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

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.

Spis treści

ROZDZIAŁ 15

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 przyjrzeć 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.

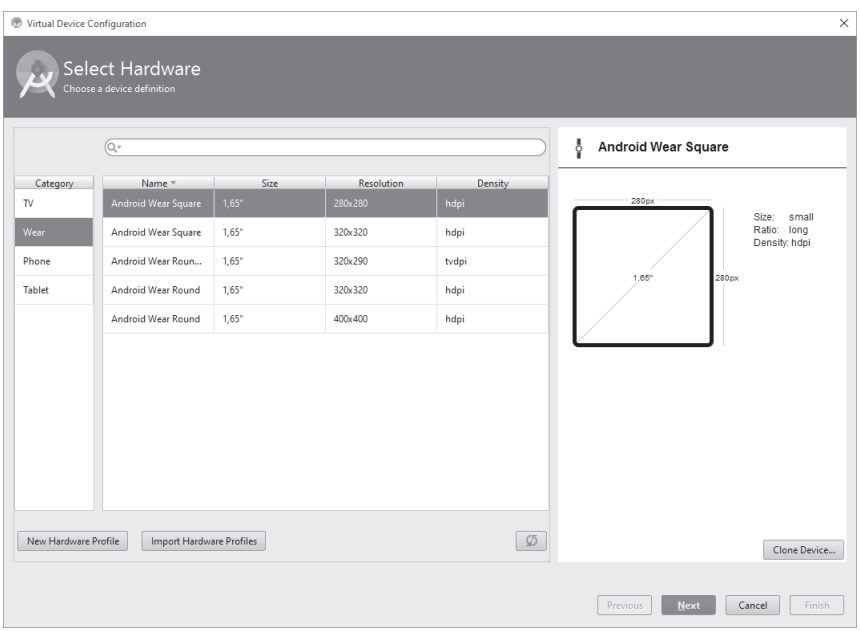
Otwórz *Menedżer urządzeń* i znajdź urządzenie na liście wewnątrz grupy *Inne urządzenia*. Kliknij je prawym klawiszem myszy i wybierz polecenie *Aktualizuj oprogramowanie sterownika*. W oknie, które się pojawi, wybierz drugą opcję (*Przeglądaj mój komputer w poszukiwaniu oprogramowania sterownika*), a następnie opcję *Pozwól mi wybrać z listy sterowników urządzeń ma 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*.

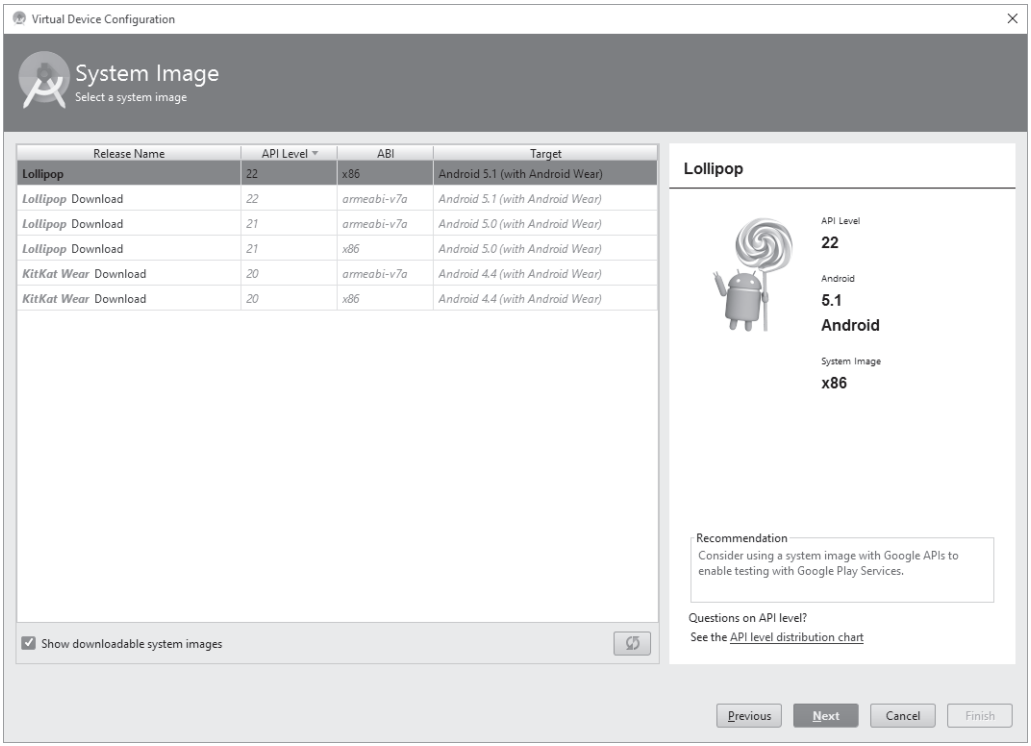
Konfiguracja urządzenia wirtualnego Wear

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 pojawią 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*

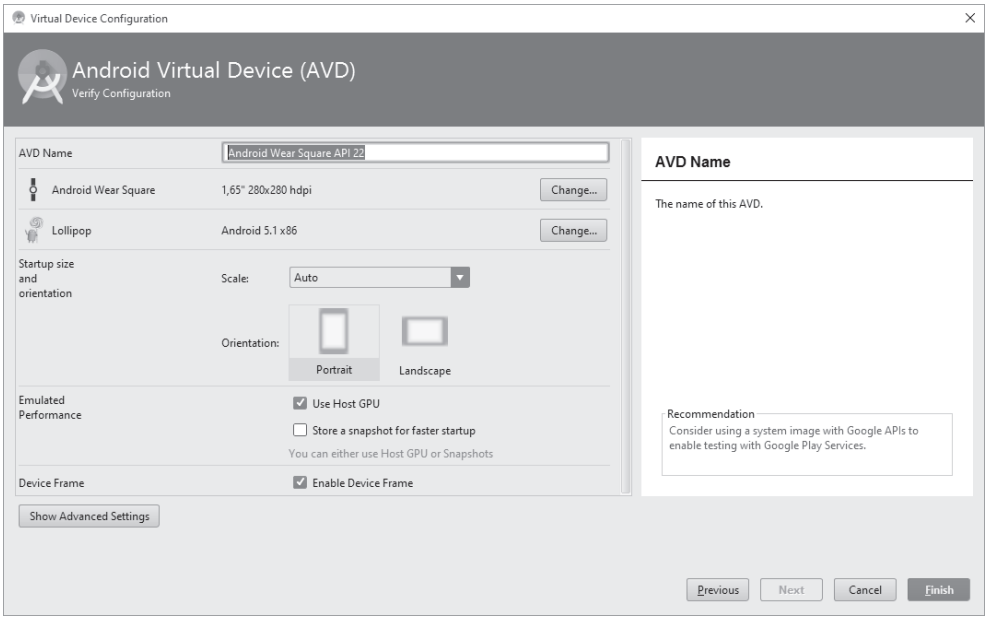
Nadaj AVD nazwę Android Wear Square API 22 na ekranie przedstawionym na rysunku 15.3.

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.

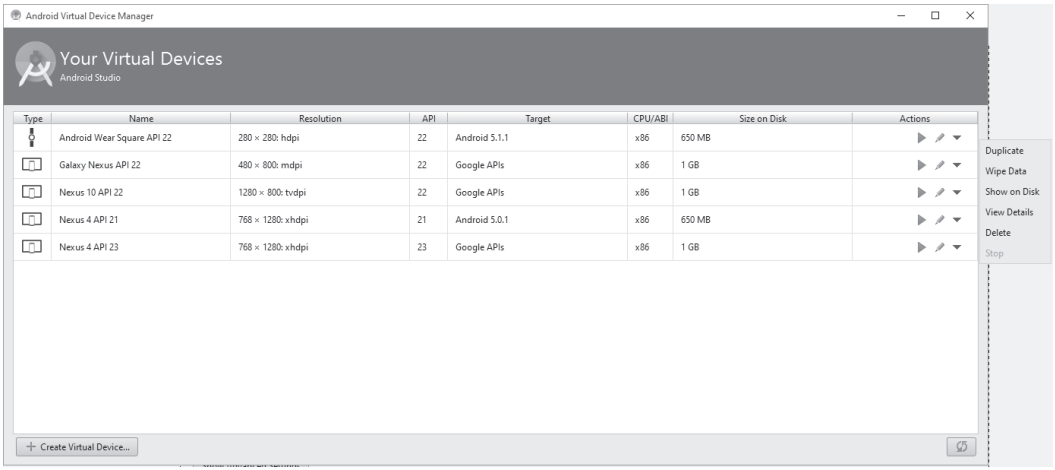
Konfiguracja sprzętu Android Wear

Osoby posiadające urządzenie Android Wear muszą je odpowiednio skonfigurować, aby móc go używać do testowania aplikacji. Aplikacje 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. *Zduplikuj AVD Android Wear Square API 22*

Włączenie trybu programisty

Jeśli nigdy wcześniej nie był włączany 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. Wejdź do nowego elementu i włącz 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
```

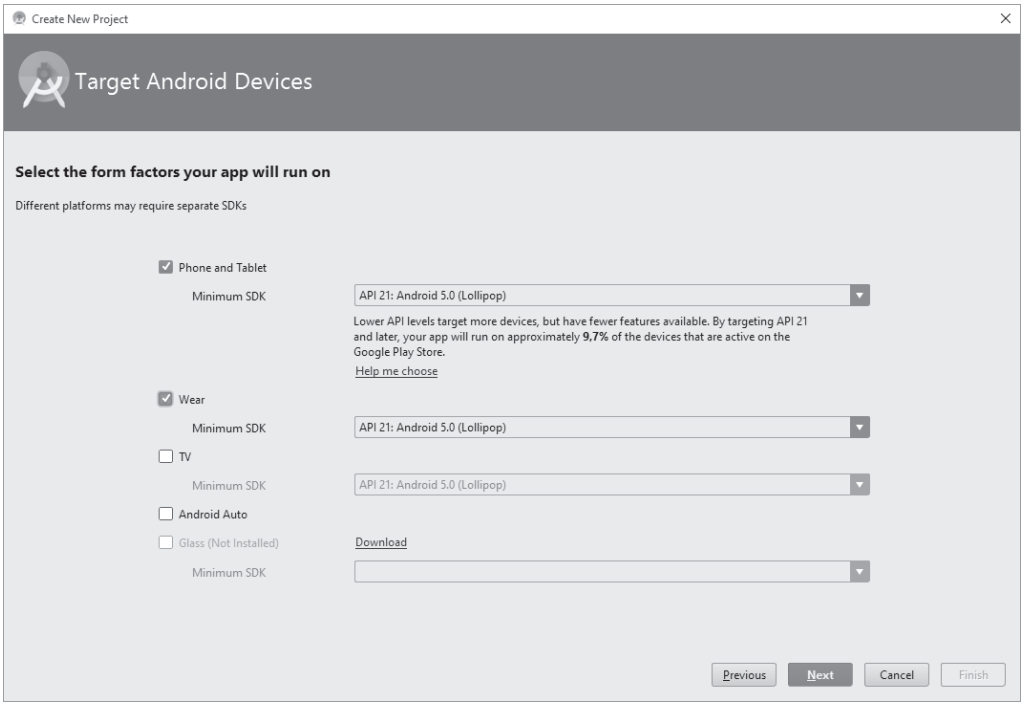
Od tego momentu aplikacja i zegarek są gotowe do instalacji aplikacji.

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 zegara 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.sendEmptyMessageDelayed(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 zadanym 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 zwrotne `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.sendEmptyMessage(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 zwrotnego `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 aktualizowania 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. Zacznijmy 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/π. 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);
```

Wskazówka 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);
```

Wskazówka 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

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

Z

Ż

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:

- **płynnie i bezproblemowo rozpoczniesz pisanie aplikacji w Android Studio**
- **zaczniesz korzystać z systemu kontroli wersji Git**
- **nauczysz się stosować skrypty Gradle**
- **będziesz używać frameworka Android Wear**
- **nauczysz się testować aplikacje i debugować kod za pomocą Android Studio**
- **dowiesz się, jak najskuteczniej zarządzać 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 nauczenia 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

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

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

Informatyka w najlepszym wydaniu

ISBN 978-83-283-2009-3

cena: 67,00 zł