace Inheritance With Delegation, 90
 - Reset Head, 156, 159
 - Resources, 201
 - Run, 273
 - Select Methods to Implement, 245
 - Settings, 372, 373
 - Specify Inspection Scope, 269
 - Use Style Where Possible, 229
 - Version Control, 167, 222
 - opcja Launch Emulator, 39
 - opcje
 - dotyczące Surround With, 76
 - dotyczące szablonów, 72
 - komentowania kodu, 66
 - narzędzia logcat, 276
 - okna podglądu, 178
 - organizacji kodu, 75
 - uzupełniania kodu, 63
 - Wear, 358
 - zawijania kodu, 62
 - operacja
 - Change Signature, 81
 - Convert Anonymous to Inner, 93
 - Copy, 84
 - Encapsulate Fields, 91
 - Extract Constant, 85
 - Extract Field, 86
 - Extract Method, 87
 - Extract Parameter, 86
 - Extract Variable, 85
 - Git Rebase, 163
 - Move, 82
 - Pull Members Up, 89
 - Push Members Down, 89
 - Rename, 80
 - Replace Constructor with Factory Method, 92
 - Replace Inheritance with Delegation, 89
 - Safe Delete, 84
 - Type Migration, 82
 - Wrap Method Return Value, 92
 - operacje
 - CRUD, 113
 - refaktoryzacji, 79
 - wydobywania, 84
 - optymalizacja, 358
 - otwarcie bazy danych, 113
 - otwieranie starszych projektów, 322
- ## P
- paczka, bundle, 237
 - pasek
 - akcji, 219
 - głównego menu, 51
 - narzędziowy, 52
 - nawigacyjny, 52
 - statusu, 52
 - znaczników, 48
 - paski narzędziowe, 377

- perspektywa, 330
- pierwszy projekt, 35
- piksel niezależny od gęstości, 199
- platforma IntelliJ IDEA, 299
- plik
 - .gitignore, 300
 - activity_main.xml, 210, 223, 342
 - activity_splash.xml, 234
 - AndroidManifest.xml, 204, 242
 - build.gradle, 302
 - colors.xml, 226
 - dialog_about.xml, 161
 - dialog_custom.xml, 131
 - gradle.properties, 300
 - gradlew.bat, 300
 - JDK 8 Update 60.pkg, 30
 - JSONParser.java, 231
 - keys.properties, 251
 - list_view.xml, 193
 - local.properties, 300, 303
 - MainActivity.java, 61, 242, 263, 304
 - menu_main.xml, 241
 - NationalWeatherRequest.java, 311, 316
 - NationalWeatherRequestData.java, 317
 - Project.iml, 300
 - R.java, 259
 - relative_example.xml, 188
 - ReminderAlarmReceiver.java, 149
 - reminders_row.xml, 106
 - RemindersActivity, 126
 - settings.gradle, 300, 303
 - spinner_closed.xml, 243
 - SplashActivity.java, 235
 - TemperatureAdapter.java, 306, 318
 - TemperatureItem.java, 307
 - WeatherParseTest.java, 314
- pliki zmian, 173
- pobieranie JDK, 23, 29
- pobranie klucza API, 252
- podpowiedzi, 64, 149
- pole
 - tekstowe, 178
 - TextView, 218
- polecenia cofania i przywracania, 53
- polecenie
 - Auto-Indent Lines, 74
 - Checkout, 157
 - Class, 56
 - cofnięcia, 153
 - Continue Rebasing, 170
 - Declaration, 57
 - File, 57
 - Find, 58
 - Find in Path, 58
 - git revert, 154
 - Last Edit Location, 57
 - Line, 57
 - Rearrange Code, 75
 - Reformat Code, 76
 - Related File, 57
 - Replace, 58
 - Replace in Path, 58
 - Reset, 159
 - Resolve Conflicts, 168
 - Select In, 56
 - Surround With, 76, 77
 - Type Hierarchy, 57
- pomoc, 55
- prognoza pogody, 302
- program Setup Wizard, 34
- programowanie, 61, 146
 - sterowane testami, 261
- projekt
 - MegaDroid, 357
 - prognozy pogody, 302
- projektowanie
 - logowania, 339
 - interfejsu użytkownika, 337
 - układu graficznego, 175, 198
- przechowywanie klucza API, 251
- przeglądanie
 - historii zmian, 145
 - interfejsu aplikacji, 331
- przeglądarka
 - hierarchii, 329
 - punktów wstrzymania, 289
- przełączanie perspektyw, 331
- przenoszenie kodu, 73, 74
- przesłanianie metod, 68
- przesyłanie zmian, 172
- przycisk, 257
 - zmiany orientacji, 105
- przyciski narzędzi, 48
- przypomnienia, 95, 110, 119
- punkty wstrzymania, 283, 289
 - opcje, 290
 - warunkowe, 292

R

refaktoryzacja
 kodu, 79
 zaawansowana, 89
 referencje względne, 166
 rejestracja usługi, 361
 repozytorium Git, 97, 221
 rozdmuchanie, inflacje, 239
 rozdzielczość ekranu, 198
 rozmiar obrazu, 200
 rozwiązywanie konfliktów, 167
 rozwidlanie, 143
 rysowanie
 tarczy zegara, 366
 tła, 367

S

selektor przycisku, 257
 serwer API, 345
 serwery chmury Google, 349
 sesja terminalu, 268
 składnia Gradle, 298
 skróty klawiaturowe, 56, 376
 skrypt budowania narzędzia Gradle, 299
 specyfikacja aplikacji, 217
 stos wywołań, 270, 286
 stosowanie stylów, 228
 styl, 228, 362
 kodu, 74, 371
 label.curr, 229
 label.desc, 229
 layout_back, 230
 symulacja, 37
 system
 budowania Gradle, 299
 HAXM, 354
 kontroli wersji, VCS, 139
 szablon, 70, 72
 Blank Activity, 342
 CurrencyLayout, 378
 szablony
 kodu, 377
 plików, 377

Ś

ścieżka pozytywna, 275

T

tarcza zegara, 366
 terminal, 268, 340
 test instrumentacji, 266
 testowanie, 261
 testy instrumentacyjne, 262
 tryb
 odłączenia od gałęzi, 164
 programisty, 356
 projektowania, 180
 projektowania pliku, 99
 tła, 358
 tworzenie
 aktywności, 338
 aktywności ekranu, 232
 biblioteki Android, 309
 gałęzi, 146, 302
 ikon, 259
 interfejsu użytkownika, 101
 klasy, 64
 klasy JSONParser, 231
 menedżera ustawień, 246
 menu kontekstowego, 123
 menu opcji, 241
 metod testujących, 266
 modułu, 344
 modułu serwerowego, 343
 nowego projektu, 96
 okna dialogowego, 132
 projektu MegaDroid, 357
 repozytorium Git, 99
 rewizji, 152
 rozgałęzień w Git, 152
 stylów, 228
 szablonu, 378
 testu instrumentacyjnego, 262
 układu spinner_closed, 243
 usługi WatchFace, 360

U

układ
 graficzny, 102, 103, 177
 spinner_closed, 243
 z fragmentami, 212
 uprawnienia, 352
 uruchamianie
 aplikacji, 39, 41
 MainActivity, 236
 testu instrumentacji, 266

urządzenie wirtualne Wear, 354
 usługa WatchFace, 360
 usługi Google, 351
 usprawnienia wizualne, 106
 usuwanie przypomnień, 119, 124, 128
 uzupełnianie kodu, 63, 65
 uzyskiwanie pomocy, 55
 użycie MultiChoiceModeListener, 127

V

VCS, Version Control System, 139

W

waluty, 217, 239
 warunkowe punkty wstrzymania, 292
 wątek, 252

- główny, 253

 wdrożenie, 349
 wielkość ekranu, 199
 wklejanie, 54
 własna ikona, 137
 własne okna dialogowe, 122, 129, 132
 właściwości punktu wstrzymania, 290
 właściwość layout_weight, 223
 włączenie opcji, 358
 wskazówki zegara, 368
 wybór

- kategorii Wear, 354
- metod, 71
- obrazu systemu, 355
- opcji, 37

 wycinanie, 54
 wycofanie metod, 153
 wydobycie kodu waluty, 247
 wygląd, 374

- aplikacji, 175
- domyślny, 48

 wyświetlanie obrazu, 359
 wyjątek NumberFormatException, 267
 wykorzystanie stosu wywołań, 286
 wywołanie zwrotne, 264, 364

Z

zagnieżdżanie komponentów, 188
 zakładka

- Allocation Tracker, 328
- Arrangement, 373

Code Generation, 67
 Heap, 326
 MainActivity.java, 47
 Memory, 332
 Network Statistics, 329
 Threads, 325
 Wrapping and Braces, 373
 zakładki edytora, 46
 zależności, 270
 zależność

- od bibliotek Javy, 312
- od bibliotek systemu Android, 307

 zamknięcie bazy danych, 113
 zapis danych w dzienniku, 278
 zarys zawinięcia, 62
 zasoby, 362
 zastępowanie tekstu, 58
 zastosowanie kolorów, 227
 zawijanie kodu, 61
 zawinięty blok kodu, 62
 zaznaczanie tekstu, 53
 zdalne repozytoria Git, 172
 zegar, 357, 362
 zintegrowane środowisko programistyczne, IDE, 46
 zmiana

- bazy, 163
- historii, 159
- sygnatury metody, 81

 zmienna

- JAVA_HOME, 28
- PATH, 28

 zmienne środowiskowe, 24
 znajdowanie

- ostatnio otwieranych plików, 54
- tekstu, 58

 zrzut ekranu, 335

Ż

żądanie pobrania zmian, 143, 172

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>

Platforma Android dynamicznie powiększa swój udział w rynku. Jej niewątpliwymi zaletami są modułowa architektura, wysoka elastyczność i otwarty charakter systemu. Na świecie działają miliardy urządzeń pracujących pod kontrolą Androida i z pewnością będzie ich coraz więcej. A programiści, którzy zdobędą umiejętność efektywnego pisania atrakcyjnych aplikacji w tym systemie, będą mogli spokojnie patrzeć w świetlaną przyszłość.

Niniejsza książka stanowi wyczerpujący podręcznik do nauki obsługi środowiska Android Studio — efektywnego, intuicyjnego, potężnego i niezwykle wygodnego IDE. Opisano tu m.in. wszystkie istotne narzędzia tego środowiska, system kontroli wersji Git i skrypty Gradle oraz ich integrację z IDE. Dodatkowo przedstawiono techniki tworzenia i rozwijania aplikacji w Androidzie. Książka zawiera cztery pełne przykładowe projekty, udostępnione do pobrania z publicznego repozytorium Git.

Dzięki tej książce:

- **plynnie i bezproblemowo rozpoczniesz pisanie aplikacji w Android Studio**
- **zacznieš korzystac z systemu kontroli wersji Git**
- **nauczysz się stosowac skrypty Gradle**
- **będiesz używac frameworka Android Wear**
- **nauczysz się testowac aplikacje i debugowac kod za pomoca Android Studio**
- **dowiesz się, jak najskuteczniej zarzadzac wieloma podprojektami**

Adam Gerber — pracuje na Uniwersytecie Illinois. Jest członkiem Chicago Innovation Exchange. Był jednym z pierwszych użytkowników Android Studio i — jak twierdzi — od razu odkrył jego zalety. Używa go zarówno w celu tworzenia profesjonalnych aplikacji, jak i w celu nauczania studentów programowania.

Clifton Craig — od kilkunastu lat pracuje jako inżynier oprogramowania. Zajmował się systemami mobilnymi J2ME/BlackBerry, Android oraz iOS, a także systemami serwerowymi opartymi na JEE. Pracował przy wielu wysokoprofilowych projektach, takich jak portal MapQuest Gas Prices, MapQuest for Mobile dla platform J2ME i Android, a także Skype dla platform iOS i Android.

Helion

40805

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>

Informatyka w najlepszym wydaniu

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-283-2009-3



9 788328 320093

cena: 67,00 zł