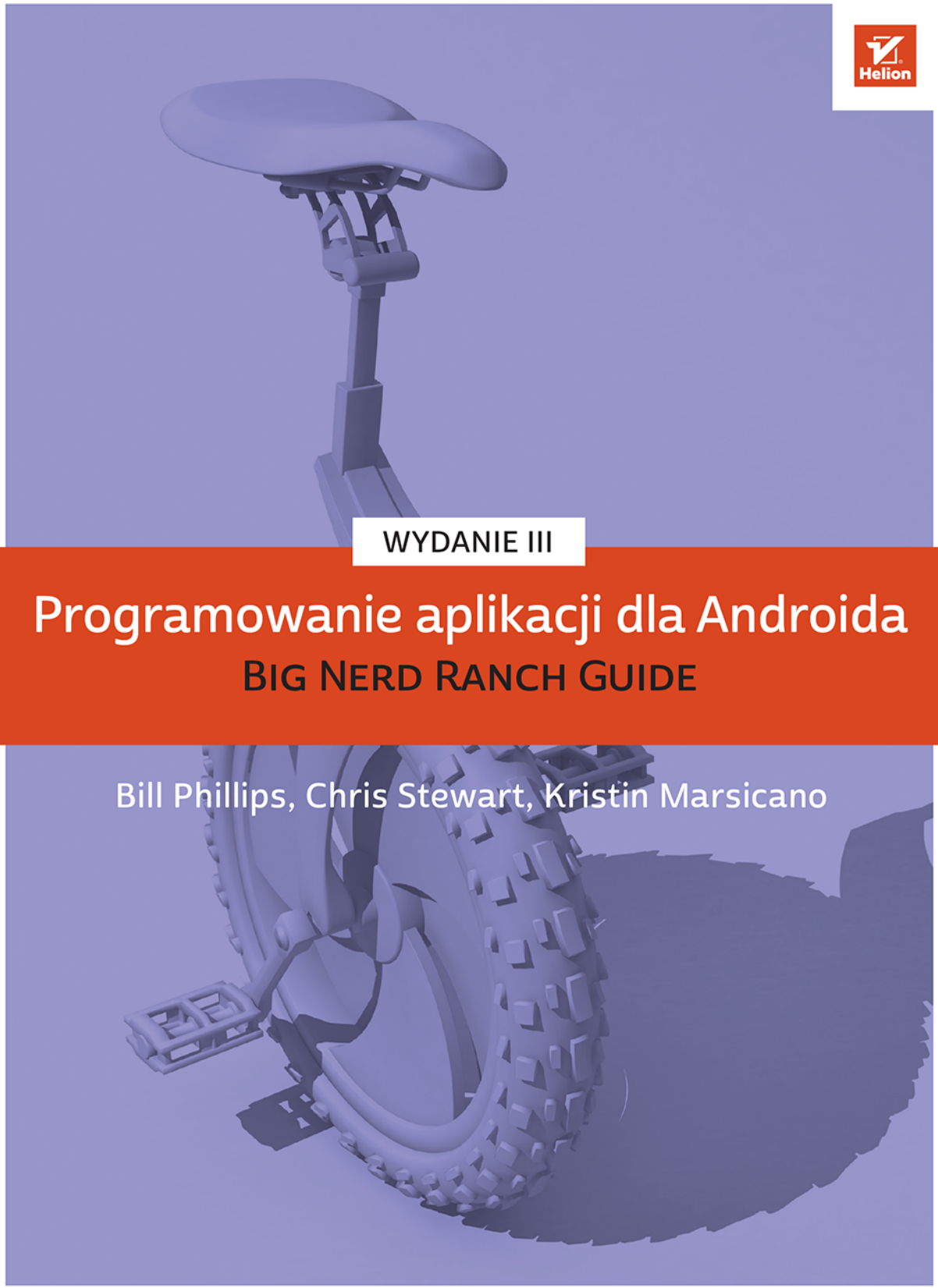


WYDANIE III

# Programowanie aplikacji dla Androida

## Big Nerd Ranch Guide

Bill Phillips, Chris Stewart, Kristin Marsicano

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

Tłumaczenie: Grzegorz Kowalczyk

ISBN: 978-83-283-3636-0

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


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)

# Spis treści

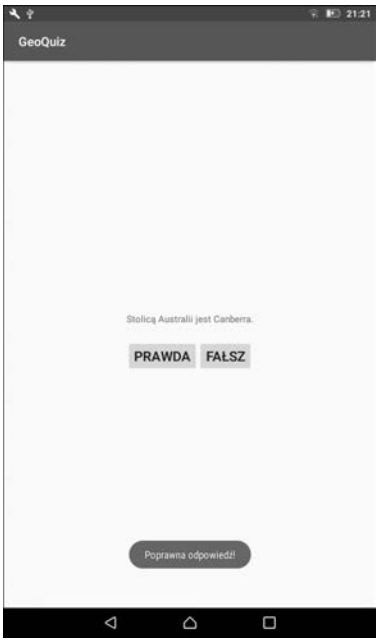
# 1

# Twoja pierwsza aplikacja dla systemu Android

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

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

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




Rysunek 1.1 Czy pochodzisz z antypodów?

# Podstawowe elementy aplikacji

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

- **Aktywność** jest instancją klasy `Activity`, wchodzącej w skład biblioteki Android SDK. Aktywności odpowiadają za zarządzanie interakcjami z użytkownikiem na ekranie aplikacji.

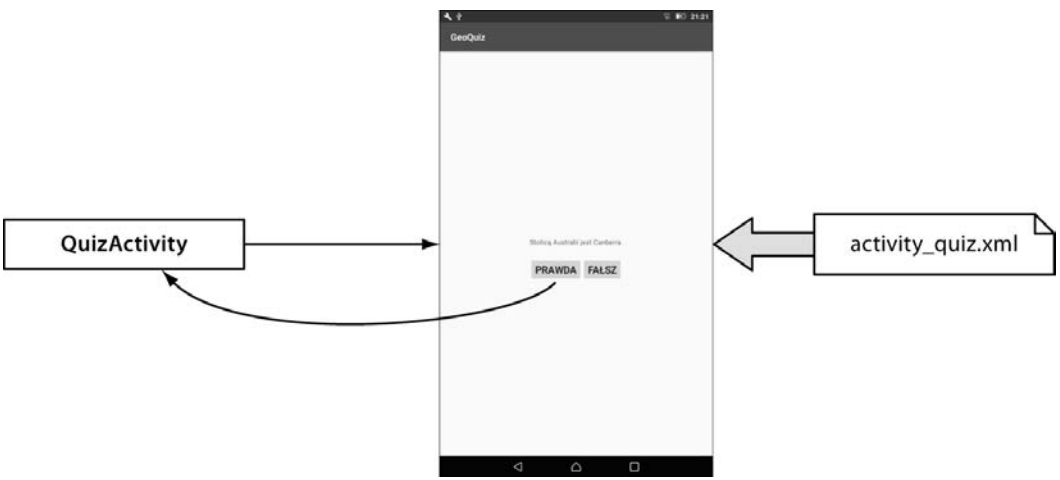
  Aby zaimplementować poszczególne funkcjonalności aplikacji, będziesz tworzył odpowiednie podklasy klasy `Activity`. W prostych aplikacjach często wystarczy tylko jedna podklasa, bardziej złożone aplikacje mogą wymagać zaimplementowania bardzo wielu różnych podklas.

  GeoQuiz jest prostą aplikacją, zatem będzie mieć tylko jedną podklasę klasy `Activity`, której nadamy nazwę `QuizActivity`. Zadaniem podklasy `QuizActivity` będzie zarządzanie interfejsem użytkownika, który został przedstawiony na rysunku 1.1.

- Za pomocą **układu** definiujemy zestaw obiektów interfejsu użytkownika i ich położenie na ekranie aplikacji. Układy mają postać zbiorów definicji obiektów zapisanych w postaci kodu XML. Poszczególne definicje są wykorzystywane do tworzenia obiektów pojawiających się na ekranie, takich jak przyciski czy tekst.

  Częścią aplikacji GeoQuiz będzie plik układu o nazwie *activity_quiz.xml*. Kod XML zapisany w tym pliku zawiera definicje elementów interfejsu użytkownika przedstawionego na rysunku 1.1.

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




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

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

# Tworzenie projektu aplikacji dla systemu Android

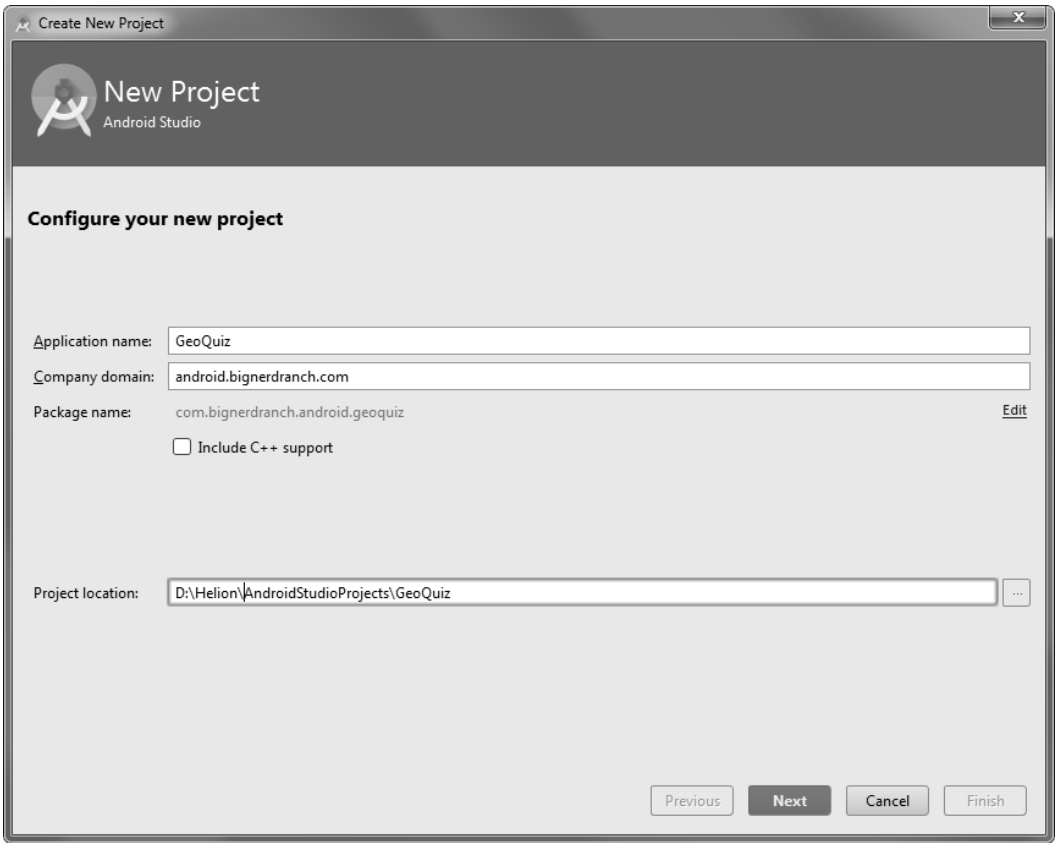
Pierwszym etapem tworzenia aplikacji dla systemu Android jest utworzenie jej **projektu**. Taki projekt zawiera wszystkie pliki składowe aplikacji dla systemu Android. Aby utworzyć nowy projekt, musisz najpierw uruchomić program Android Studio.

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


Rysunek 1.3. Okno powitalne programu Android Studio

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

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



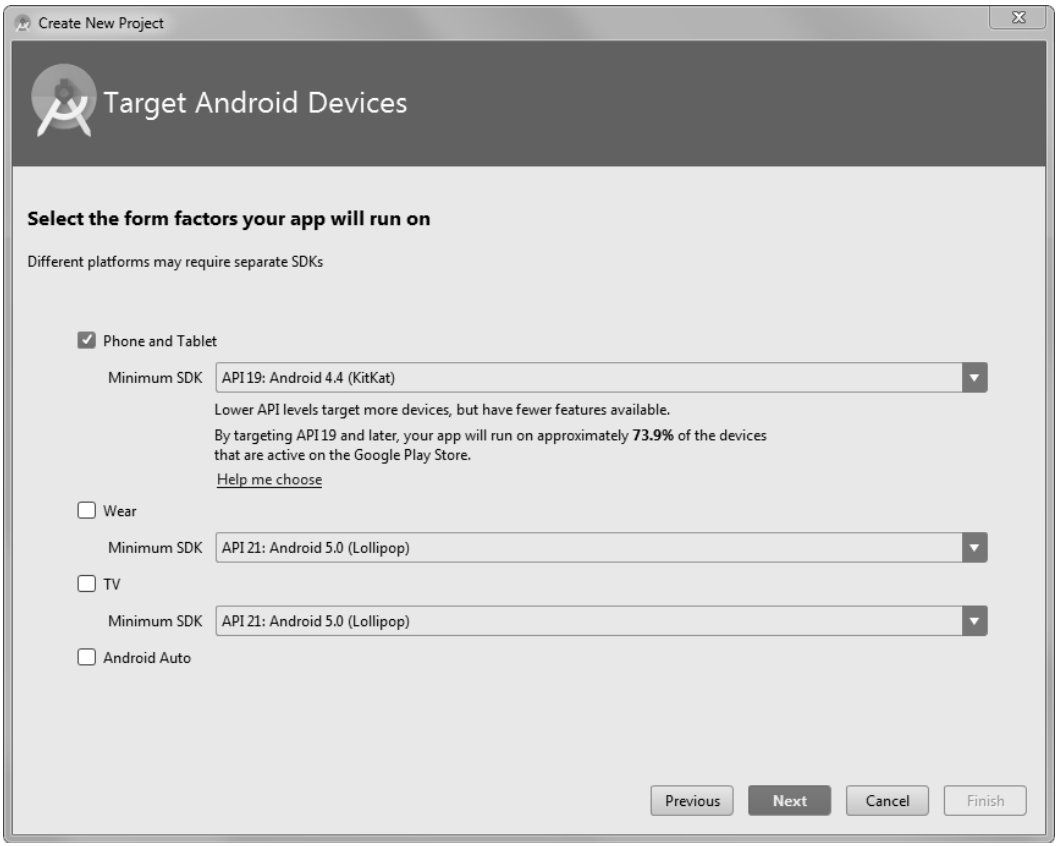

Rysunek 1.4. Tworzenie nowej aplikacji

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

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

Naciśnij przycisk *Next*.

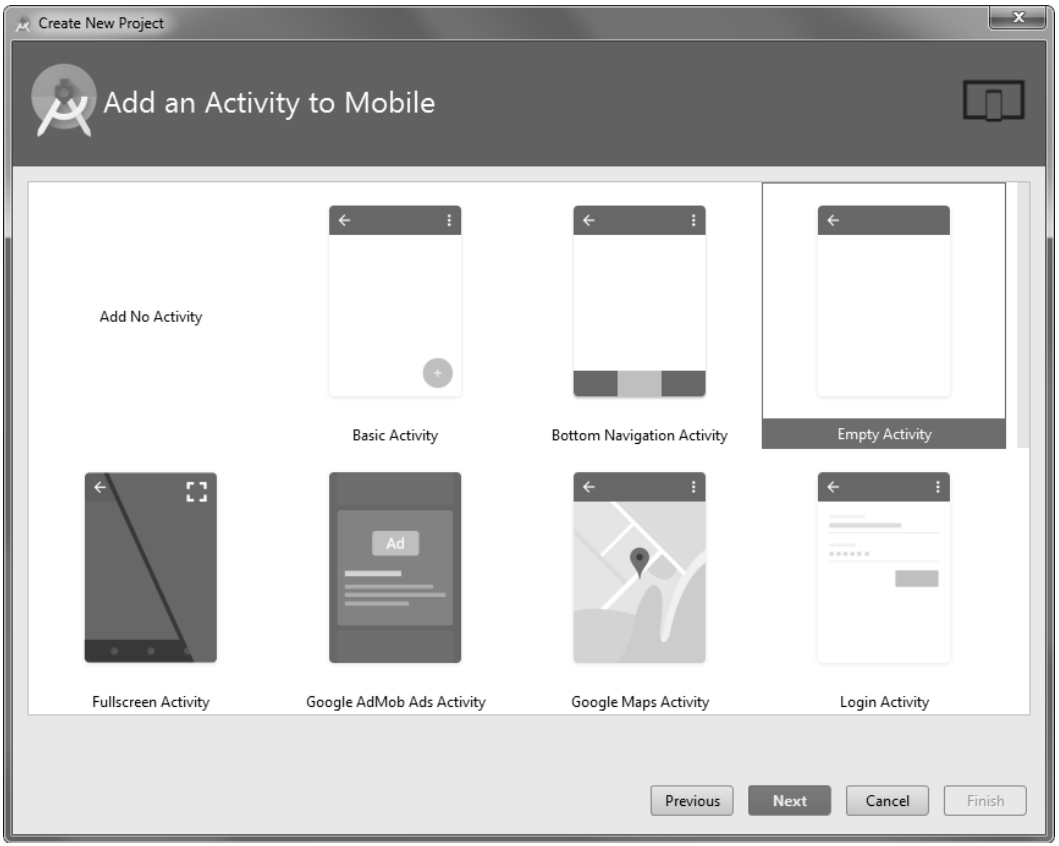
Na kolejnym ekranie kreatora będziesz miał możliwość wybrania szablonu dla pierwszego ekranu aplikacji GeoQuiz (zobacz rysunek 1.6). Wybierz najprostszy, pusty szablon o nazwie *Empty Activity* (pusta aktywność) i naciśnij przycisk *Next*.




Rysunek 1.5. Wybieranie obsługiwanych urządzeń

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

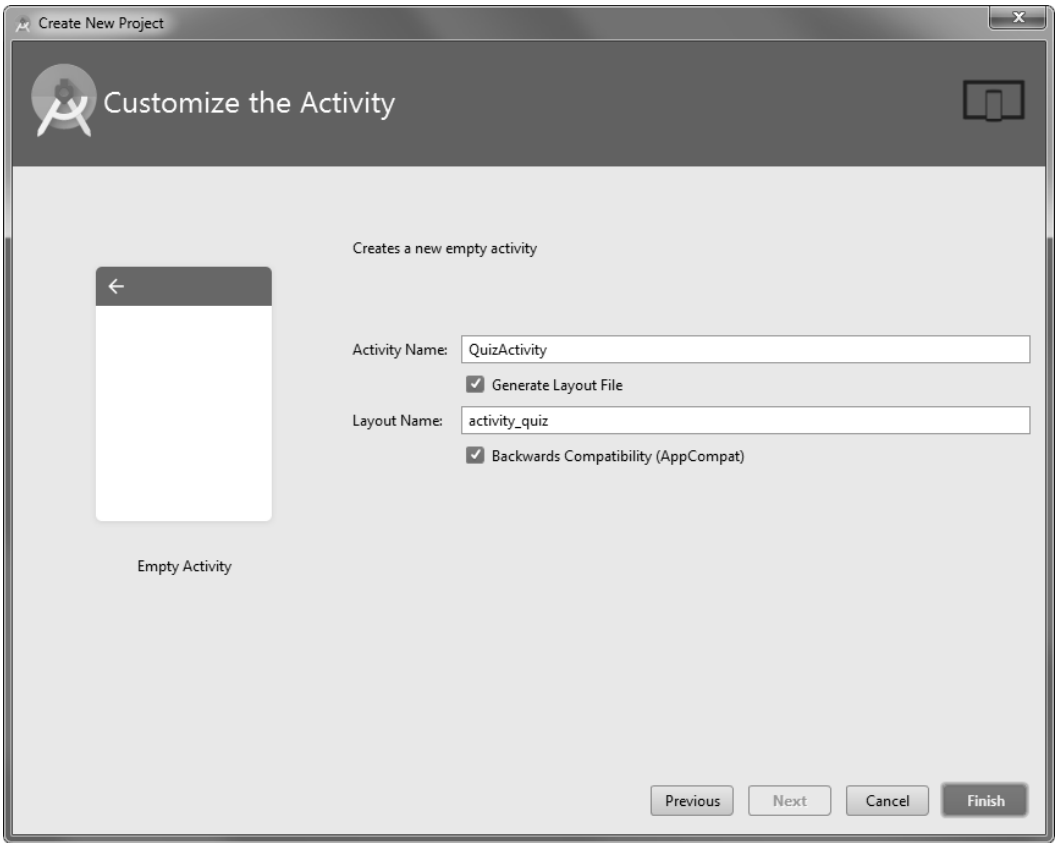
W ostatnim oknie kreatora jako nazwę podklasy aktywności w polu *Activity Name* wpisz **`QuizActivity`** (zobacz rysunek 1.7). Zwróć uwagę na przyrostek `Activity` w nazwie klasy. Stosowanie takich przyrostków nie jest oczywiście w żaden sposób obowiązkowe, niemniej jednak zachowanie takiej konwencji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




Rysunek 1.6. Wybieranie typu aktywności

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

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




Rysunek 1.7. Konfigurowanie nowej aktywności

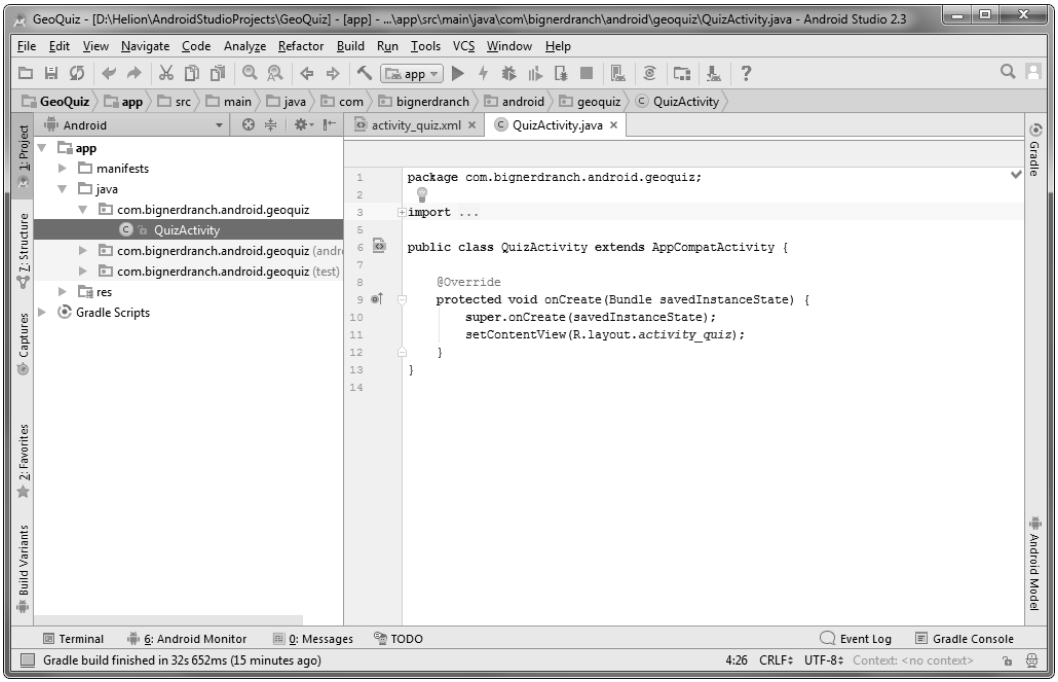
# Poruszanie się w Android Studio

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

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

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

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




Rysunek 1.8. Okno nowo utworzonego projektu

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

# Tworzenie układu interfejsu użytkownika

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

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

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

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

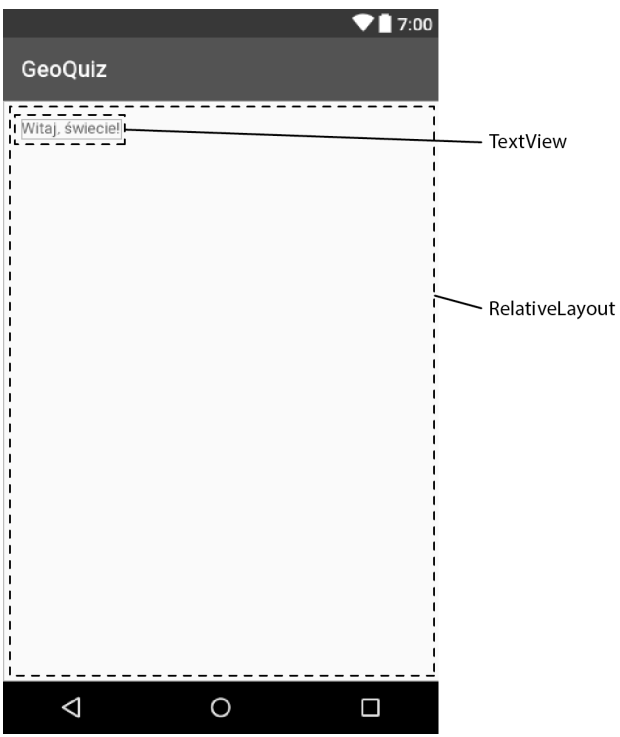
```
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.bignerdranch.android.geoquiz.QuizActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
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

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

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

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

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

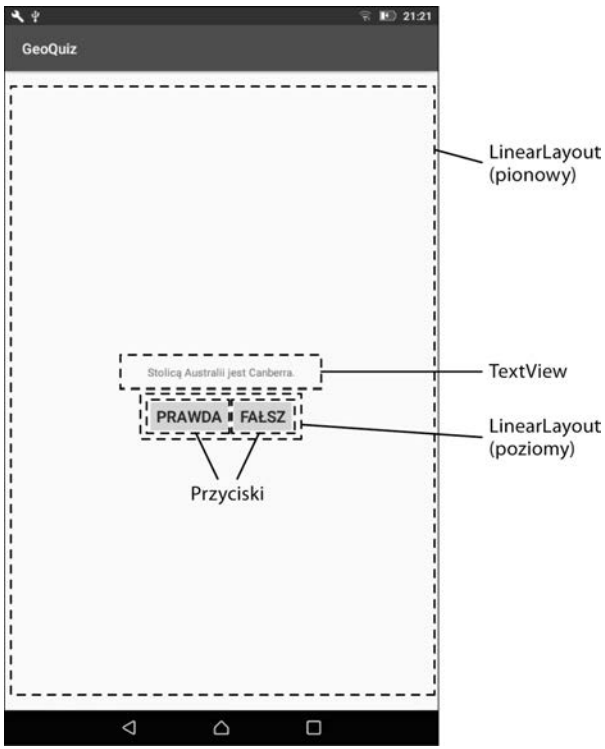



Rysunek 1.9. Wygląd domyślnych widgetów na ekranie

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

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

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




Rysunek 1.10. Planowany wygląd widgetów na ekranie

Teraz musimy zdefiniować nasze widgety w pliku układu *activity_quiz.xml*.

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

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

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

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

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

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

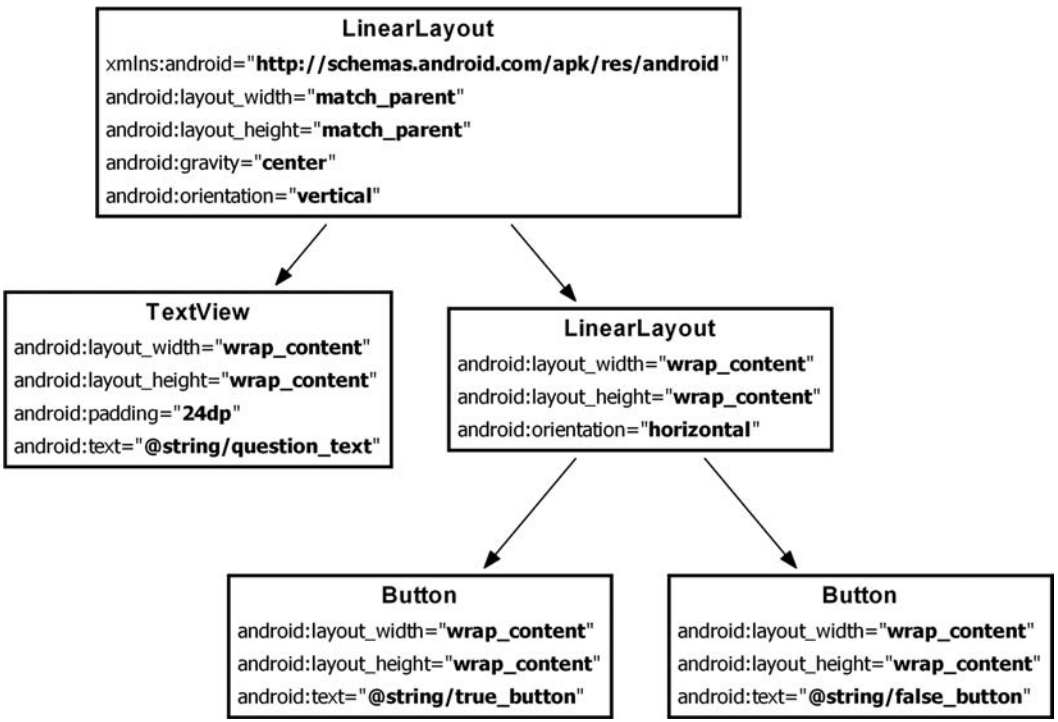
Porównaj kod XML z wyglądem interfejsu użytkownika przedstawionego na rysunku 1.10. Każdemu widgetowi przypisany jest odpowiedni element kodu XML, którego nazwa determinuje typ definiowanego widgetu.

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

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

## Hierarchia widoków

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




Rysunek 1.11. Hierarchiczny układ widgetów i atrybutów

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

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

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

# Atrybuty widgetów

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

## Atrybuty android:layout_width oraz android:layout_height

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

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

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

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

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

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

## Atrybut android:orientation

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

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

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

### Atrybut android:text

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

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

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

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

## Tworzenie zasobów tekstowych

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

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

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

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

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

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

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

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

## Podgląd układu

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


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

# Od pliku układu XML do obiektów klasy View

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

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

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

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

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

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

public class QuizActivity extends AppCompatActivity {

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

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

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

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

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

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

public void setContentView(int layoutResID)

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

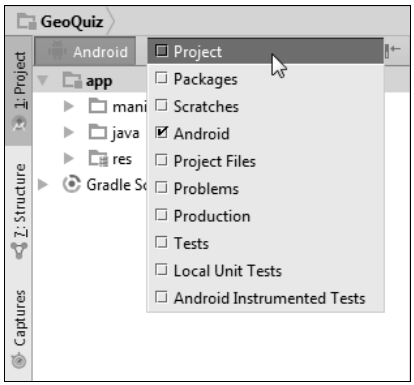
## Zasoby i identyfikatory zasobów

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

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

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

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




Rysunek 1.13. Zmiana sposobu wyświetlania projektu

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# Podłączanie widgetów

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

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

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

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

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

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

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

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

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

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

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

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

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

## Pobieranie odwołań do widgetów

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

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

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

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

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

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

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

## Tworzenie obiektów nasłuchujących

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

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

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

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

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

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

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

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

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

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

## Zastosowanie anonimowych klas wewnętrznych

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

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

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

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

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

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

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

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

# Tworzenie komunikatów toast

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


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

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

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

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

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

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

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

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

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

## Zastosowanie mechanizmu dopełniania kodu

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

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

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

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

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

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

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

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

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

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

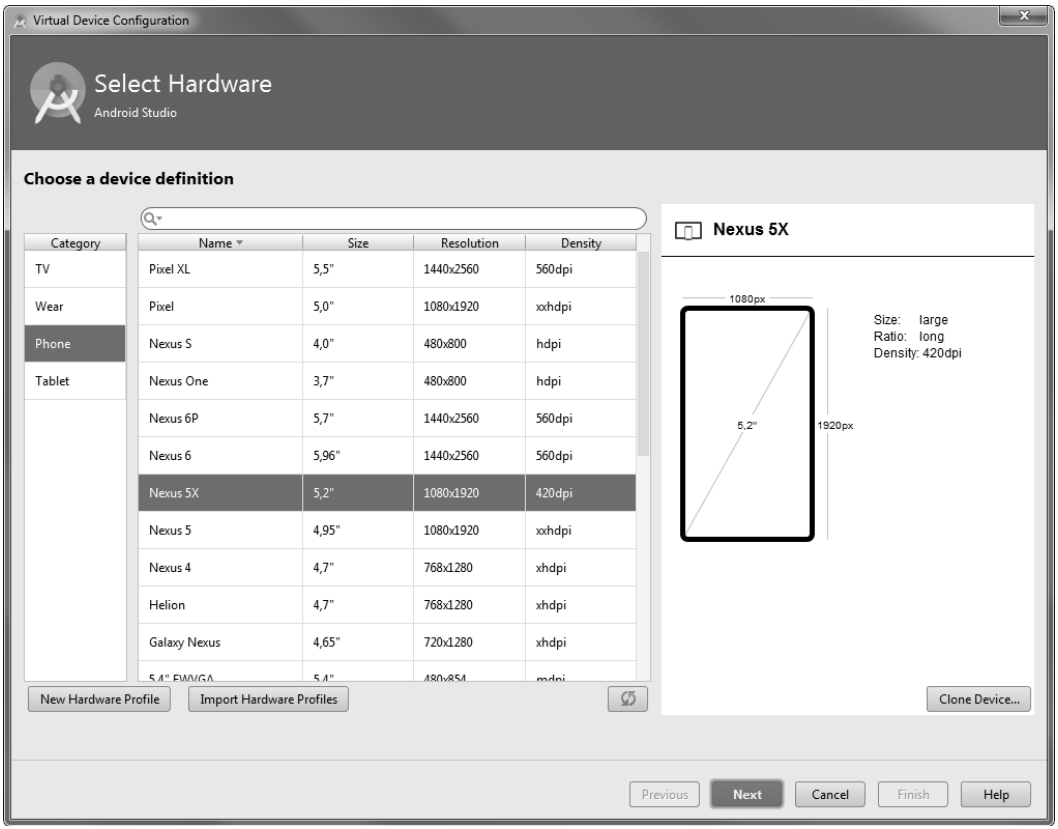
Zobaczmy teraz naszą nową aplikację w działaniu.

# Uruchamianie aplikacji w emulatorze

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

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

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

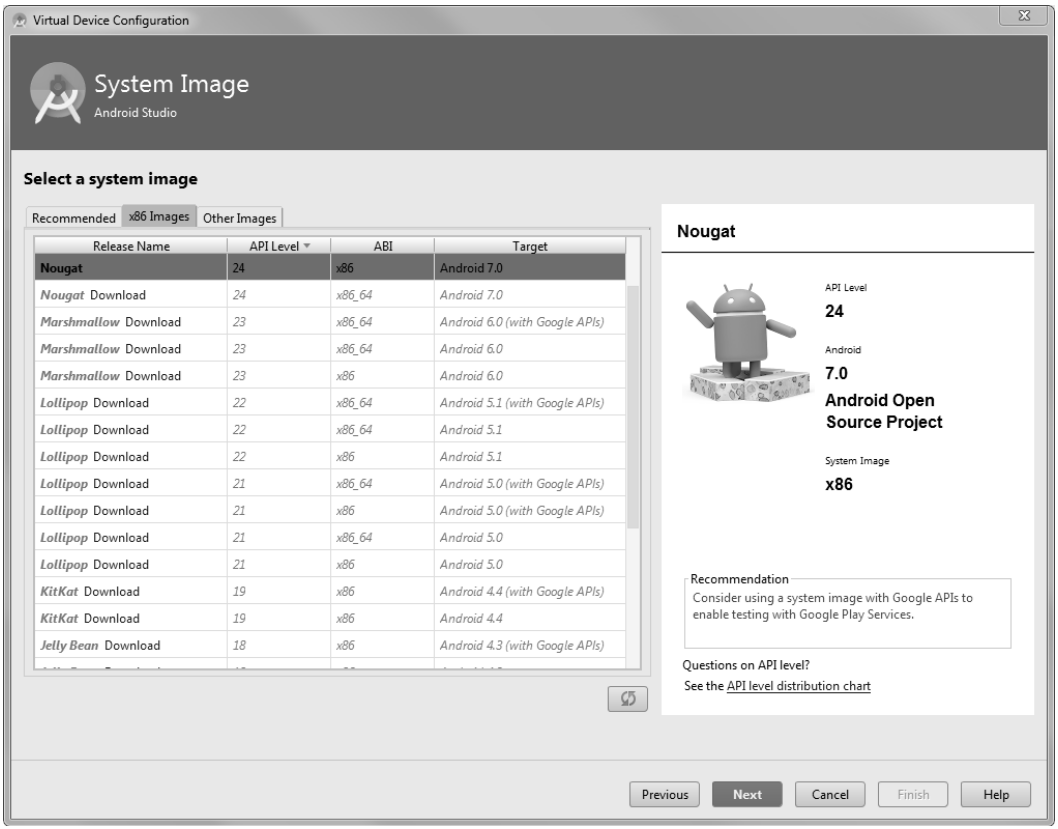



Rysunek 1.15. Wybieranie urządzenia wirtualnego

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

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

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



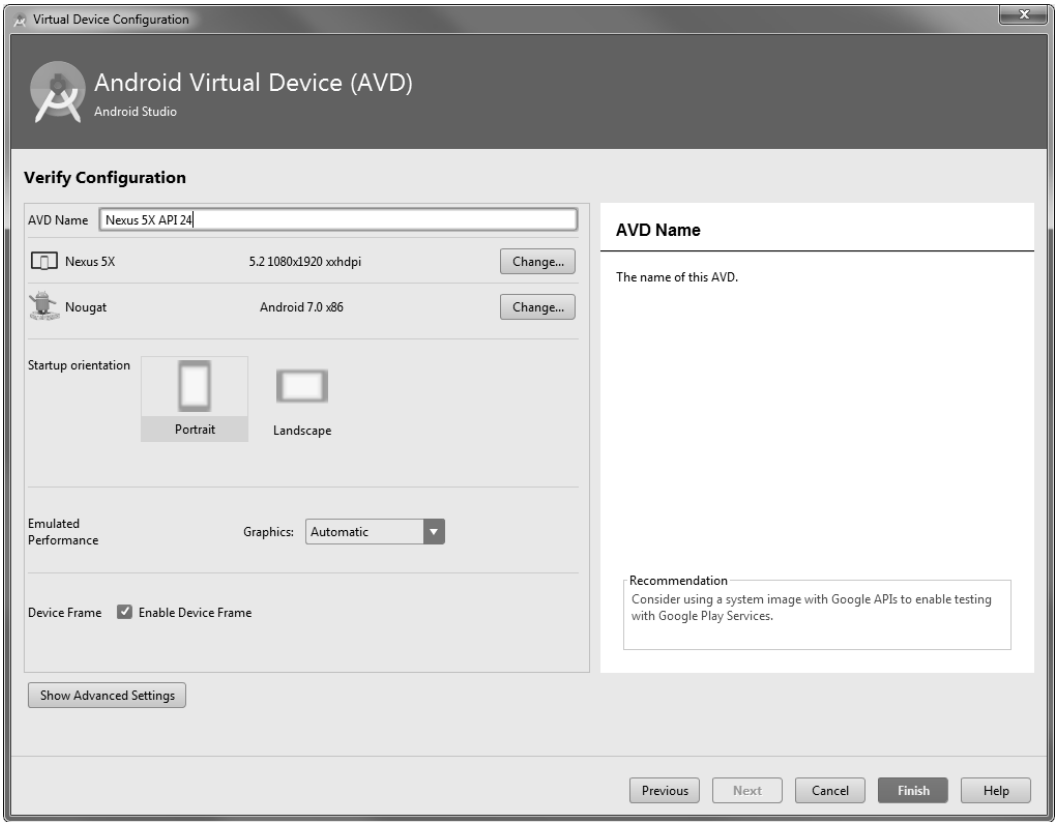

Rysunek 1.16. Wybieranie obrazu systemu dla emulatora

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

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

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

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




Rysunek 1.17. Ustawianie właściwości emulatora

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

Rysunek 1.18. Przykładowy wyjątek NullPointerException w wierszu 21

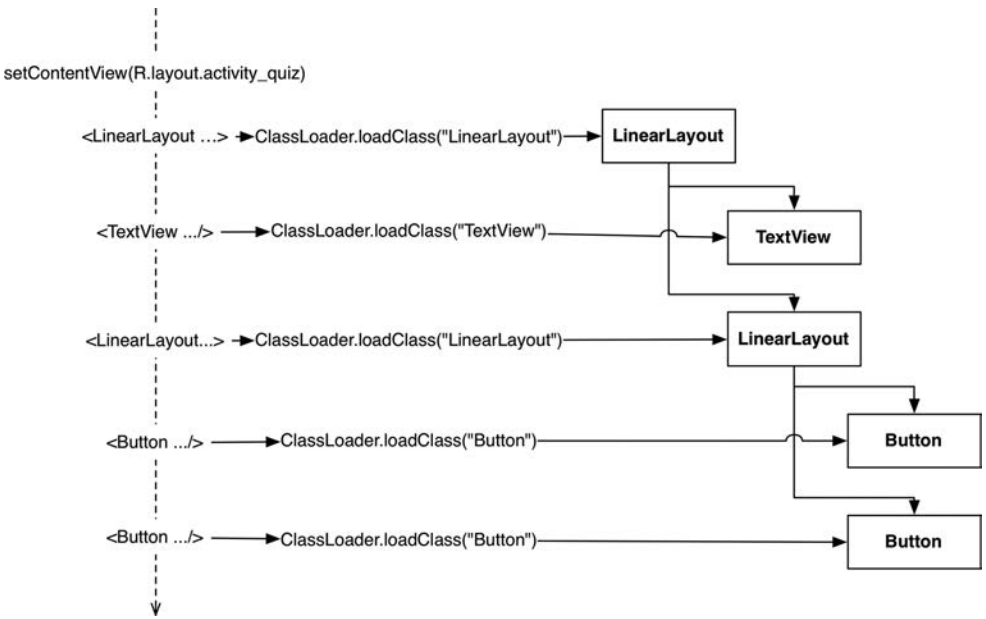
Aplikację możesz zatrzymać, naciskając przycisk *Back* (wstecz) emulatora. Przycisk *Back* ma kształt trójkąta skierowanego w lewo, a w starszych wersjach systemu Android kształt zawracającej w lewo strzałki. Po zatrzymaniu aplikacji wprowadź niezbędne modyfikacje, ponownie spróbuj uruchomić aplikację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

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

# Dla dociekliwych: proces budowania aplikacji

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

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




Rysunek 1.19. Rozwijanie układu activity_quiz.xml

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

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

## Narzędzia wspomagające budowanie aplikacji

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

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

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

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

```
$ ./gradlew tasks
```

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

```
> gradlew.bat tasks
```

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

```
$ ./gradlew installDebug
```

W systemie Windows takie polecenie będzie wyglądało następująco:

```
> gradlew.bat installDebug
```

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

# Wyzwania

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

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

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

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

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

# Wyzwanie: dostosowywanie komunikatów toast do własnych potrzeb

W tym ćwiczeniu będziesz musiał zmodyfikować sposób wyświetlania komunikatów *toast* tak, aby wyświetlały się w górnej części okna zamiast w dolnej. Aby to zrobić, powinieneś użyć metody `setGravity` klasy `Toast`. Jako wartości parametru `gravity` powinieneś użyć stałej `Gravity.TOP`. Więcej szczegółowych informacji na temat metody `setGravity` znajdziesz w dokumentacji na stronie *https://developer.android.com/reference/android/widget/Toast.html - setGravity(int, int, int)*.

# Skorowidz

## A

## B



## C



## D

## H



## I



## J



## K

## N



## O

## P

## R

## S



## Ś



## T

## U

## W

## Android — zaprogramuj przyszłość!

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

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

**W tej książce:**

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

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


**Bill Phillips**

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

**Chris Stewart**

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

**Kristin Marsicano**

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



Helion

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Informatyka w najlepszym wydaniu

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

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

Big Nerd Ranch

ISBN 978-83-283-3636-0

9 788328 336360

cena: 99,00 zł