

Marcin Płonkowski



Android Studio

Tworzenie aplikacji mobilnych

Aplikacje do zadań specjalnych...
pisz tylko w **Android Studio!**

Helion 

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

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

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

Opieka redakcyjna: Ewelina Burska

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

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

Drogi Czytelniku!

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

ISBN: 978-83-283-3593-6

Copyright © Helion 2018

Printed in Poland.

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

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

Spis treści

Wstęp	7
Dlaczego Android Studio?	7
Rozdział 1. Poznajemy Android Studio	9
1.1. Instalacja środowiska Android Studio	9
1.2. Pierwsze uruchomienie i pierwszy projekt środowiska Android Studio	15
1.3. Konfiguracja emulatora AVD	20
1.4. Konfiguracja urządzenia fizycznego	26
1.5. Pierwsze uruchomienie aplikacji	32
1.6. Opis środowiska Android Studio	34
1.6.1. Podstawowe elementy okna głównego	35
1.6.2. Opis dostępnych okien narzędziowych	36
1.7. Opis okna edytora kodu	39
1.7.1. Podstawowe elementy edytora kodu	39
1.7.2. Lokalizacja błędów w kodzie programu	42
1.7.3. Funkcja lupy	42
1.7.4. Podział okna edycyjnego	42
1.7.5. Uzupełnianie kodu	44
1.7.6. Uzupełnianie składni	45
1.7.7. Generowanie kodu	46
1.7.8. Informacje o parametrach metod	47
1.7.9. Podgląd dokumentacji	47
1.7.10. Zwijanie i rozwijanie kodu	48
1.7.11. Formatowanie kodu	49

Rozdział 2. Podstawy tworzenia interfejsu użytkownika51

2.1. Edytor układów	51
2.2. Praca z układami	54
2.2.1. Układ LinearLayout (Horizontal)	55
2.2.2. Układ LinearLayout (Vertical)	63
2.2.3. Układ RelativeLayout	70
2.2.4. Układ TableLayout	76
2.2.5. Układ GridLayout	80
2.2.6. Układ FrameLayout	83
2.2.7. Zagnieżdżanie układów	84
2.2.8. Komponent ScrollView	86
2.3. Komponenty interfejsu użytkownika	87
2.3.1. Kategoria Widgets	88

Rozdział 3. Style i tematy 101

3.1. Style	101
3.2. Dziedziczenie stylów	105
3.3. Tematy	107
3.4. Obrazy typu Nine-patch	109
3.5. Selektory	112
3.6. Kształty	115

Rozdział 4. Aktywności i fragmenty 123

4.1. Aktywności	123
4.2. Okna dialogowe	130
4.2.1. Okna dialogowe postępu	130
4.2.2. Informacyjne okna dialogowe	138
4.3. Intencje	144
4.3.1. Intencje jawne	146
4.3.2. Intencje domniemane	157
4.4. Fragmenty	171

Rozdział 5. Powiadomienia 183

5.1. Powiadomienia proste	183
5.2. Powiadomienia zawierające akcje	188
5.3. Powiadomienia z rozszerzonym widokiem	190
5.4. Priorytety powiadomień	193
5.5. Grupowanie powiadomień	197
5.6. Powiadomienia z odpowiedzią	200

Rozdział 6. Praca z komponentami interfejsu użytkownika207

6.1. Zachowywanie stanu aktywności	207
6.2. Obsługa zmian konfiguracji	215
6.2.1. Zmiana orientacji ekranu	215
6.2.2. Pozostałe zmiany konfiguracyjne	220
6.3. Oprogramowywanie kontrolki interfejsu użytkownika	221
6.3.1. Kontrolka AutoCompleteTextView	221
6.3.2. Kontrolka ListView	225
6.3.3. Kontrolka Spinner	237
6.3.4. Klasy ListActivity i ListFragment	241
6.3.5. Klasa DialogFragment	251
6.3.6. Klasy TimePicker i DatePicker	264
6.3.7. Klasa SeekBar	275
6.3.8. Klasa RatingBar	279

Rozdział 7. Obrazy i animacje283

7.1. Praca z obrazami	283
7.1.1. Klasa GridView	283
7.1.2. Klasa HorizontalScrollView	290
7.1.3. Klasa ImageSwitcher	297
7.1.4. Obrazy w kontrolce ListView	302
7.2. Rysowanie kształtów	308
7.2.1. Klasa Canvas	308
7.2.2. Rysowanie wewnątrz kontrolki	318
7.2.3. Klasa Shader	324
7.3. Animacje	334
7.3.1. Animacja pokłatkowa	334
7.3.2. Animowanie kontrolki	339

Rozdział 8. Zapisywanie i odczytywanie danych351

8.1. Zapisywanie wartości typów prostych	351
8.2. Zapisywanie danych do pliku	356
8.3. Dostęp do listy kontaktów	361
8.4. Korzystanie z bazy danych	371

Dodatek A Usprawnienia w pisaniu kodu377

A.1. Skróty klawiaturowe	377
A.2. Szablony	385

Skorowidz393

Rozdział 3.

Style i tematy

W tym rozdziale zostaną opisane style (ang. *styles*) i tematy (ang. *themes*), dzięki którym nasze aplikacje będą wyglądały profesjonalnie i przyjaźnie dla użytkowników. Musimy pamiętać, że strona wizualna aplikacji jest równie ważna jak kod, który odpowiada za jej działanie. Dlatego rozdział ten stanowi ważny krok w procesie tworzenia aplikacji dla urządzeń z systemem Android.

3.1. Style

Idea stylów w systemie Android jest bardzo podobna do stylów CSS wykorzystywanych w tworzeniu stron internetowych. Definiujemy tutaj jakiś zbiór atrybutów, z którego będą mogły korzystać używane przez nas komponenty. Zanim jednak przejdziemy do omawiania stylów, przeanalizujemy prosty przykład, w którym nasze komponenty ich nie wykorzystują.

Załóżmy, że chcemy umieścić pole tekstowe (TextView) wraz z atrybutami pokazanymi na listingu 3.1.

LISTING 3.1. Pole tekstowe wraz z atrybutami opisującymi jego wygląd

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="tekst1"
    android:textColor="#434230"
    android:background="#c6ecc6"
    android:typeface="serif"
    android:textSize="15sp"
    android:textStyle="bold"
    android:padding="10dp"
/>
```

Możemy to zrobić w sposób przedstawiony na listingu 3.1 i wszystko będzie działać oraz wyglądać poprawnie. Jeśli jednak będziemy chcieli umieścić inne kontrolki wykorzystujące takie same atrybuty, będziemy musieli je powielić w każdej z kontrolerek.

Wydaje się, że to nic trudnego, bo wystarczy skopiować ten sam fragment i powklejać go do pozostałych kontroltek. Problem pojawia się w sytuacji, kiedy chcemy zmienić jeden z atrybutów, np. kolor tekstu (`android:textColor`). W takim wypadku będziemy musieli przeprowadzić tę samą zmianę we wszystkich kontrolkach. Jak się niedługo dowiemy, aplikacja może składać się z kilku plików XML, więc przeglądanie ich wszystkich i szukanie kontroltek, które wymagają modyfikacji, okaże się bardzo czasochłonnym zadaniem.

Stąd też w systemie Android mamy możliwość definiowania stylów. Dzięki nim w jednym miejscu określimy wszystkie wymagane atrybuty, z których będą korzystały nasze kontrolki. Jeżeli zaistnieje potrzeba dokonania jakiejś zmiany, to wystarczy wprowadzić ją w jednym pliku (tam, gdzie zdefiniowaliśmy dany styl). Kod kontrolki `TextView` wykorzystującej style będzie wyglądał następująco (listing 3.2):

LISTING 3.2. Kontrolka `TextView` wykorzystująca style

```
<TextView
    android:id="@+id/textView1"
    style="@style/MojStyl"
    android:text="tekst"
/>
```

Widzimy, że wpisany tekst jest o wiele krótszy, co znacznie ułatwia pracę z kodem. Wszystkie pozostałe atrybuty są zaś zdefiniowane w stylu o nazwie *MojStyl* (przez atrybut `style`). Pokażemy teraz, gdzie możemy tworzyć własne style.

W oknie narzędziowym projektu przechodzimy do pliku *styles.xml* w lokalizacji *app/res/values/*. Po otwarciu pliku zobaczymy jeden automatycznie zdefiniowany styl (listing 3.3).

LISTING 3.3. Automatycznie zdefiniowany styl `AppTheme` w pliku *styles.xml*

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

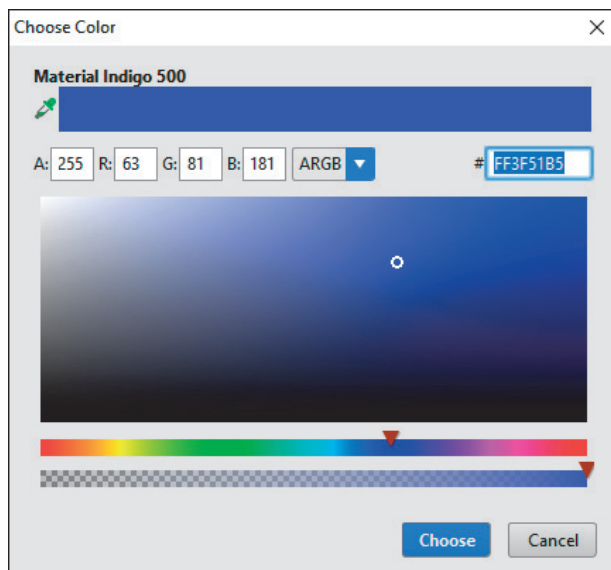
Widzimy tu trzy pozycje: `colorPrimary` o wartości `@color/colorPrimary`, `colorPrimaryDark` o wartości `@color/colorPrimaryDark` oraz `colorAccent` o wartości `@color/colorAccent`. Wszystkie kolory są zdefiniowane w pliku *colors.xml* w lokalizacji *app/res/values/*. Jeżeli zajrzemy do tego pliku, to zobaczymy nasze trzy kolory (listing 3.4).

LISTING 3.4. Wartości kolorów w pliku colors.xml

```
<resources>
  <color name="colorPrimary">#3F51B5</color>
  <color name="colorPrimaryDark">#303F9F</color>
  <color name="colorAccent">#FF4081</color>
</resources>
```

Wartości są podawane szesnastkowo, a reprezentowany przez nie kolor możemy zobaczyć po lewej stronie na pasku (trzy kolorowe kwadraciki). Jeżeli klikniemy jeden z tych kwadracików, to pokaże nam się okno dialogowe o nazwie *Choose Color* pozwalające na wybór koloru (rysunek 3.1).

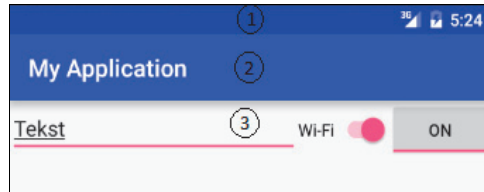
RYСУNEK 3.1.
Okno dialogowe
pozwalające na
wybór koloru



Kolor możemy wybrać przez kliknięcie myszką pożądanego obszaru. Za pomocą pierwszego suwaka wybieramy odcień, który nas interesuje, a za pomocą drugiego przezroczystość koloru. Dodatkowo możemy podać wartości w formie liczbowej (od 0 do 255), wpisując je w pola oznaczone literami *A* (przezroczystość), *R* (nasycenie koloru czerwonego), *G* (nasycenie koloru zielonego), *B* (nasycenie koloru niebieskiego). Dla wartości *A* liczba 0 oznacza pełną przezroczystość, a 255 całkowitą nieprzezroczystość. Dla wartości *R*, *G*, *B* liczba 0 oznacza brak danej składowej, a 255 pełne nasycenie danej składowej.

Wróćmy jeszcze do atrybutów *colorPrimary*, *colorPrimaryDark* i *colorAccent* zdefiniowanych w stylu *AppTheme*, który jest głównym tematem (ang. *theme*) naszej aplikacji. Na rysunku 3.2 zostały oznaczone trzy miejsca głównego ekranu naszej aplikacji, w których możemy zobaczyć zastosowanie powyższych kolorów.

RYСУNEK 3.2.
Podstawowe
kolory tematu
AppTheme



Na powyższym rysunku cyfrą 1 oznaczono pasek stanu (ang. *status bar*), który przyjmie wartość koloru zdefiniowaną w atrybucie `colorPrimaryDark`. Cyfrą 2 oznaczono pasek narzędziowy (ang. *toolbar*), który przyjmie wartość koloru zdefiniowaną w atrybucie `colorPrimary`. W ostatnim rzędzie oznaczonym cyfrą 3 mamy przykład trzech komponentów, które wykorzystują wartość atrybutu `colorAccent`. Pierwszy z nich to `EditText`, który dodaje na dole różową linię oznaczającą, że właśnie to pole edycyjne jest aktywne. Drugi komponent to przełącznik (`Switch`), w momencie włączenia zmienia on kolor kropki na różowy. Ostatni z nich to inny rodzaj przełącznika (`ToggleButton`), który w stanie `ON` będzie miał u dołu różową linię. Oczywiście te oznaczenia są różowe ze względu na ustawioną wartość atrybutu `colorAccent`, którą jeżeli będziemy chcieli, możemy zmienić (np. w sposób pokazany na rysunku 3.1).

Skoro już rozumiemy zawartość pliku *styles.xml*, to możemy przejść do zdefiniowania naszych własnych stylów. Wróćmy do listingów 3.1 oraz 3.2. W tym pierwszym mieliśmy przykład użycia komponentu `TextView` wraz z kilkoma jego atrybutami. Jednakże powiedzieliśmy już, że takie rozwiązanie nie sprawdzi się dobrze, jeśli użyjemy innych komponentów korzystających z tych atrybutów. Aby zagwarantować spójny wygląd naszej aplikacji, warto utworzyć odpowiednie style i wskazać je wybranym komponentom (listing 3.2).

Przejdźmy więc do pliku *styles.xml* i zdefiniujmy nasz styl o nazwie *MojStyl* (listing 3.5).

LISTING 3.5. Definicja własnego stylu o nazwie *MojStyl*

```
<style name="MojStyl">
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#434230</item>
    <item name="android:background">#c6ecc6</item>
    <item name="android:typeface">serif</item>
    <item name="android:textSize">15sp</item>
    <item name="android:textStyle">bold</item>
    <item name="android:padding">10dp</item>
</style>
```

Styl przedstawiony na listingu 3.5 umieszczamy w pliku *styles.xml* w znaczniku `<resources>`. Od tej pory będzie on mógł być wykorzystywany w kilku komponentach, a wszelkie modyfikacje będą wprowadzane tylko w jednym miejscu (w pliku *styles.xml*). Na koniec wyjaśnimy jeszcze znaczenie nowych atrybutów, które pojawiły się na listingu 3.5:

- `android:textColor` — kolor tekstu,
- `android:background` — kolor tła,

- `android:typeface` — krój pisma, z czego do wyboru mamy:
 - `normal` — zwykłą czcionkę,
 - `serif` — czcionkę szeryfową (z ozdobnikami w postaci poprzecznie lub ukośnie zakończonych głównych kresek liter danego kroju pisma),
 - `sans` — czcionkę bezszeryfową (bez ozdobników),
 - `monospace` — czcionkę o stałej szerokości znaków.
- `android:textStyle` — odmiana pisma, z czego do wyboru mamy:
 - `normal` — tekst zwykły,
 - `bold` — tekst pogrubiony,
 - `italic` — tekst pochylony (kursywę).

3.2. Dziedziczenie stylów

Do tej pory poznaliśmy proste style, które tworzą zamknięty zbiór zdefiniowanych atrybutów. Natomiast w przypadku bardziej skomplikowanych aplikacji przy definiowaniu nowych stylów będziemy wykorzystywać te zdefiniowane wcześniej. Nazywa się to dziedziczeniem stylów. Sam mechanizm dziedziczenia jest dobrze znany wszystkim programistom.

Tworząc nowy styl, nie musimy od nowa definiować wszystkich atrybutów, może skorzystać ze stylu zdefiniowanego wcześniej i dodać do niego tylko nowe elementy. Dzięki temu unikniemy niepotrzebnego powtarzania (często obszernych) fragmentów kodu. Dodatkowo będziemy mieli spójną i przejrzystą hierarchię stylów, która ułatwi nam pisanie kodu i wprowadzanie wszelkich modyfikacji. Style, które będziemy tworzyć, mogą dziedziczyć po naszych własnych stylach (wcześniej już zdefiniowanych) lub też po stylach zdefiniowanych w pakiecie Android SDK.

Najpierw prześledzimy sytuację, w której w dziedziczeniu wykorzystamy nasz własny styl. Spróbujmy zatem dodać nowy styl o nazwie `MojStyl.NiebieskaKursywa`. Zaadaptuje on wszystkie atrybuty ze stylu `MojStyl` oraz nadpisze dwa z nich. Pierwszy to kolor tekstu (atrybut `android:textColor`), który zmienimy na niebieski, drugi to odmiana pisma (atrybut `android:textStyle`), którą zmienimy na kursywę. Zerknijmy na listing 3.6.

LISTING 3.6. Dziedziczenie własnych stylów

```
<style name="MojStyl.NiebieskaKursywa">
  <item name="android:textColor">#0000ff</item>
  <item name="android:textStyle">italic</item>
</style>
```

Najpierw zwróćmy uwagę na nazwę naszego stylu (atrybut `name`), która składa się z dwóch części oddzielonych kropką. Pierwsza część `"MojStyl"` oznacza nazwę stylu, po którym dziedziczymy, druga `"NiebieskaKursywa"` nazwę naszego nowego stylu. W tym przypadku

nasz nowy styl odziedziczył wszystkie atrybuty po stylu bazowym i nadpisał dwa atrybuty (`android:textColor` i `android:textStyle`), nadając im nowe wartości.

Oczywiście proces dziedziczenia możemy kontynuować. Spróbujmy utworzyć kolejny styl o nazwie `MojStyl.NiebieskaKursywa.Maly`, który będzie dziedziczył po naszych wcześniejszych stylach. Zmienimy w nim rozmiar tekstu na 10sp (listing 3.7).

LISTING 3.7. Dziedziczenie własnych stylów (notacja z kropkami)

```
<style name="MojStyl.NiebieskaKursywa.Maly">
    <item name="android:textSize">10sp</item>
</style>
```

Tym razem widzimy nazwę składającą się z trzech członów oddzielonych kropkami, która pokazuje nam hierarchię dziedziczenia w obrębie naszych stylów.

Jak teraz zastosować ten styl do kontrolki `TextView`? Sprawa jest prosta, gdyż wykorzystujemy tę samą konwencję co w przypadku stylów bez dziedziczenia. Musimy jednak pamiętać, że nazwa stylu zdefiniowanego w listingu 3.7 jest trójczłonowa (listing 3.8).

LISTING 3.8. Wykorzystanie stylu w kontrolce `TextView`

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@style/MojStyl.NiebieskaKursywa.Maly"
/>
```

Zamiast notacji z kropką możemy użyć do dziedziczenia stylów atrybutu `parent`. Listing 3.9 pokazuje dziedziczenie z wykorzystaniem tego atrybutu.

LISTING 3.9. Dziedziczenie własnych stylów za pomocą atrybutu `parent`

```
<style name="NiebieskaKursywa" parent="MojStyl">
    <item name="android:textColor">#0000ff</item>
    <item name="android:textStyle">italic</item>
</style>

<style name="Maly" parent="NiebieskaKursywa">
    <item name="android:textSize">10sp</item>
</style>
```

W przypadku dziedziczenia własnych stylów możemy stosować zarówno notację z kropką, jak i atrybut `parent`. Natomiast na dziedziczenie po stylach wbudowanych w platformę Android SDK pozwala tylko atrybut `parent` (notacja z kropką nie będzie działać).

Założmy, że chcemy utworzyć czerwony tekst wykorzystujący atrybuty stylu `@android:style/TextAppearance`, który jest odpowiedzialny za domyślny wygląd tekstu. W tym celu użyjemy atrybutu `parent` oraz atrybutu `android:textColor` (listing 3.10).

LISTING 3.10. Dziedziczenie po wbudowanych stylach

```
<style name="Czerwony" parent="@android:style/TextAppearance">
    <item name="android:textColor">#ff0000</item>
</style>
```

Kiedy poznaliśmy już metody tworzenia stylów oraz mechanizm dziedziczenia, pojawia się pytanie, skąd mamy wiedzieć, jakie atrybuty możemy w danym stylu zdefiniować? Żeby rozstrzygnąć ten problem, musimy najpierw odpowiedzieć sobie na pytanie, do której kontrolki (bądź zestawu kontrolki) będziemy stosować dany styl. Jeżeli będzie to na przykład kontrolka `TextView`, to wystarczy w dokumentacji technicznej sprawdzić, jakie atrybuty są dla niej dostępne. Jak już wspominaliśmy, dokumentacja techniczna systemu Android znajduje się pod adresem <https://developer.android.com>, a widoczna w prawym górnym rogu wyszukiwarka pozwoli nam odnaleźć interesujące nas zagadnienia. Jeżeli jednak zdarzy nam się, że w definicji naszego stylu użyliśmy atrybutu, którego dana kontrolka nie obsługuje, to ten atrybut zostanie po prostu zignorowany.

Istnieją jednak takie atrybuty stylów, które nie są obsługiwane przez żadną kontrolkę i mogą być stosowane tylko do tematów (ang. *themes*). Właściwości te dotyczą całego okna aplikacji, a nie żadnej konkretnej kontrolki. Przykładem tego typu właściwości są `m.in.:` ukrywanie tytułu aplikacji, ukrywanie paska stanu i zmiana tła okna.

Istnieje prosty sposób, w jaki możemy odróżnić style stosowane do tematów od stylów przeznaczonych dla kontrolki. Te pierwsze są poprzedzone prefiksem `window.` Przykładem mogą tu być takie atrybuty, jak `windowNoTitle` (określa, czy tytuł okna będzie wyświetlany), `windowTitleSize` (określa wysokość paska tytułowego okna) czy `windowBackground` (określa tło okna). Ale o tematach powiemy więcej w następnym podrozdziale.

3.3. Tematy

Temat (ang. *theme*) jest stylem stosownym do danej aktywności (ang. *Activity*) lub aplikacji. Aktywności to jeden z kluczowych elementów w aplikacjach systemu Android i powiemy o nich więcej w kolejnych rozdziałach. Teraz wystarczy nam utożsamienie aktywności z oknem aplikacji.

Sposób, w jaki będziemy definiować tematy, jest identyczny jak w przypadku stylów, które stosowaliśmy do naszych kontrolki. Przejdźmy zatem do pliku `styles.xml` i dodajmy dwa nowe atrybuty (listing 3.11).

LISTING 3.11. Definiowanie atrybutów w temacie `AppTheme`

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
```

```
<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
<item name="android:windowFullscreen">true</item>
<item name="android:colorBackground">@color/Zielony</item>
</style>
</resources>
```

Zwróćmy uwagę na nazwę naszego tematu oraz na temat bazowy. Nazwa to `AppTheme`, a temat bazowy, po którym dziedziczymy, to `Theme.AppCompat.Light.DarkActionBar`. Do jego podstawowej definicji, którą możemy zobaczyć na listingu 3.3, dodaliśmy dwa atrybuty. Pierwszy to `android:windowFullscreen` o wartości `true`, która pozwala na ustawienie aplikacji w trybie pełnoekranowym (aplikacja zajmie cały dostępny obszar ekranu). Drugi atrybut to `android:colorBackground`, który już poznaliśmy, ustawia on tło naszej aplikacji. Oczywiście w pliku *colors.xml* należy wcześniej zdefiniować wartość koloru o nazwie `Zielony`. Plik *colors.xml* i zawarte w nim predefiniowane kolory widzieliśmy już na listingu 3.4. Teraz wystarczy do niego dodać linię z kolorem tła (listing 3.12).

LISTING 3.12. Definicja nowego koloru w pliku *colors.xml*

```
<resources>
  <color name="colorPrimary">#3F51B5</color>
  <color name="colorPrimaryDark">#303F9F</color>
  <color name="colorAccent">#FF4081</color>
  <color name="Zielony">#00ff00</color>    <!-- Kolor zielony -->
</resources>
```

Teraz zobaczmy, gdzie w naszym programie jest ustawiony ten temat. W tym celu otworzymy plik *AndroidManifest.xml* (listing 3.13) w katalogu *app/manifests*.

LISTING 3.13. Zawartość pliku *AndroidManifest.xml*

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.marcin.myapplication">
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">    <!-- Temat naszej aplikacji -->
    <activity android:name=".MainActivity"
      android:theme="@style/AppTheme">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Zerknijmy na atrybut `android:theme` o wartości `"@style/AppTheme"`. Widzimy, że został tu wykorzystany nasz temat `AppTheme`. Dzięki temu wszystkie aktywności w aplikacji będą stosowały zdefiniowane w nim atrybuty. Poniżej definicji tematu widzimy znacznik `<activity>`, to tutaj określono główną aktywność aplikacji. Na razie jest tylko jedna, ale może być ich więcej. Jeżeli chcielibyśmy, aby dany temat był stosowany tylko do wybranej aktywności (a nie całej aplikacji), to w znaczniku `<activity>` definiujemy atrybut `android:theme` z odpowiednim tematem.

O pozostałych elementach przedstawionych w pliku *AndroidManifest.xml* opowiemy w dalszej części książki.

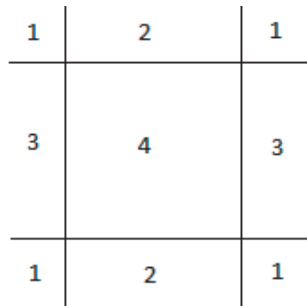
3.4. Obrazy typu Nine-patch

Obrazy typu *nine-patch* (ang. *dziewięć łatek*) są specjalnymi, rozciągalnymi bitmapami, które system Android potrafi w ładny sposób wyświetlać. Urządzenia pracujące pod kontrolą systemu Android mają różne rozdzielczości, rozmiary oraz gęstości pikseli. Dlatego uruchomione na nich aplikacje będą wyskalowane w różny sposób, co może powodować rozciąganie obrazów i znaczne pogorszenie ich wyglądu.

Z tego powodu powstała technologia obsługi obrazów znana jako *nine-patch*. Polega ona na tym, że obraz jest dzielony na dziewięć części (może ich też być więcej), tak jak pokazano na rysunku 3.3.

RYСУNEK 3.3.

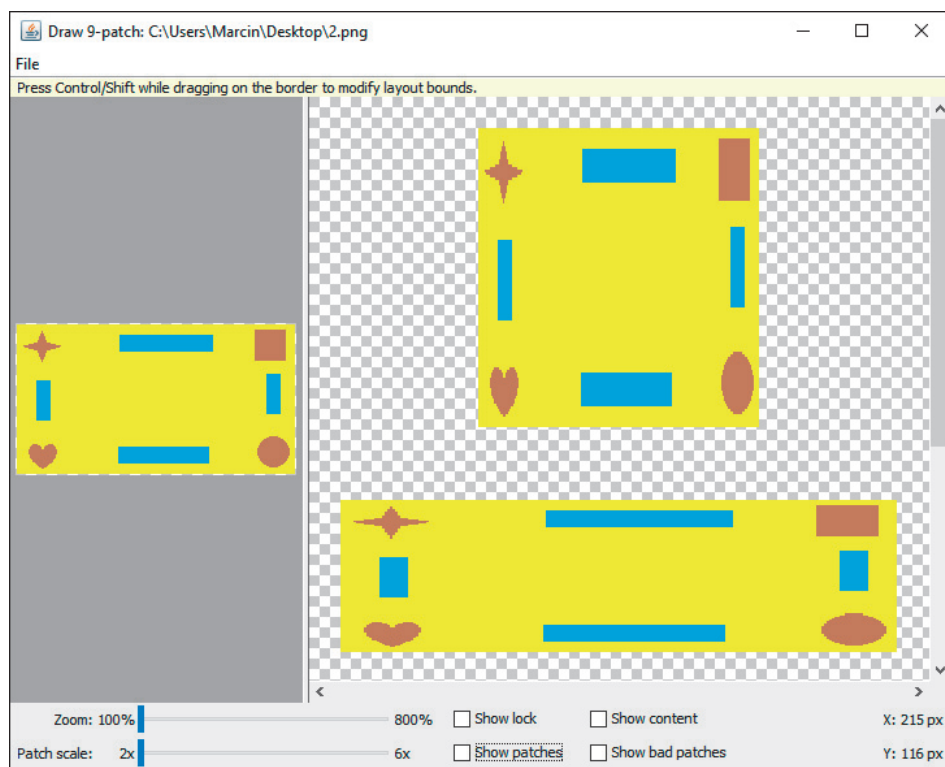
Obszary obrazu
typu nine-patch



Obszary narożne, oznaczone cyfrą 1, nie są skalowane, co oznacza, że zawsze wyglądają tak samo. Obszary oznaczone cyfrą 2 mogą być skalowane tylko w poziomie, a obszary oznaczone cyfrą 3 tylko w pionie. Obszar środkowy, oznaczony cyfrą 4, jest skalowany zarówno w pionie, jak i poziomie. Podsumowując, w obszarach oznaczonych cyfrą 1 umieszczamy elementy, które nie powinny być skalowane, żeby zawsze wyglądać tak samo (np. zaokrąglone narożniki przycisku lub małe ikony). Natomiast w obszarze 4 możemy umieszczać elementy, które wyglądają dobrze w każdym rozmiarze (np. jednolity kolor). W obszarach oznaczonych cyframi 2 i 3 umieszczamy takie elementy, które dobrze się skalują w jednym wymiarze (poziomo lub pionowo).

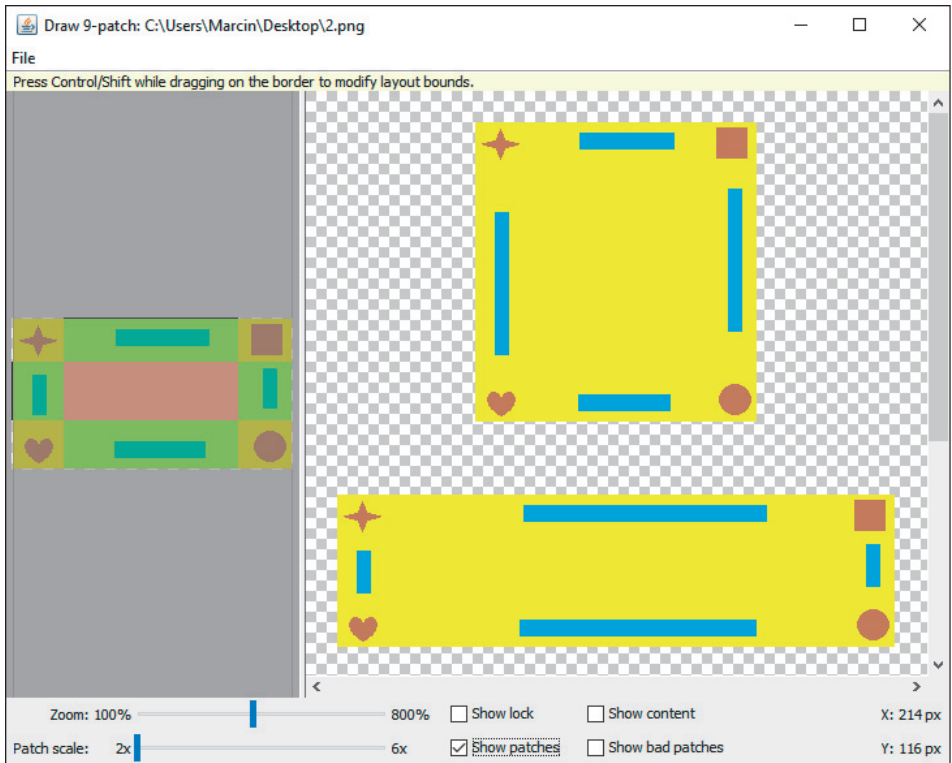
Technicznie plik obrazu, który wykorzystuje technologię *nine-patch*, różni się od zwykłego obrazu tylko dwoma rzeczami. Pierwsza to nazwa pliku, która ma postać *<nazwa obrazka>.9.png* (zwykła nazwa pliku to *<nazwa obrazka>.png*), druga to ramka otaczająca nasz obraz o szerokości 1 piksela.

Obraz tego typu z łatwością przygotujemy w narzędziu dostarczonym wraz z pakietem Android SDK. Aplikacja, którą wykorzystamy, to *Draw 9-patch*. Uruchamiamy ją przez skrypt *draw9patch.bat* umieszczony w folderze *<lokalizacja Android SDK>/tools/*. Po czym za pomocą menu *File* otwieramy obraz, który chcemy przygotować w technologii *nine-patch* (rysunek 3.4).



Rysunek 3.4. Aplikacja Draw 9-patch

W lewym oknie programu widzimy nasz oryginalny plik, a po prawej stronie — dwa obrazy. Pierwszy rozciągnięty w pionie, a drugi w poziomie. Jeżeli przesuniemy pasek przewijania, to zobaczymy jeszcze trzeci obraz rozciągnięty jednocześnie w pionie i poziomie. Zauważmy, że po rozciągnięciu koło stanie się elipsą, a kwadrat prostokątem. My natomiast chcemy, aby figury narożne zawsze zachowywały swój oryginalny kształt. W tym celu w oknie po lewej zaznaczamy dwie linie: po lewej stronie i u góry obrazu. Zaznaczamy też pole wyboru *Show patches*. Efekt powinien być taki jak na rysunku 3.5.



Rysunek 3.5. Przygotowanie obrazu w technologii nine-patch

Zwróćmy uwagę na czarne linie po lewej stronie i u góry. Zaznaczyliśmy je, tworząc dziewięć obszarów. Zobaczmy, że elementy narożne po przeskalowaniu obrazu pozostają niezmienione. Elementy na brzegach (prostokąty) zmieniają kształt tylko w jednym kierunku (co dla prostokątów nie ma większego znaczenia). Natomiast środkowy obszar jest wypełniony jednolitym kolorem, co oznacza, że jego skalowanie nie jest żadnym problemem.

Tak przygotowany obraz zapisujemy w naszym projekcie, w lokalizacji *<lokalizacja naszego projektu>/app/src/main/res/drawable*. Szybszym sposobem na otwarcie katalogu *drawable* w eksploratorze Windows jest skorzystanie z menu podręcznego katalogu *drawable* (w naszym projekcie) i wybranie opcji *Show in Explorer*.

Teraz, aby użyć tak przygotowanego obrazu jako tła, wystarczy umieścić kontrolkę android: `background` z ustawionym na niego atrybutem (listing 3.14).

LISTING 3.14. Wykorzystanie obrazu typu nine-patch

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:text="Przycisk"
        android:id="@+id/button"
        android:background="@drawable/obrazek" />
</LinearLayout>

```

Przypomnijmy jeszcze, że wystarczy podać tylko nazwę obrazu (bez rozszerzenia).

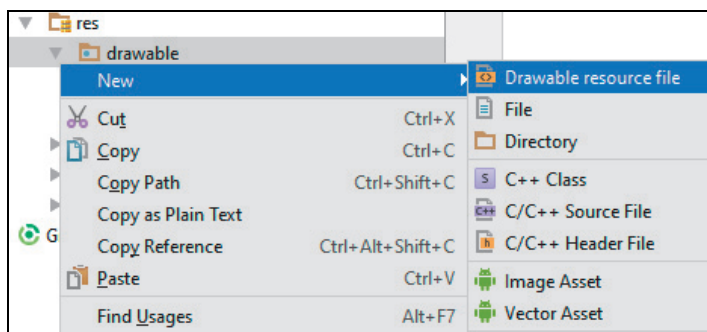
O czym warto wspomnieć, w programie *Draw 9-patch*, oprócz linii z lewej strony i u góry obrazu, możemy dodać linie po prawej stronie i u dołu. Zdefiniują nam one marginesy wewnętrzne, czyli takie, w obrębie których będzie prezentowana treść kontrolki (np. napis danego przycisku).

3.5. Selektory

Selektory (ang. *selectors*) to prosta, ale bardzo użyteczna konstrukcja pozwalająca dostosować wygląd komponentu w zależności od stanu, w jakim się znajduje. Rozważmy przykład przycisku (ang. *button*), który będzie inaczej wyglądał w zależności od tego, czy w danym momencie jest wciśnięty, niewciśnięty (w stanie normalnym) lub przejął skupienie (ang. *focus*).

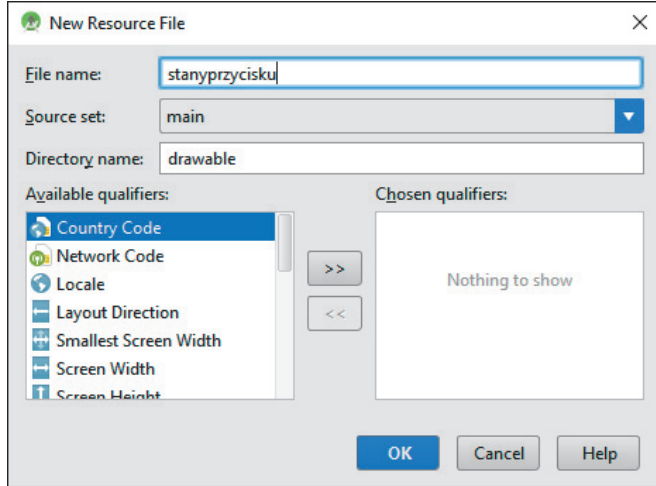
Najpierw utwórzmy trzy pliki graficzne, po jednym dla każdego stanu. Nazwiemy je odpowiednio *wcisniety.png*, *normalny.png* i *skupienie.png*, a następnie umieścimy w katalogu *res/drawable*. Teraz musimy utworzyć plik XML, w którym wykorzystamy nasz selektor do wyboru odpowiedniej grafiki w zależności od stanu przycisku. W tym celu klikamy prawym przyciskiem myszy folder *drawable*, po czym wybieramy opcję *New/Drawable resource file* (rysunek 3.6).

RYСУNEK 3.6.
Tworzenie nowego
pliku XML



Następnie zobaczymy okno *New Resource File*, w którym w polu *File name* wpisujemy wartość *stanyprzycisku* (rysunek 3.7).

Po wciśnięciu przycisku OK pojawi się otwarty plik o nazwie *stanyprzycisku.xml*. Powinniśmy zobaczyć szablon z wpisaniem już znacznikiem `<selector>`, który wypełnimy odpowiednimi stanami przycisku, tak jak pokazano na listingu 3.15.

RYSUNEK 3.7.Okno New
Resource File**LISTING 3.15.** Wykorzystanie znacznika <selector>

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
        android:drawable="@drawable/wcisniety" />
    <item android:state_focused="true"
        android:drawable="@drawable/skupienie" />
    <item android:drawable="@drawable/normalny" />
</selector>
```

Zwróćmy uwagę na dwa zaprezentowane stany. Pierwszy z nich to `android:state_pressed` oznaczający wciśnięcie przycisku. Drugi to `android:state_focused`, w którym kontrolka przejęła skupienie (ang. *focus*). Ostatni to stan domyślny, czyli normalny.

Teraz w naszym układzie umieścimy komponent `ImageButton`. To przycisk, który zamiast tekstu prezentuje użytkownikowi obraz. Wykorzystujemy do tego atrybut `android:src`, w którym podajemy nazwę obrazu lub nazwę pliku XML (z odpowiednio wypełnionym selektorem). Na listingu 3.16 pokazano kompletny kod pliku XML z naszym układem.

LISTING 3.16. Wykorzystanie selektora w komponencie `ImageButton`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/stanyprzycisku" <!-- Wskazanie na plik XML -->
    </ImageButton>
</LinearLayout>
```

Przetestujmy działanie naszej aplikacji. Powinniśmy zobaczyć, że w momencie wciśnięcia przycisku zmienia się obraz. W emulatorze stan przejścia skupienia przez przycisk uzyskamy za pomocą tabulatora (klawisz *Tab*).

Kolejny przykład z wykorzystaniem selektora będzie dotyczył zwykłego przycisku (Button) i zmiany koloru jego tła. W tym celu najpierw tworzymy plik XML (o nazwie *kolorowy-przycisk*), w którym definiujemy selektor, tak jak pokazano na listingu 3.17.

LISTING 3.17. Selektor pozwalający na zmianę kolorów przycisku

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_pressed="true">
    <color android:color="#FF0000" /> <!-- pressed -->
  </item>
  <item android:state_focused="true">
    <color android:color="#00FF00" /> <!-- focused -->
  </item>
  <item>
    <color android:color="#0000FF" /> <!-- default -->
  </item>
</selector>
```

W powyższym przykładzie wykorzystaliśmy znacznik `<color>` definiujący odpowiedni kolor. Oczywiście, jak już wspominaliśmy, lepszym sposobem jest zdefiniowanie kolorów w pliku *colors.xml*. Jednak w tej sytuacji nie chciałem już rozbudowywać przykładu, ponieważ w takiej formie jest wystarczająco zrozumiały. Pozostaje nam jeszcze dodanie przycisku i ustawienie w odpowiedni sposób atrybutu `android:background` (listing 3.18).

LISTING 3.18. Przycisk wykorzystujący selektor

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Przycisk"
    android:id="@+id/button"
    android:background="@drawable/kolorowyprzycisk" />

</LinearLayout>
```

Najistotniejsza jest ostatnia linia w definicji komponentu Button ustawiająca atrybut `android:background` na nasz plik XML (z definicją selektora).

W powyższych dwóch przykładach wykorzystaliśmy tylko dwa stany: `android:state_pressed` (wciśnięty) oraz `android:state_focused` (zaznaczony). Istnieją też inne, których możemy używać w taki sam sposób:

- `android:state_pressed` — kontrolka jest wciśnięta,
- `android:state_focused` — kontrolka przejęła skupienie (ang. *focus*),

- `android:state_selected` — zostaje wybrana pozycja z listy,
- `android:state_checkable` — kontrolka może zmienić swój stan (np. `CheckBox`),
- `android:state_checked` — kontrolka (np. `Checkbox`) zmieniła swój stan na zaznaczony,
- `android:state_enabled` — kontrolka jest włączona (ang. *enabled*),
- `android:state_activated` — kontrolka (np. `Checkbox`) lub jej rodzic zmienili swój stan na zaznaczony,
- `android:state_window_focused` — okno kontrolki przejęło skupienie (ang. *focus*).

3.6. Kształty

Kształt (ang. *shape*) to kolejny znacznik, który pozwoli nam za pomocą odpowiednich wpisów w pliku XML uatrakcyjnić wygląd naszej aplikacji. Dzięki znacznikowi `<shape>` uzyskamy wiele ciekawych efektów, ale żeby dobrze zrozumieć jego działanie, spójrzmy na jego definicję (listing 3.19):

LISTING 3.19. Definicja znacznika `<shape>`

```
<shape
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape=["rectangle" | "oval" | "line" | "ring"] >
  <corners
    android:radius="integer"
    android:topLeftRadius="integer"
    android:topRightRadius="integer"
    android:bottomLeftRadius="integer"
    android:bottomRightRadius="integer" />
  <gradient
    android:angle="integer"
    android:centerX="float"
    android:centerY="float"
    android:centerColor="integer"
    android:endColor="color"
    android:gradientRadius="integer"
    android:startColor="color"
    android:type=["linear" | "radial" | "sweep"]
    android:useLevel=["true" | "false"] />
  <padding
    android:left="integer"
    android:top="integer"
    android:right="integer"
    android:bottom="integer" />
  <size
    android:width="integer"
    android:height="integer" />
  <solid
    android:color="color" />
  <stroke
```

```
        android:width="integer"  
        android:color="color"  
        android:strokeWidth="integer"  
        android:strokeDash="integer" />  
</shape>
```

Opiszmy teraz po kolei wszystkie dostępne opcje. Pierwszy istotny element w znaczniku `<shape>` to atrybut `android:shape`. Jak widzimy, może on przyjąć cztery wartości:

- `rectangle` — kształt prostokątny (opcja domyślna) wypełniający cały dostępny obszar kontrolki;
- `oval` — kształt owalny dopasowujący się do rozmiarów kontrolki;
- `line` — linia biegnąca poziomo przez środek kontrolki; wartość ta wymaga znacznika `<stroke>` do zdefiniowania szerokości linii;
- `ring` — kształt pierścienia.

Wartość `ring` może wykorzystywać jeszcze dodatkowe opcje:

- `android:innerRadius` — promień wewnętrznej części pierścienia (środek pierścienia pokrywa się ze środkiem kontrolki).
- `android:innerRadiusRatio` — promień wewnętrznej części pierścienia wyrażony jako stosunek szerokości pierścienia. Na przykład jeżeli ustawimy `android:innerRadiusRatio="7"`, to promień będzie równy szerokości pierścienia podzielonej przez 7. Wartość ta jest nadpisywana przez `android:innerRadius`.
- `android:thickness` — grubość pierścienia.
- `android:thicknessRatio` — grubość pierścienia wyrażona jako stosunek szerokości pierścienia. Na przykład jeżeli `android:thicknessRatio="2"`, wtedy grubość pierścienia równa jest szerokości pierścienia podzielonej przez 2.
- `android:useLevel` — ustawiamy wartość `true`, jeżeli pierścień jest używany jako `LevelListDrawable` (zasób, który zarządza grupą obiektów graficznych). We wszystkich innych sytuacjach powinniśmy ustawić tę wartość jako `false`, inaczej nasz pierścień się nie pojawi!

Kolejny przydatny znacznik to `<corners>`. Dzięki niemu możemy tworzyć prostokąty z zaokrąglonymi rogami. Stąd też występuje tylko z kształtem o wartości `rectangle` (pol. prostokąt). Atrybuty, które możemy wraz z nim wykorzystać, to:

- `android:radius` — promień dla wszystkich narożników wartość ta jest nadpisywana przez szczegółowe ustawienia poszczególnych z nich;
- `android:topLeftRadius` — promień lewego górnego narożnika;
- `android:topRightRadius` — promień prawego górnego narożnika;
- `android:bottomLeftRadius` — promień lewego dolnego narożnika;
- `android:bottomRightRadius` — promień prawego dolnego narożnika.

Następny znacznik to `<gradient>`, pozwala on na realizację przejścia tonalnego pomiędzy (co najmniej) dwoma kolorami. Są z nim skojarzone następujące atrybuty:

- `android:angle` — kąt określający kierunek przejścia tonalnego. Wartość 0 (domyślna) oznacza kierunek od lewej do prawej, zaś 90 kierunek z dołu do góry, przy czym każda przypisana wartość musi być wielokrotnością liczby 45.
- `android:centerX` — względna współrzędna X wyznaczająca środek gradientu. Wartość ta musi mieścić się w przedziale [0, 1].
- `android:centerY` — względna współrzędna Y wyznaczająca środek gradientu. Wartość ta musi mieścić się w przedziale [0, 1].
- `android:centerColor` — opcjonalny kolor, który jest kolorem pośrednim pomiędzy początkowym a końcowym.
- `android:endColor` — końcowy kolor gradientu.
- `android:gradientRadius` — promień gradientu. Ma on zastosowanie tylko wtedy, gdy atrybut `android:type` ustawiony jest na wartość `radial`.
- `android:startColor` — początkowy kolor gradientu.
- `android:type` — typ gradientu z kilkoma dostępnymi opcjami:
 - `linear` — gradient liniowy (wartość domyślna);
 - `radial` — gradient promieniowy (radialny), kolor początkowy znajduje się w środku okręgu;
 - `sweep` — kierunek gradientu wyznaczony jest przez wskazówkę zegara.
- `android:useLevel` — ustawiamy wartość `true`, jeżeli gradient używany jest jako `LevelListDrawable` (zasób, który zarządza grupą obiektów graficznych). We wszystkich innych sytuacjach powinniśmy użyć wartości `false`.

Kolejny atrybut to `<padding>`, czyli margines wewnętrzny. Wyznacza on pozycję zawartości kontrolki (np. tekstu na przycisku), a nie pozycję kształtu. Atrybuty dostępne wraz ze znacznikiem `<padding>` to:

- `android:left` — margines wewnętrzny lewy,
- `android:top` — margines wewnętrzny górny,
- `android:right` — margines wewnętrzny prawy,
- `android:bottom` — margines wewnętrzny dolny.

Atrybut `<size>` definiuje rozmiar kształtu. Wraz z nim dostajemy dwa następujące atrybuty:

- `android:height` — wysokość kształtu,
- `android:width` — szerokość kształtu.

Rozmiar kontrolki będzie wyskalowany zgodnie z powyższymi wartościami. Dlatego kiedy używamy kontrolki `ImageView` i chcemy zablokować skalowanie, musimy ustawić atrybut `android:scaleType` na wartość `center`.

Atrybut `<solid>` ustawia jednolity kolor kształtu. Ma on tylko jedną opcję, a mianowicie `android:color`, w której podajemy wartość koloru.

Ostatni z omawianych znaczników to `<stroke>`, który rysuje ramkę otaczającą kontrolkę. Do dyspozycji mamy cztery atrybuty:

- `android:width` — grubość ramki,
- `android:color` — kolor ramki,
- `android:strokeWidth` — odległość pomiędzy kreskami (w ramce rysowanej linią przerywaną) wykorzystywaną wraz z atrybutem `android:strokeDash`,
- `android:strokeDash` — długość kresek (w ramce rysowanej linią przerywaną) wykorzystywaną wraz z atrybutem `android:strokeWidth`.

Omówiliśmy wszystkie elementy dostępne dla znacznika `<shape>`. Pora na przykład, który podsumuje zdobytą przez nas wiedzę.

PRZYKŁAD

Naszym celem jest utworzenie przycisku z gradientowym tłem, które będzie przechodziło w jednolity kolor w momencie wciśnięcia przycisku.

A więc zaczynamy:

1. Tworzymy nowy projekt. W tym celu w menu głównym wybieramy:
File/New/New Project...
2. Przechodzimy przez kolejne okna kreatora projektu, pozostając przy opcjach domyślnych. Opcje te były już szczegółowo opisywane w podrozdziale 1.2.
3. Teraz definiujemy styl naszego przycisku. W tym celu otwieramy plik *styles.xml* (w katalogu *app/values* projektu). Jak wiemy, będzie tam już jeden styl, a mianowicie temat naszej aplikacji. Dopisujemy styl zgodnie z listingiem 3.20.

LISTING 3.20. Styl naszego przycisku

```
<style name="StylPrzycisku">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#ffffff</item>
    <item name="android:gravity">center</item>
    <item name="android:layout_margin">10dp</item>
    <item name="android:textSize">25sp</item>
    <item name="android:textStyle">bold</item>
    <item name="android:shadowColor">#0a0a0a</item>
```



```
<item name="android:shadowDx">-2</item>
<item name="android:shadowDy">-2</item>
<item name="android:shadowRadius">1</item>
</style>
```

Opiszmy pokrótce wykorzystane przez nas atrybuty:

- `android:layout_width` (o wartości `match_parent`) — ustala szerokość komponentu na całą szerokość rodzica,
- `android:layout_height` (o wartości `wrap_content`) — ustala wysokość komponentu zgodnie z jego zawartością (np. dla przycisku będzie to rozmiar tekstu),
- `android:textColor` (o wartości `#ffffff`) — ustawia kolor tekstu na biały,
- `android:gravity` (o wartości `center`) — ustawia tekst na środku przycisku,
- `android:layout_margin` (o wartości `10dp`) — ustawia marginesy (zewnętrzne) na `10dp`,
- `android:textSize` (o wartości `25sp`) — ustawia rozmiar tekstu na `25sp`,
- `android:textStyle` (o wartości `bold`) — ustawia styl tekstu na pogrubiony,
- `android:shadowColor` (o wartości `#0a0a0a`) — ustawia kolor cieni tekstu,
- `android:shadowDx` (o wartości `-2`) — określa pozycję cienia w kierunku poziomym, wartość ujemna oznacza przesunięcie w lewo (dodatnia w prawo);
- `android:shadowDy` (o wartości `-2`) — określa pozycję cienia w kierunku pionowym, wartość ujemna oznacza przesunięcie w górę (dodatnia w dół);
- `android:shadowRadius` (o wartości `1`) — określa promień cienia.

4. W tym kroku realizujemy bardziej zaawansowane opcje wyglądu naszego przycisku. Zaczynamy od utworzenia nowego pliku XML (w katalogu *drawable*). Klikamy prawym przyciskiem myszy folder *drawable* i wybieramy opcję *New/Drawable resource file*. W oknie *File name* wpisujemy nazwę *gradient*. Po wciśnięciu przycisku *OK* nasz plik zostanie otwarty w oknie edycyjnym. Następnie wpisujemy do niego treść kodu z listingu 3.21.

LISTING 3.21. Definicja selektora wykorzystywanego przez przycisk

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_pressed="true" > <!-- stan wciśnięty -->
    <shape>
      <solid android:color="#52a4ef" />
      <stroke
        android:width="2dp"
        android:color="#245684" />
      <corners
        android:radius="5dp" />
      <padding
        android:left="12dp"
        android:top="12dp"
```

```

        android:right="12dp"
        android:bottom="12dp" />
    </shape>
</item>
<item>    <!-- stan domyślny -->
    <shape>
        <gradient
            android:startColor="#52a4ef"
            android:endColor="#245684"
            android:angle="45" />
        <stroke
            android:width="2dp"
            android:color="#245684" />
        <corners
            android:radius="5dp" />
        <padding
            android:left="12dp"
            android:top="12dp"
            android:right="12dp"
            android:bottom="12dp" />
    </shape>
</item>
</selector>

```

Opiszmy pokrótce użyte przez nas atrybuty:

- **<selector>** — pozwala na zmianę wyglądu przycisku w zależności od stanu, w jakim się znajduje; w naszym przykładzie mamy opisane dwa stany: atrybut `android:state_pressed` oznaczający przycisk wciśnięty oraz brak atrybutu w znaczniku `<item>` oznaczający stan domyślny (normalny);
 - **<shape>** — nadaje odpowiedni kształt naszemu komponentowi, a brak wartości oznacza domyślny prostokątny kształt; kolor definiujemy za pomocą znacznika `<solid>` i atrybutu `android:color`;
 - **<stroke>** — określa obramowanie komponentu; atrybut `android:width` ustawia grubość ramki, a atrybut `android:color` jej kolor;
 - **<corners>** — wprowadza zaokrąglone rogi, promień zaokrągleń definiuje atrybut `android:radius`;
 - **<padding>** — ustawia poszczególne marginesy wewnętrzne dzięki atrybutom `android:left` (lewy), `android:top` (górny), `android:right` (prawy), `android:bottom` (dolny);
 - **<gradient>** — ustawia przejście tonalne pomiędzy kolorem początkowym (atrybut `android:startColor`) a kolorem końcowym (atrybut `android:endColor`), kąt przejścia definiujemy dzięki atrybutowi `android:angle`.
5. Ostatnia rzecz, jaka nam pozostała, to wstawienie przycisku i ustawienie odpowiednich atrybutów. W tym celu przechodzimy do edytora tekstowego pliku układu głównej aktywności (domyślna nazwa to `activity_main.xml`) i wpisujemy kod przedstawiony na listingu 3.22.

LISTING 3.22. Plik układu głównej aktywności

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:text="OK"
        style="@style/StylPrzycisku"
        android:background="@drawable/gradient" />
</LinearLayout>
```

Zwróćmy uwagę na dwa atrybuty. Pierwszy to atrybut `style`, w którym podajemy nazwę naszego stylu zdefiniowaną w pliku *styles.xml*. Drugi zaś to `android:background`, podajemy w nim nazwę pliku, w którym zdefiniowaliśmy selektor.

6. Teraz wystarczy uruchomić naszą aplikację. Zobaczymy w niej przycisk z gradientowym tłem, które zmienia się na jednolity kolor po wciśnięciu tego przycisku.
-

Do tej pory zajmowaliśmy się tylko wyglądem naszej aplikacji. Jednakże aby mogła ona wykonywać żądania użytkownika, musimy odpowiednio ją zaprogramować. Ale o tym opowiemy w następnym rozdziale.

Skorowidz

A

- aktywność, activity, 123
 - cykl życia, 124
 - główna, 165
 - przekazywanie danych, 151, 155
 - stan aktywny, 126
 - stan wstrzymania, 126
 - stan zatrzymania, 126
 - stan zniszczenia, 126
- aktywowanie okien narzędziowych, 36
- alarm, 170
- alert zabezpieczeń systemu Windows, 20
- Android API, 250
- Android Marshmallow, 369
- Android Monitor, 128, 218
- Android SDK, 12
- Android Studio
 - instalacja środowiska, 9
 - importowanie ustawień, 12
 - Intel® HAXM, 14
 - wybór komponentów, 12
- okno edytora kodu, 39
- okno główne, 35
- opis środowiska, 34
- pierwsze uruchomienie aplikacji, 32
- pierwsze uruchomienie środowiska, 15
- tworzenie projektu, 15
- animacja poklatkowa, 334, 338
- animowanie kontroltek, 339
- aplikacja Draw 9-patch, 110
- atrybut
 - android:adjustViewBounds, 95, 97
 - android:angle, 117
 - android:baseline, 95
 - android:baselineAlignBottom, 95
 - android:bottom, 117
 - android:bottomLeftRadius, 116
 - android:bottomRightRadius, 116
 - android:centerColor, 117
 - android:centerX, 117
 - android:centerY, 117
 - android:color, 118
 - android:configChanges, 218, 220
 - android:cropToPadding, 95
 - android:dashGap, 118
 - android:dashWidth, 118
 - android:datePickerMode, 269
 - android:endColor, 117
 - android:gradientRadius, 117
 - android:gravity, 119
 - android:height, 117
 - android:innerRadius, 116
 - android:innerRadiusRatio, 116
 - android:inputType, 99
 - android:layout_above, 72
 - android:layout_alignBaseline, 73
 - android:layout_alignBottom, 73
 - android:layout_alignEnd, 73
 - android:layout_alignLeft, 73
 - android:layout_alignParentBottom, 70
 - android:layout_alignParentEnd, 70
 - android:layout_alignParentLeft, 70
 - android:layout_alignParentRight, 70
 - android:layout_alignParentStart, 70
 - android:layout_alignParentTop, 71, 75
 - android:layout_alignRight, 73
 - android:layout_alignStart, 73
 - android:layout_alignTop, 73
 - android:layout_below, 73

atrybut

- android:layout_centerHorizontal, 71
- android:layout_centerInParent, 71
- android:layout_centerVertical, 71
- android:layout_columnSpan, 81
- android:layout_gravity, 67, 69
- android:layout_height, 69, 75, 119
- android:layout_margin, 119
- android:layout_marginBottom, 70
- android:layout_marginLeft, 70
- android:layout_marginRight, 70
- android:layout_marginTop, 70
- android:layout_rowSpan, 81
- android:layout_toEndOf, 73
- android:layout_toLeftOf, 73
- android:layout_toRightOf, 73
- android:layout_toStartOf, 73
- android:layout_weight, 64, 69
- android:layout_width, 69, 75, 99, 119
- android:left, 117
- android:maxHeight, 95
- android:maxWidth, 95
- android:oneshot, 338
- android:orientation, 69
- android:radius, 116
- android:right, 117
- android:scaleType, 95
- android:shadowColor, 119
- android:shadowDx, 119
- android:shadowDy, 119
- android:shadowRadius, 119
- android:shape, 116
- android:src, 96
- android:startColor, 117
- android:state_activated, 115
- android:state_checkable, 115
- android:state_checked, 115
- android:state_enabled, 115
- android:state_focused, 113, 114
- android:state_pressed, 114
- android:state_selected, 115
- android:text, 78
- android:textColor, 119
- android:textSize, 90, 119
- android:textStyle, 119
- android:thickness, 116

- android:thicknessRatio, 116
- android:timePickerMode, 267
- android:tint, 96
- android:tintMode, 96
- android:top, 117
- android:topLeftRadius, 116
- android:topRightRadius, 116
- android:type, 117
- android:useLevel, 116, 117
- android:width, 117, 118, 294

AVD, Android Virtual Device, 12

- konfiguracja emulatora, 20
- tworzenie emulatora, 22
- uruchomienie menedżera, 21
- wybór systemu operacyjnego, 24
- wybór urządzenia, 23

B

- baza danych, 371
 - SQLite, 373
- błąd w instalacji aplikacji, 34
- Button, 91

C

- CheckBox, 80, 93
- cykl życia aktywności, 124

D

- debugowanie USB, 27
- definicja
 - animacji poklatkowej, 335
 - selektora, 119
 - stylu, 104
 - zasobu tekstowego, 60
 - znacznika, 115
- dodawanie
 - aktywności, 19, 147
 - filtra intencji, 157
- dostęp do listy kontaktów, 361
- dostosowywanie parametrów aktywności, 19
- drzewo komponentów, 52
- dziedziczenie stylów, 105

E

EditText, 98
 edycja właściwości Text, 57
 edytor kodu, 39
 elementy, 39
 formatowanie kodu, 49
 generowanie kodu, 46
 lokalizacja błędów, 42
 lupa, 42
 parametry metod, 47
 podgląd dokumentacji, 47
 podział okna edycyjnego, 42
 uzupełnianie kodu, 44
 uzupełnianie składni, 45
 zwijanie i rozwijanie kodu, 48
 edytor układów, 51
 ekran powitalny, 15
 emulator AVD, 20

F

filtr intencji, 145, 157
 filtrowanie informacji, 128
 formatowanie kodu, 49
 fragmenty, fragments, 171
 funkcja lupy, 42

G

generowanie kodu, 46
 gęstość pikseli, 66
 Google USB Driver, 28
 grupowanie powiadomień, 197, 199

I

ImageButton, 94, 113
 ImageView, 95
 importowanie klasy, 129
 informacja
 o błędach, 41
 o parametrach metod, 47
 informacyjne okna dialogowe, 138
 inspekcje, 69
 instalacja
 sterowników, 29
 środowiska Android Studio, 9

Intel® HAXM, 14
 intencje, intents, 144
 domniemane, 145, 157
 jawne, 145, 146
 interfejs użytkownika, 51
 komponenty, 87

J

Java SDK, 9
 JDK, 9
 jednostka
 in, 66
 mm, 66
 pt, 66
 px, 66
 sip, 66
 język aplikacji, 54

K

kategoria Widgets, 88
 klasa
 AnimationDrawable, 335, 338
 BufferedReader, 360
 Canvas, 308
 ContentResolver, 362
 CursorLoader, 367
 DatePicker, 264
 DialogFragment, 251, 253, 259
 GridView, 283
 HorizontalScrollView, 290
 ImageSwitcher, 297
 ListActivity, 241
 ListFragment, 241
 MainActivity, 153, 212, 284, 346, 364
 RatingBar, 279
 SeekBar, 275
 Shader, 324
 SharedPreferences, 351, 354, 358
 StringBuilder, 360
 TimePicker, 264, 266
 VERSION_CODES, 249
 kolory tematu AppTheme, 104
 komponent
 Android SDK, 12
 Android Virtual Device, 12
 CheckBox, 80

komponent

- GridView, 283
- HorizontalScrollView, 290
- ImageButton, 113
- ImageSwitcher, 297
- ListView, 241
- ScrollView, 86

komponenty

- edycyjne, 98
- interfejsu użytkownika, 87, 207

konfiguracja

- aktywności, 147
- emulatora AVD, 20
- urządzenia fizycznego, 26

konstruktor `OutputStreamWriter()`, 359

kontakty, 361

kontrolka

- `AutoCompleteTextView`, 221, 224
- `GridView`, 286
- `HorizontalScrollView`, 293
- `ListView`, 225, 235, 302
- `MultiAutoCompleteTextView`, 224
- `RatingBar`, 281
- `SeekBar`, 277
- `Spinner`, 237
- `TextView`, 102, 330, 333
- `TimePicker`, 274

kontrolki animowane, 339

kształty, 115, 308

L

lista kontaktów, 361

logowanie, 79

lokalizacja błędów, 42

lupa, 42

M

manifest aplikacji, 159, 169

metoda

- `addFrame()`, 335
- `checkSelfPermission()`, 370
- `delete()`, 362
- `drawArc()`, 310
- `drawBitmap()`, 311
- `drawCircle()`, 310

`drawLine()`, 310`drawOval()`, 311`drawRect()`, 311`drawText()`, 311`findViewById()`, 260, 300`finish()`, 207`getCount()`, 288`getDuration()`, 335`getFrame()`, 335`getItem()`, 288`getItemId()`, 288`getNumberOfFrames()`, 335`getPreferences()`, 352`getSharedPreferences()`, 351`getType()`, 362`getView()`, 288`insert()`, 362`isOneShot()`, 335`isRunning()`, 335`onClick()`, 154, 185, 198, 375`onClickButtonLokalizacja()`, 165`onClickButtonMiasto()`, 164`onClickButtonStronaWWW()`, 164`onClickHigh()`, 195`onClickNormal()`, 195`onConfigurationChanged()`, 218`onCreate()`, 178, 202, 210, 267`onCreateView()`, 181, 259`onDestroy()`, 210`onPasue()`, 208`onRestoreInstanceState()`, 208, 213, 214`onSaveInstanceState()`, 208, 213`openFileOutput()`, 360`putBoolean()`, 355`putString()`, 214`query()`, 362`setAntiAlias()`, 309`setColor()`, 309`setFactory()`, 297`setInAnimation()`, 297`setOneShot()`, 335`setOutAnimation()`, 297`setStrokeCap()`, 309`setStrokeJoin()`, 309`setStrokeWidth()`, 310`setStyle()`, 309`shouldShowRequestPermission`

→ `Rationale()`, 370

start(), 335
 stop(), 335
 update(), 362

O

obrazy, 95, 283, 302
 typu Nine-patch, 109, 111
 obsługa
 zdarzenia, 134, 148
 zmian konfiguracji, 215
 obszar ekranu aplikacji, 315
 odczytywanie
 danych, 351
 kontaktów, 366
 okna
 dialogowe, 140
 edytora kodu, 35, 39, 41
 główne środowiska, 35
 informacyjne, 138
 logowania, 79
 narzędziowe projektu, 35
 podpowiedzi, 44
 porad, 20, 21
 postępu, 130, 132
 wirtualnych urządzeń, 25
 wyboru urządzenia, 23
 z opcjami do wyboru, 143
 okna narzędziowe, 36
 Android Monitor, 38, 128, 218
 Build Variants, 37
 Captures, 37
 Event Log, 38
 Extract Resource, 61
 Favorites, 37
 Gradle, 38
 Gradle Console, 38
 Project, 36
 Run, 39
 Structure, 36
 Switcher, 39
 Terminal, 38
 TODO, 38
 Opcje
 folderów, 29
 programistyczne, 27

opis
 okna edytora kodu, 39
 środowiska, 34
 oprogramowanie sterowników, 31
 oprogramowywanie kontrolerek, 221
 orientacja urządzenia, 179, 215
 ostrzeżenia, 68

P

paleta, 52
 pasek
 narzędziowy, 35, 52
 nawigacyjny, 35
 postępu, 136
 stanu, 36
 pierwsze uruchomienie aplikacji, 32
 pierwszy projekt, 15
 plik
 activity_main.xml, 58
 activity_druga_aktywnosc.xml, 153, 184
 activity_main.xml, 51, 74, 131, 135, 139,
 162, 175, 184, 344
 activity_trzecia_aktywnosc.xml, 159
 AndroidManifest.xml, 108, 149, 159
 animacja.xml, 336
 bitmapa_clamp.xml, 325
 bitmapa_mirror.xml, 325
 bitmapa_repeat.xml, 325
 blink.xml, 343
 colors.xml, 103
 dialogfragment.xml, 257
 DrugaAktywnosc.java, 202
 fade_out.xml, 343
 Fragment1.java, 174, 251
 fragment1.xml, 173, 180
 fragment2.xml, 174, 181
 jdk-8u144-windows-x64.exe, 10
 MainActivity.java, 169
 MyListAdapter.java, 305
 rotate.xml, 344
 strings.xml, 62, 233
 styles.xml, 258
 TimePickerFragment.java, 273
 translate.xml, 344
 układu activity_main.xml, 168
 układu XML, 173
 zoom_out.xml, 343

- pliki XML, 112
- podgląd dokumentacji, 47
- podpięcie zdarzenia do przycisku, 134, 148
- podział okna edycyjnego, 42
- pole tekstowe, 88
- powiadomienie, notification, 183
 - grupowanie, 197, 199
 - priorytety, 193
 - proste, 183
 - z odpowiedzią, 200, 203
 - z rozszerzonym widokiem, 190
- program
 - Aktywnosci01, 127
 - Dialog01, 131
 - Dialog02, 134
 - Dialog03, 139, 141
 - Dialog04, 142
- przekazywanie danych do aktywności, 151
- przełącznik, 94
- przycisk, 91
 - obrazkowy, 94
 - opcji, 92
 - wstecz, 149
 - wyboru, 93

R

- RadioButton, 92
- rozmiar ekranu, 22, 315
- rozwijanie kodu, 48
- rysowanie
 - bitmap, 324
 - kształtów, 308
 - obiektów graficznych, 313
 - po ekranie, 321
 - wewnątrz kontrolki, 318

S

- ScrollView, 86
- selektory, 112, 114
- siatka, 78
- skojarzenia aplikacji, 162
- skróty klawiaturowe, 377
- sterowniki Google USB Driver, 28
- strumień
 - bajtowy, 359
 - znakowy, 359

- styl, 101
 - AppTheme, 102
 - przycisku, 118
- Switch, 94
- szablony, 385

T

- technologia nine-patch, 111
- temat, theme, 107
 - aplikacji, 53
 - AppTheme, 107
- TextView, 88, 102
- timer, 170
- ToggleButton, 94
- tryb widoku, 53
- tworzenie
 - emulatora AVD, 22
 - interfejsu użytkownika, 51
 - nowego pliku, 172
 - nowego projektu, 16
 - pliku układu, 173
 - pliku XML, 112
 - zasobu tekstowego, 60, 61
- typ urządzenia, 53
- typy proste, 351

U

- uaktywnianie opcji programistycznych, 26
- układ, layout, 54
 - activity_main.xml, 143, 152
 - ConstraintLayout, 54
 - FrameLayout, 54, 83
 - GridLayout, 54, 80
 - LinearLayout, 54, 63, 263
 - RelativeLayout, 55, 70, 262
 - TableLayout, 55, 76
- układy zagnieżdżone, 84
- ukryte foldery, 29
- uruchamianie
 - aktywności, 161
 - Android SDK, 28
- urządzenie
 - fizyczne, 26
 - nierozpoznane, 29
 - poprawnie zainstalowane, 31

usługi działające w tle, 144
ustawianie alarmu, 170
uzupełnianie
 kodu, 44
 składni, 45

W

warianty układów, 54
wartość gęstości pikseli, 66
wersje
 Android API, 53, 250
 SDK, 247
Widgets, 88
widok rozszerzony, 190
właściwości, 52
 komponentu, 57
 przycisku, 58, 59
właściwość text, 57
wybór
 aktywności, 161
 docelowego urządzenia, 33
 koloru, 103
 przeznaczenia aplikacji, 17
 przycisku, 56
 sposobu instalacji sterowników, 30
 systemu operacyjnego, 24
 urządzenia, 23
 wersji SDK, 247
 wersji systemu, 18
wyjątek FileNotFoundException, 360
wykorzystanie fragmentów, 176
wyłączanie ostrzeżeń, 68
wyszukiwanie oprogramowania
 sterowników, 31
wyświetlanie
 bitmapy, 328
 obrazów, 286, 300

Z

zachowywanie stanu aktywności, 207
zagnieżdżanie układów, 84
zapisywanie
 danych, 351
 danych do pliku, 356
 stanu kontrolki, 213
 wartości typów prostych, 351

zarządzanie inspekcjami, 69
zatrzymanie uruchamiania aplikacji, 34
zdarzenie
 onClick(), 148, 150
 OnCheckedChangeListener(), 271
zgoda na debugowanie USB, 33
zmiana
 kolorów przycisku, 114
 orientacji ekranu, 53, 215
 szerokości przycisku, 64
zmiany konfiguracyjne
 addHeaderView(), 230
 getText(), 231
 keyboardHidden, 220
 onClick(), 231
 orientation, 220
 screenSize, 220
 setAdapter(), 230
znacznik, tag, 130
 <action>, 158
 <activity>, 109, 150, 158, 217
 <application>, 150
 <Button>, 75, 181
 <category>, 158
 <CheckedTextView>, 236
 <color>, 114
 <corners>, 116, 120
 <data>, 167
 <dimen>, 91
 <EditText>, 98, 180
 <fragment>, 175
 <gradient>, 117, 120
 <ImageView>, 306
 <intent-filter>, 145, 157, 167
 <LinearLayout>, 56, 177
 <ListView>, 227
 <manifest>, 158
 <padding>, 117, 120
 <RadioGroup>, 92
 <selector>, 112, 120
 <shape>, 115, 118, 120
 <stroke>, 116, 118, 120
 <TableRow>, 76, 80, 264
 <TextView>, 177, 185, 224
 <uses-permission>, 169
zwijanie kodu, 48

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



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

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

Czy umiesz sobie wyobrazić świat urządzeń cyfrowych, w którym nagle zabrakło Androida? Czy wiesz, jak wiele codziennych czynności okazałoby się wówczas nie do wykonania? Przez kilka ostatnich lat Android niepostrzeżenie ulokował się we wszystkich smartfonach czy tabletach, a aplikacje pisane dla tego systemu służą nam wszędzie: w kinie, w sklepie, na biwaku, na nartach i u lekarza. I wciąż potrzebujemy nowych! Jeśli chcesz dołączyć do grona innowacyjnych kreatorów, tworzących aplikacje ułatwiające i uprzyjemniające codzienne życie setek milionów ludzi, czym prędzej zapoznaj się z możliwościami Android Studio! To pozwoli Ci zacząć przygodę z Javą i efektywnie pisać dobre programy.

Dzięki informacjom zawartym w tej książce uda Ci się szybko opanować podstawy pracy w Android Studio. Dowiesz się, jak zainstalować środowisko programistyczne, utworzyć i dopracować interfejs użytkownika, stosować tematy i style, generować komunikaty, wprowadzać elementy wizualne i multimedialne. Poznasz sposoby dodawania funkcjonalności, zobaczysz, jak zapewnić zapisywanie i odtwarzanie danych. Na końcu książki umieszczono dodatek, w którym znajdziesz także skróty klawiaturowe i informacje o szablonach. Na co czekasz? Czas wreszcie przekuć Twoje pomysły w prawdziwe aplikacje. Android czeka na Ciebie!

- Poznajemy Android Studio
- Podstawy tworzenia interfejsu użytkownika
- Style i tematy
- Aktywności i fragmenty
- Powiadomienia
- Praca z komponentami interfejsu użytkownika
- Obrazy i animacje
- Zapisywanie i odczytywanie danych
- Usprawnienia w pisaniu kodu

Zainspiruj się Androidem!

Helion 	
	helion.pl
	0 801 339900
	0 601 339900

Sprawdź nasze szkolenia!

SZKOLENIA



AKADEMIA IT & BUSINESS

WWW.SZKOLENIA.HELION.PL

KOD KORZYŚCI
Sięgnij po więcej! ►



ISBN 978-83-283-3593-6



9 788328 335936

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 69,00 zł