



Helion



Andrzej Stasiewicz

ANDROID

Podstawy tworzenia aplikacji

Przekonaj się, że nie taki Android straszny, jak go malują...

Poznaj najpopularniejszy system operacyjny dla urządzeń mobilnych

Naucz się tworzyć proste aplikacje i analizować kod przykładów

Dowiedz się, jakich narzędzi potrzebujesz, aby rozpocząć pracę

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.

Redaktor prowadzący: Michał Mrowiec

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

Fotografia na okładce została wykorzystana za zgodą Shutterstock.com

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/andrpa>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:

<ftp://ftp.helion.pl/przyklady/andrpa.zip>

ISBN: 978-83-246-7006-2

Copyright © Helion 2013

Printed in Poland.

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

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

Spis treści

Wstęp	5
Rozdział 1. Instalowanie środowiska programistycznego	9
Instalowanie Android SDK, Javy i edytora Eclipse	9
Konfiguracja środowiska programistycznego Eclipse	11
Pierwsza aplikacja	15
Rozdział 2. Wygląd pierwszej aplikacji	19
Katalog res — zasoby aplikacji	19
Layouts, czyli wyglądy aplikacji	22
LinearLayout — obiekty ułożone obok siebie	24
TableLayout — obiekty ułożone w oczkach sieci	27
AbsoluteLayout — rozłożenie swobodne	32
Rozdział 3. Graficzne zasoby aplikacji	39
Struktura katalogów drawable	39
Emulatory o ekranach różnej jakości	41
Bitmap, czyli mapa bitowa	43
Mapa bitowa opakowana w atrybuty XML	48
Wiele map bitowych w jednym opakowaniu XML	52
Rozdział 4. Więcej o wyglądzie aplikacji	57
ScrollView — ekran z gumy	57
Kolory	61
Shapes — kształty	68
Rozdział 5. Programowanie czas zacząć!	77
Przycisk „Koniec”	77
Zegarek dla ubogich	82
Kółko i krzyżyk, a przy okazji definiowanie stylów	87
Rozdział 6. Efekty specjalne	97
Przygotowanie animacji	98
Przygotowanie interfejsu użytkownika	103
Uruchomienie animacji	105
Animacja innych komponentów	108
Łączenie animacji	110
Animacja poklatkowa map bitowych	113

Rozdział 7. Własne komponenty graficzne	121
Komponenty rozszerzają bazowy superkomponent View	122
Przegląd możliwości graficznych	132
Rozdział 8. Mapy bitowe	145
Mapa bitowa zaczerpnięta z zasobów aplikacji	145
Rysowanie na mapie bitowej zaczerpniętej z zasobów aplikacji	152
Przekształcenie mapy bitowej techniką piksel po pikselu	154
Uzyskanie nowej, czystej mapy bitowej	156
Rozdział 9. Wątek w drugim planie	167
Klasa AsyncTask i rysowanie w drugim planie	167
Ściąganie danych z internetu	181
Rozdział 10. Więcej ekranów dla aplikacji	191
Okno główne	192
Ta sama aplikacja napisana lepiej, bo krócej	205
Słowniczek	211
Skorowidz	213

Rozdział 3.

Graficzne zasoby aplikacji

W tym rozdziale omówię dokładniej, jakie zasoby można przechowywać w podfolderach *res/drawable*. Materiały do tego projektu znajdują się w pliku o nazwie *Rozdział_3*.

Struktura katalogów drawable

Już wiesz, że wyglądu aplikacji w Androidzie raczej się nie programuje (choć można), ale opisuje się go w specjalny sposób, umieszczając odpowiednie informacje w odpowiednich folderach i plikach. Folder o nazwie *res* (skrót od ang. wyrazu *resources* — zasoby) jest tym miejscem, w którym umieszczasz swoje zasoby. Folder ten ma dalszą strukturę podfolderów i teraz omówię podfoldery o nazwach *drawable*, co należy przetłumaczyć jako „elementy rysowalne”. Podfoldery *drawable* zawierają elementy graficzne aplikacji, a także pewne ich opisy, jakby opakowania, przygotowane w języku XML.

Rozpocznij nowy projekt. Wybierz opcję *File*, a dalej *New* i *Android Application Project*. Określ nazwę aplikacji (nazwa ta będzie widoczna w Twoim telefonie), nazwę projektu w środowisku Eclipse, wymyśl nazwę dla pakietu Javy, najlepiej stosując zasadę odwracania kolejności wyrazów z nazwy domeny internetowej. Jeśli zainstalowałeś kilka różnych wersji Androida, wskaż jedną z nich jako zalecaną dla Twojej aplikacji. Określ też minimalne parametry platformy, rozwojowo najstarszej i najuboższej, dla których Twoja aplikacja powinna działać.

Środowisko Eclipse utworzy rodzinę folderów. W tym rozdziale najważniejsze będą dla Ciebie foldery *drawable*, znajdujące się w folderze *res*.

Foldery *drawable* zawierają elementy graficzne aplikacji, przygotowane do wyświetlenia na różnej klasy urządzeniach. Jeszcze niedawno były trzy foldery *drawable*, teraz jest ich więcej, bo pojawiają się coraz lepsze urządzenia mobilne. Postfix *-ldpi* należy rozwinąć jako *low dot per inch*, co w tym wypadku oznacza „urządzenie o małej

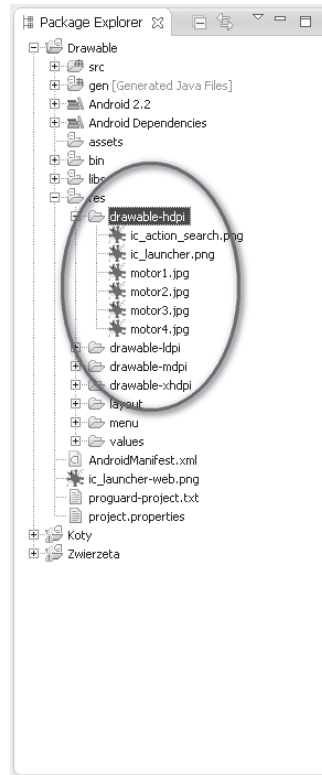
gęstości pikseli”. W folderze *drawable-ldpi* umieścisz wersje grafik dla kiepskich ekraników. Grafiki te powinny być mniejsze, ale także mogą zawierać mniej detali.

Katalogi z postfixami *-mdpi*, *-hdpi* i *-xhdpi* zawierają zasadniczo te same grafiki, ale coraz większe, staranniejsze, z większą ilością detali.

Widzisz więc, jaką ścieżką poszli twórcy systemu. Zamiast zastosować algorytm dopasowywania grafiki do rzeczywistego rozmiaru ekranu, dali programiście szansę, aby sam przygotował kilka wersji rozwiązań graficznych. Takie posunięcie umożliwia przygotowanie równie pięknych wersji aplikacji na malutki telefon i na olbrzymi tablet.

Rodzina folderów *drawable* powinna zawierać zasoby o tych samych nazwach. System przeanalizuje dane o jakości ekranu urządzenia mobilnego i sam ustali, do którego folderu *drawable* skierować program po zasób o konkretnej nazwie (rysunek 3.1).

Rysunek 3.1.
Rodzina folderów
drawable

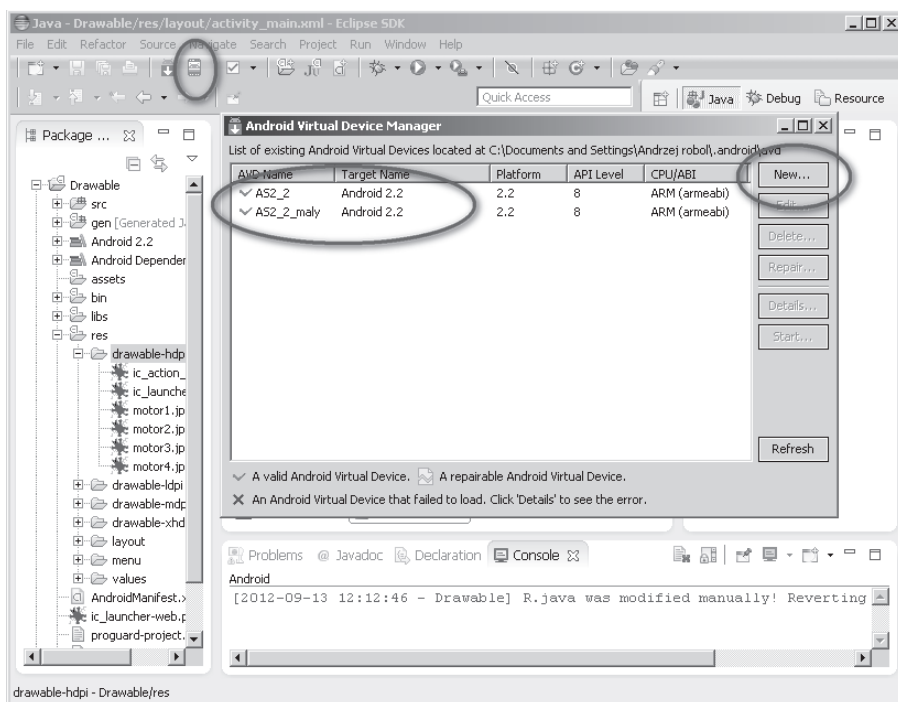


Nie ma dokładnych wytycznych, jak sporządzać i dzielić zasoby na poszczególne foldery *drawable*. Przyjmuje się, że jeśli dla urządzenia typowego, czyli średniego *-mdpi*, przygotowuje się mapę bitową o boku np. 400 pikseli, to dla urządzenia małego *-ldpi* trzeba będzie ją zmniejszyć do 75 procent, dla urządzenia dużego *-hdpi* powiększyć do 150 procent, a dla ekstradużego *-xhdpi* do 200 procent. Oprócz samego zmieniania rozmiarów możesz „ręcznie” zadbać o więcej albo mniej szczegółów. To jest już praca artystyczna.

Emulatory o ekranach różnej jakości

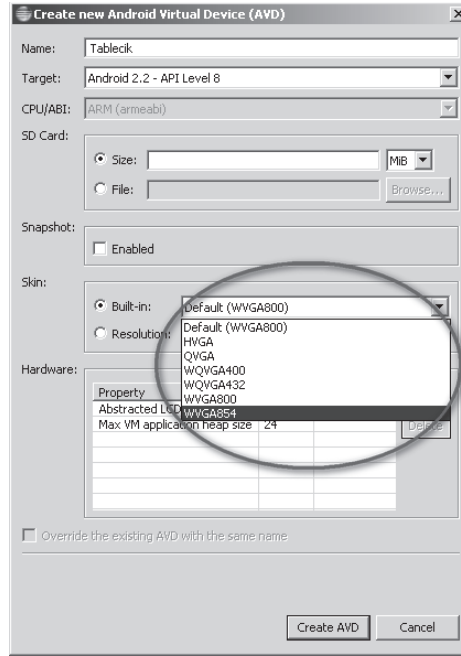
Żeby móc testować zachowanie się programu na urządzeniach różnej klasy, należy zdefiniować kilka wirtualnych AVD (*Android Virtual Device*), różniących się klasą ekranów. Prawdopodobnie pierwsze urządzenie zdefiniowałeś jako *default*, czyli domyślne, teraz więc postaraj się o wyraźnie lepszy albo gorszy telefon (rysunek 3.2). W tym celu kliknij ikonę menedżera urządzeń i stwórz sobie wirtualne urządzenie z małym i dużym ekranem.

Podczas opisywania typu nowego urządzenia zwróć uwagę na techniczne oznaczenia rodzajów wyświetlaczy. Oznaczenie najgorszego ekranu to QVGA, lepszego to HVGA i wreszcie najlepszego WVGA (rysunek 3.3). Na rysunku 3.4 widać różnice między najgorszym a najlepszym ekranem.



Rysunek 3.2. Tworzenie wirtualnego urządzenia z małym i dużym ekranem

Rysunek 3.3.
*Nadanie nazwy nowemu
 urządzeniu, wybór
 platformy i co
 najważniejsze —
 określenie jej ekranu*



Rysunek 3.4.
*Urządzenie QVGA na tle
 urządzenia WVGA800*



Bitmap, czyli mapa bitowa

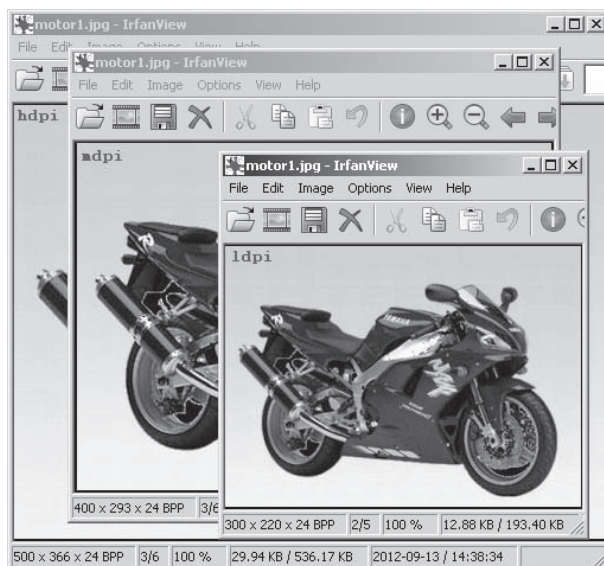
Po tych czynnościach technicznych spróbujesz wreszcie pracy z grafikami, których jakości są dopasowane do możliwości technicznych urządzenia.

Znajdź jakiś obrazek, którego większy bok ma rozmiar ok. 400 pikseli. Będzie to podstawowa wersja grafiki, którą umieścisz w katalogu z postfiksem *-mdpi*. Za pomocą edytora grafiki — np. *IrfanView* — przygotuj wersję *-ldpi* tego obrazka o boku 300 pikseli i wersję *-hdpi* o boku 500 pikseli. Możesz przygotować jeszcze wersję ekstra *-xhdpi* dla superurządzeń, ale zalecany tutaj, trochę przestarzały, choć ciągle najpopularniejszy Android 2.2 nie obsługuje takiego standardu.

Wersje obrazka należy jakoś oznaczyć, np. nadrukowując na nie w programie graficznym napisy *ldpi*, *mdpi*, *hdpi* (rysunek 3.5). Dzięki temu będziesz mógł śledzić, jak konkretne urządzenie wybiera sobie grafikę do wyświetlenia.

Rysunek 3.5.

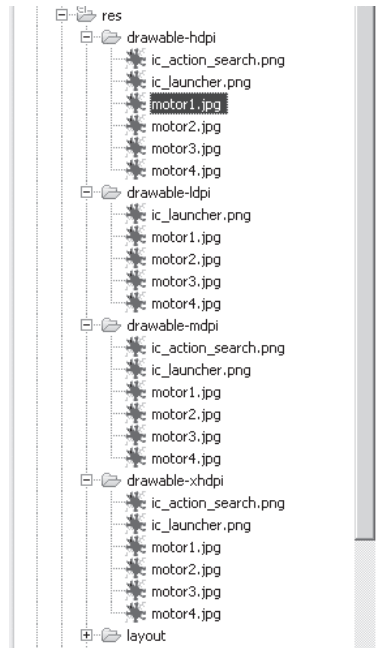
Obrazki o identycznych nazwach, ale różniące się rozmiarem (w proporcjach mniej więcej 0,75:1:1,5)



Przygotowane obrazy należy zapisać pod tą samą nazwą, ale w różnych folderach *drawable*, koniecznie zwracając uwagę, aby obrazek o odpowiedniej jakości znalazł się w odpowiednim folderze (rysunek 3.6). Po rozmieszczeniu plików w folderach środowisko Eclipse może początkowo ich nie dostrzec — wybierz z menu *File* opcję *Refresh* (odśwież widok). Zapamiętaj, bo jest to ważne podczas przygotowywania zasobów aplikacji.

Gdy teraz program przywoła obrazek o nazwie np. *motor1.jpg*, system Android sprawdzi, jakim ekranem dysponuje urządzenie, skieruje się do odpowiedniego folderu *drawable* i wyciągnie stamtąd odpowiednią wersję pliku *motor.1*. Sprawdź to.

Rysunek 3.6.
Rodzina folderów *drawable* zawiera pliki o takich samych nazwach, ale różniące się jakością i przeznaczone na różne typy wyświetlaczy urządzeń mobilnych



Zbuduj najprostszą aplikację, która wyświetli jeden z obrazków. Tak jak poprzednio, zasadniczym „polem bitwy” jest główny plik wyglądu, umieszczony w folderach *res/layout*. Jeśli nie zmieniłeś nazw w kreatorze nowego projektu, plik ten powinien się nazywać *activity_main.xml*. Kliknij go dwukrotnie, otwierając go do edycji. Na dole edytora wybierz edycję wizualną (nie zaś zwykłą, tekstową).

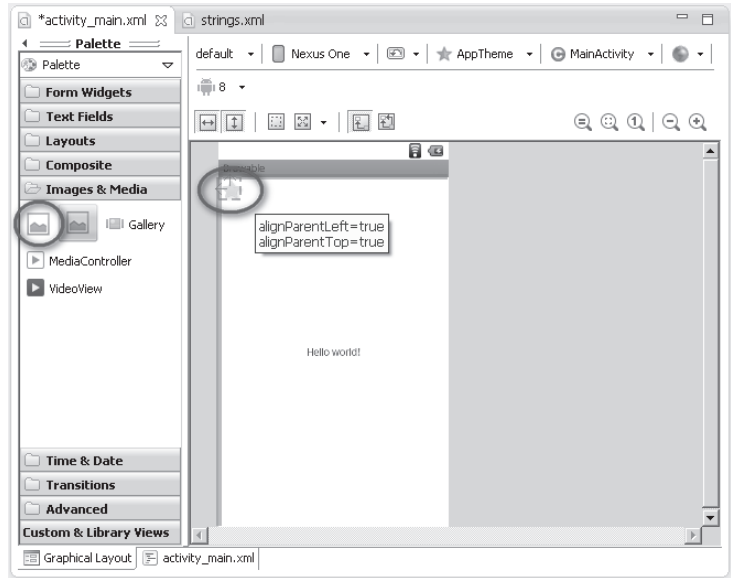
Jeśli już zaczyna Cię denerwować napis „Hello World”, reprezentowany przez komponent *TextView*, kliknij go prawym klawiszem myszki i usuń z aplikacji.

W zakładce *Images & Media* znajdź komponent *ImageView* i przeciągnij go w pole wyświetlacza urządzenia mobilnego. Jeśli umieścisz go dokładnie w rogu pola, system od razu ustawi właściwości *alignParent...* (wyrównaj w dostępnej przestrzeni — tutaj: w przestrzeni całego ekranu). Możesz umieścić *ImageView* w inny sposób — wtedy system zaproponuje właściwości *center...* albo *margin...* (rysunek 3.7).

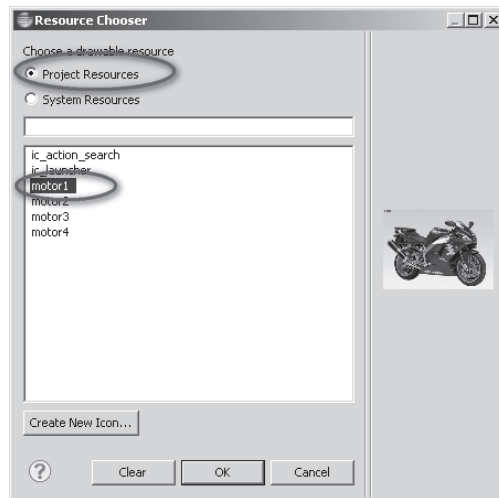
Po umieszczeniu komponentu w polu ekranu prawdopodobnie automatycznie pojawi się *Resource Chooser* (wybieranie zasobu). Jeśli zaznaczono *Project Resources* (a nie *System Resources* — przebogate zasoby własne Androida), okienko *Choosera* zajrzy do Twoich katalogów *drawable* i zaproponuje wybór któregoś obrazka (rysunek 3.8).

Jeśli postępowałeś tak, jak pokazałem to na kilku ostatnich rysunkach, plik wyglądu aplikacji *activity_main.xml* powinien mieć następującą, Zapewne już Ci znaną treść (widoczną po wybraniu odpowiedniej zakładki pod edytorkiem wizualnym):

Rysunek 3.7.
Praca na zakładce
Images & Media



Rysunek 3.8.
Wybieranie zasobu
z katalogów drawable



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

```
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:src="@drawable/motor1" />
```

```
</RelativeLayout>
```

W tej aplikacji obowiązuje nieomawiany tutaj rozkład `RelativeLayout` (określaj położenia kolejnych komponentów przez odwołania do innych, wcześniej rozlokowanych). Rozkład ten ma jedną szczególną cechę — wszystkie komponenty powinny być nazwane, tak aby inne mogły się do nich odwoływać, np. wskazując „chcę być wyrównany do lewej strony tamtego buttona”. W następnym kroku dodasz więcej komponentów i zobaczysz, jak pracuje przyjemny w edycji `RelativeLayout`.

Nazwa naszego obrazka brzmi `imageView1` i określa ją linia:

```
android:id="@+id/imageView1"
```

Od tej pory będziesz nazywać wszystkie komponenty, czego w zasadzie wymaga rozkład `RelativeLayout` i czego bezwzględnie będą wymagały techniki programowania w Javie, do których nieuchronnie się zbliżasz. Rozmiary komponentu w omawianym przypadku są zdefiniowane frazami `wrap_content`, czyli „otaczaj swoją zawartość” albo „bądź tak duży, jak twoja zawartość” — w tym przypadku mapka bitowa z motocyklem.

Z kolei położenie komponentu definiują linie:

```
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"
```

które należy czytać jako: „bądź w lewym górnym rogu swego rodzica”, czyli komponentu `RelativeLayout`, który wypełnia cały dostępny dla aplikacji ekran.

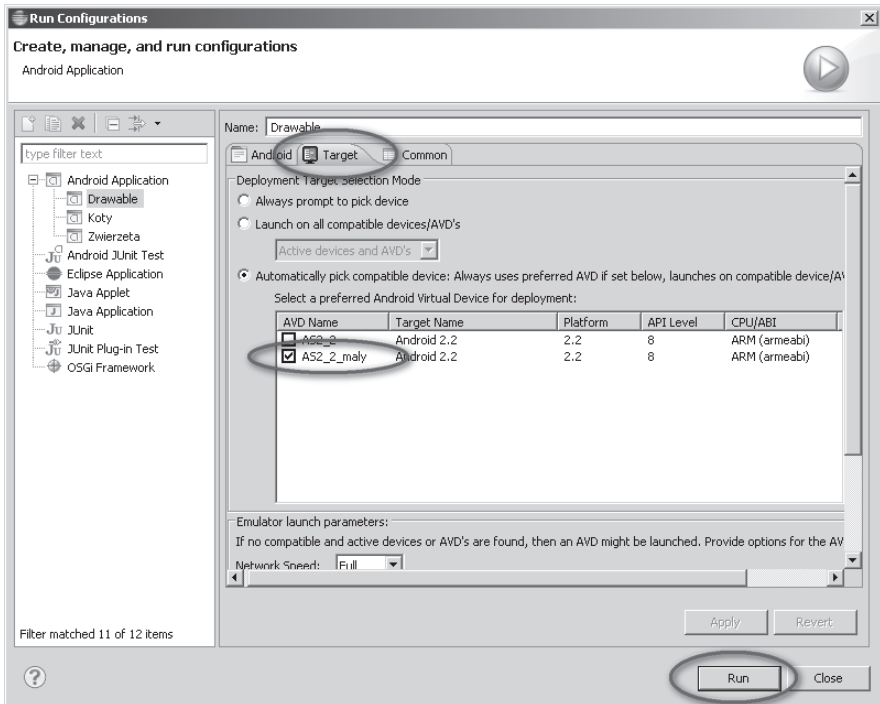
Najważniejszą linią jest połączenie komponentu o nazwie `imageView1` z mapą bitową `motor1.jpg` znajdującą się w folderach `drawable`:

```
android:src="@drawable/motor1" />
```

Zobacz, jak wygląda omawiana aplikacja na ekraniku AVD, czyli wirtualnej maszyny Androida. Pamiętaj jednak, że jest kilka maszyn AVD. Jak wybrać konkretną? W menu *Run* znajdziesz polecenie *Run Configurations* i tam wskaż maszynę, na której uruchomisz program (rysunek 3.9). Na zakładce *Target* (na czym uruchomić) powinieneś znaleźć swoje urządzenia AVD i zaznaczyć jedno z nich.

Teraz pozostało sprawdzić działanie programu (rysunek 3.10). Wszystko gra! Po uruchomieniu programu na dwóch różnych urządzeniach dostajesz tę samą fotografię motocykla. Czy na pewno tę samą? Napis w lewym górnym rogu odkrywa tajemnicę — mapy bitowe zostały pobrane z różnych folderów `drawable`!

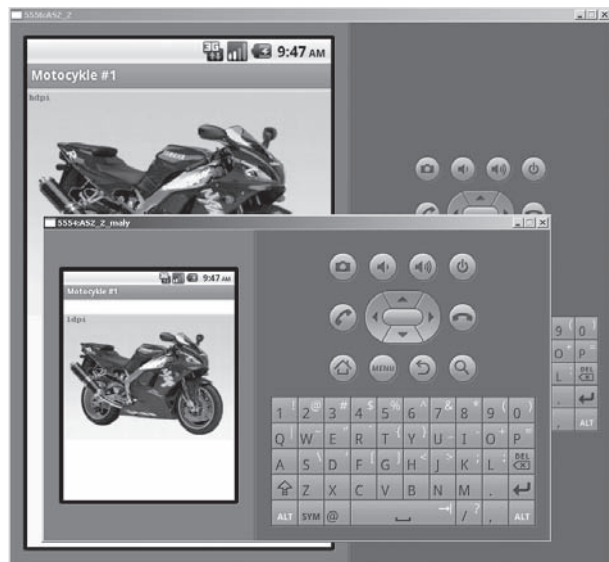
Być może zauważyłeś, że przy komponencie obrazka albo przy jego opisie w pliku XML znajduje się mały żółty trójkątik uwagi, którą chce Ci odpowiedzieć kompilator: „*Missing contentDescription attribute on image*” (zapomniałeś o parametrze `contentDescription`) — rysunek 3.11. Każdy element wizualny powinien być opisany, a opis ten jest przeznaczony do automatycznego odczytywania syntetycznym głosem dla kogoś, kto nie może zobaczyć motocykla, np. dla osoby niewidomej. W dalszej części nie będziesz dodawać atrybutu `contentDescription`.



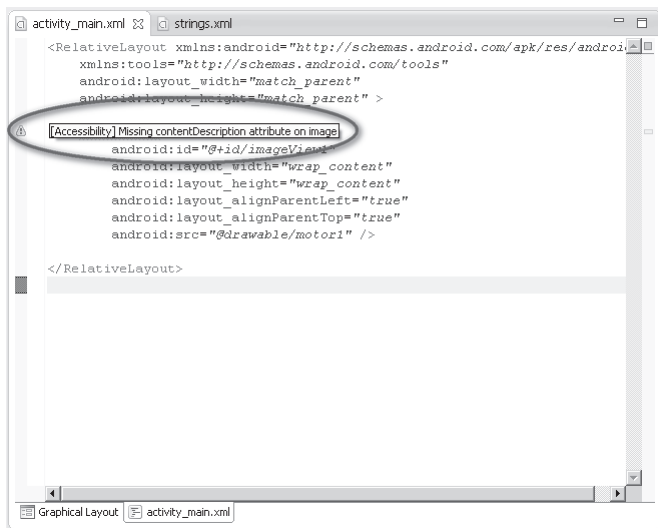
Rysunek 3.9. Menu Run i dalej Run Configurations pozwala opisać szczegóły uruchamiania aplikacji

Rysunek 3.10.

Dwa różne urządzenia i taka sama fotografia? Napis w lewym górnym rogu wskazuje, że mapy bitowe zostały pobrane z różnych folderów drawable



Rysunek 3.11.
Trójkącik uwagi, którą
podpowiada kompilator



Mapa bitowa opakowana w atrybuty XML

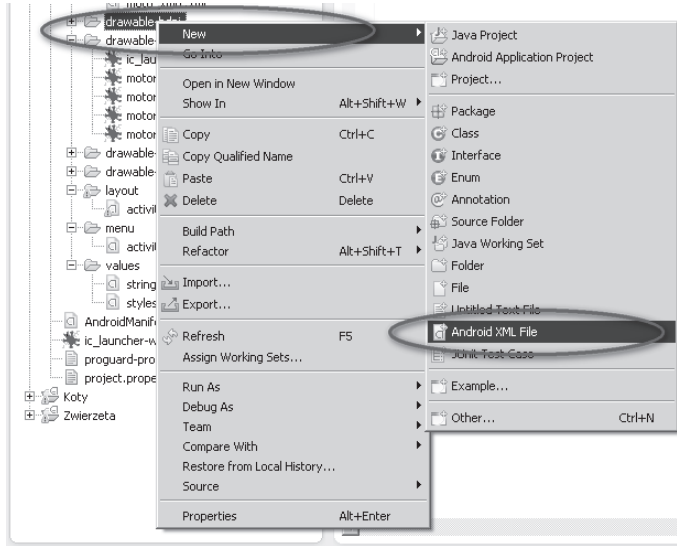
To, co zrobiłeś do tej pory z mapą bitową, można nazwać wyświetleniem surowego obrazka. Było to dobre i zazwyczaj jest wystarczające. Jednak można pójść o krok dalej, mianowicie: w folderach *drawable*, gdzieś obok obrazków, umieścić informację w rodzaju: „zrób z tego kafelki”, „zastosuj antyaliasing”, czyli wygładzanie, „zawsze rozciągaj, gdy jest miejsce” albo „przycinaj, gdy trzeba”. Jest to jakby wstępna obróbka surowej mapy bitowej za pomocą samych opisów, czyli jeszcze bez programowania w Javie.

Żeby zapoznać się tą technologią, sporządź bitmapowe tło ekranu. Przygotuj najpierw odpowiedni „kafelki” w wersjach *-ldpi*, *-mdpi* i *-hdpi* (tak naprawdę może to być dowolny, mały obrazek) i poszczególne wersje umieść w katalogach *drawable*.

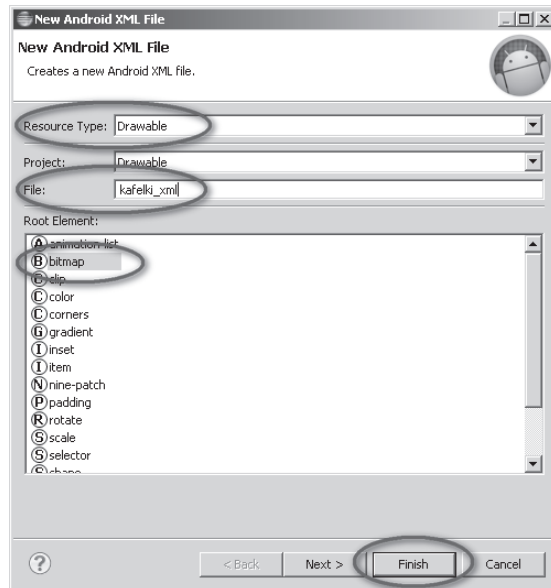
Potem przygotuj plik XML, który dodatkowo opisze naszą mapę bitową. Plik taki powinien znaleźć się w jednym, wspólnym dla wszystkich maszyn folderze *drawable*. Kreator tego pliku — opakowania na mapę bitową — automatycznie utworzy odpowiedni folder. Jak dodać zasób XML do folderów *drawable*? Kliknij prawym klawiszem którykolwiek folder *drawable*, wybierz opcję *New* i dalej *Android XML File* (rysunek 3.12).

W kreatorze pliku XML dokonaj kilku zaznaczeń (albo — gdy zostanie już utworzony — zmień w pliku kilka rzeczy ręcznie). Wpisz nazwę nowego zasobu, którym będzie teraz plik XML. Typem zasobu niech będzie *bitmap* (rysunek 3.13).

Rysunek 3.12.
Dodawanie zasobu XML
do folderów *drawable*



Rysunek 3.13.
Tworzenie nazwy
i wybór typu nowego
zasobu XML



Plik XML nie został wstawiony do znanej już Ci rodziny folderów *drawable-...*, ale do nowego, wspólnego folderu o nazwie po prostu *drawable*. Jak się pewnie domyślasz, folder *drawable* zawiera zasoby graficzne wspólne dla wszystkich maszyn, niezależnie od kategorii ich wyświetlaczy.

Nowy zasób będzie wspólny dla wszystkich maszyn, ale nie do końca. Okaze się (jak znów nietrudno się domyślić), że ostateczna wersja obrazka nadal będzie wyjmowana z odpowiedniego folderu. Wspólne jest opakowanie XML, ale prawdziwa fotografia nadal ma wiele wersji.

Przyjrzyj się więc plikowi XML opakowującemu mapkę bitową. Najpierw, po zakończeniu pracy kreatora, plik ten wygląda tak:

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

Do wnętrza tagu `<bitmap>` wpisz najważniejszy i bezwzględnie konieczny atrybut, którego wartością jest nazwa opakowywanej mapki bitowej, tutaj *kafelki*:

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/kafelki">
</bitmap>
```

Zatem zdefiniowałeś zasób nowego typu — jest to plik XML opakowujący prawdziwy zasób graficzny. Teraz go wykorzystasz jako tło aplikacji (rysunek 3.14). Niech główny plik wyglądu, o nazwie prawdopodobnie *activity_main.xml* (taką nazwę nadaje mu na samym początku kreator nowej aplikacji), ma następującą treść, napisaną „ręcznie” albo zbudowaną w edytorze wizualnym:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/kafelki_xml">
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:src="@drawable/motor1" />
</RelativeLayout>
```

Jest to w zasadzie to samo, co było poprzednio, zmieniono tylko sposób ułożenia obrazka na ekranie — teraz nie znajduje się on w lewym górnym rogu, ale na środku. I co najważniejsze, pojawiła się linia:

```
    android:background="@drawable/kafelki_xml">
```

Podobną frazę już kiedyś stosowałeś, nadając kolor podłożu aplikacji:

```
    android:background="#000080">
```

Nie byłoby sensu opakowywać w taki sposób plików graficznych, gdyby po drodze nie pojawiły się nowe możliwości. Uzupełnij plik XML o atrybut *gravity* (czyli o informację w którą stronę obrazek „ciąży”, gdy ma na ekranie miejsce):

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/kafelki"
    android:gravity="fill">
</bitmap>
```


Rysunek 3.14.

Mapa bitowa została opakowana w plik XML, który z kolei został użyty do budowy tła aplikacji



Atrybut `gravity` może przyjmować dość oczywiste wartości: `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill` (jest to wartość domyślna, ustawiana, gdy nie użyjesz atrybutu `gravity`), `clip_vertical`, `clip_horizontal`.

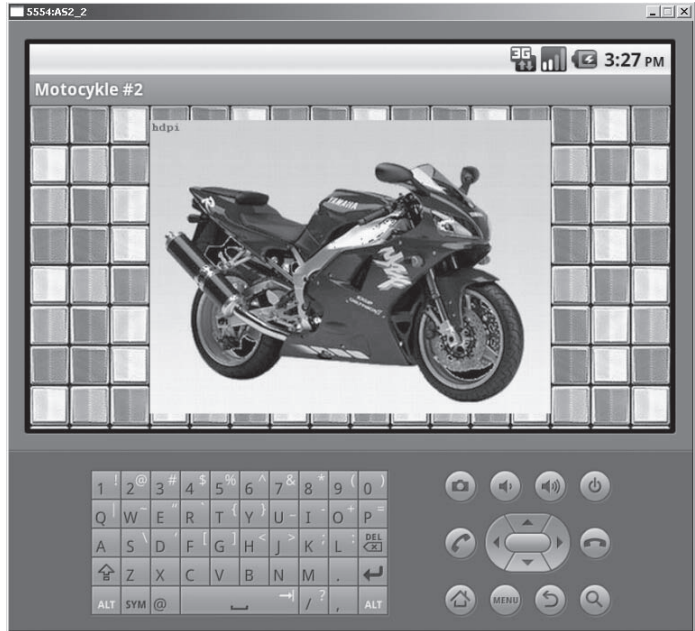
Jeszcze ciekawszym atrybutem jest `tileMode`. Atrybut ten nie może być użyty razem z poprzednim `gravity` — czyli stosujesz albo jeden, albo drugi:

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/kafelki"
    android:tileMode="repeat">
</bitmap>
```

Atrybut `tileMode` oprócz wartości `repeat` może przyjmować stany `clamp` (nie kafelkuj, tylko rozciągaj krawędzie) i `mirror` (kafelkuj, ale lustrzanie obracając co drugie rzędy i wiersze kafelków).

Oprócz atrybutów `gravity` (jak ułożyć obrazek na ekranie) oraz `tileMode` (jak zbudować kafelki z obrazka) masz do dyspozycji trzy atrybuty (rysunek 3.15) definiujące jakość obrazu i przyjmujące wartości `true` albo `false`: `android:antialias` (wygładzanie), `android:dither` (poprawianie palety kolorów) i `android:filter` (poprawianie obrazu przy ścisaniu albo rozciąganiu mapy bitowej).

Rysunek 3.15.
 Przy opakowywaniu pliku graficznego można zastosować różne atrybuty (*gravity*, *tileMode*, *android:antialias*, *android:dither*, *android:filter*)

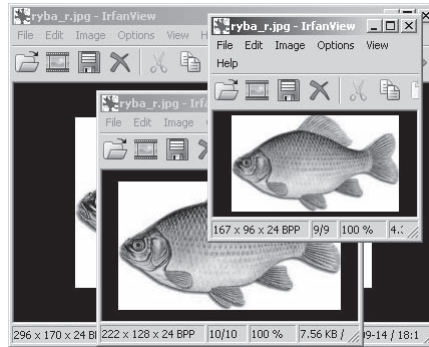


Wiele map bitowych w jednym opakowaniu XML

A jeśli chciałbyś mieć przycisk, na którym znajduje się zmieniająca się mapa bitowa: inna, gdy myszka wisi nad przyciskiem, inna, gdy go klika, inna, gdy go opuszcza? Zagadnienie to da się zrealizować bez programowania, za pomocą opisów w odpowiednich plikach XML. Gdybyś chciał poczytać o tym więcej, wpisz w Google frazy „Android State List” (lista stanów).

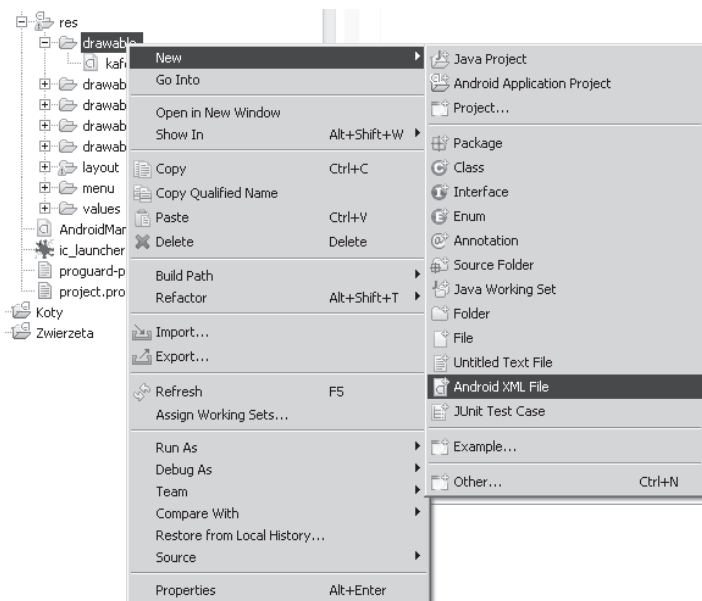
Zacznij od przygotowania prawdziwych zasobów, czyli teraz serii mapek bitowych, przedstawiających np. rybkę w trzech różnych kolorach. Potem przygotuj tę serię rybek w różnych rozmiarach i, być może, różnych jakościach, przeznaczonych na urządzenia o różnej jakości wyświetlaczy (rysunek 3.16). Te pliki — tak jak poprzednio — umieść w folderach *drawable-ldpi*, *drawable-mdpi* itd. W zależności od stanu przycisku (zaraz odpowiedni przycisk zostanie wprowadzony do gry) system Android umieści na nim inną wersję mapki bitowej. Powinieneś też przygotować pomniejszone i powiększone wersje obrazków i porozmieszczać je w folderach *drawable-ldpi*, *drawable-mdpi* i *drawable-hdpi*.

Rysunek 3.16.
Przygotowanie obrazka
o boku długości kilkuset
pikseli i różnej
kolorystyce albo
różnicy dotyczącej
jakiegoś innego
czytelnego szczegółu



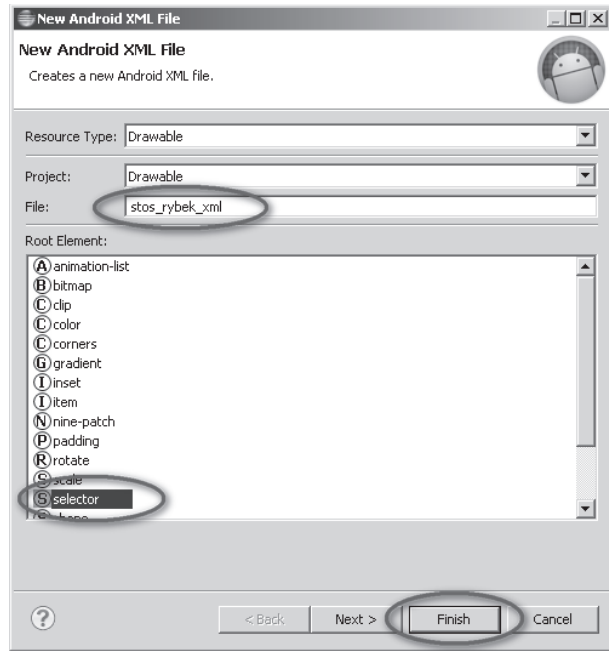
Mając w folderach *drawable-...* serię rybek o różnych kolorach, możesz przystąpić do opisywania *Listy stanów*. Kliknij prawym przyciskiem myszki folder *drawable* i z pojawiającego się menu wybierz *New* i dalej *Android XML File*, czyli plik XML (rysunek 3.17). Przed chwilą w takim pliku opisywałeś jedną mapkę bitową, teraz opiszesz cały ich stos.

Rysunek 3.17.
Nowy plik XML,
opisujący cały stos
mapek bitowych



Wykorzystaj pojawiający się kreator pliku XML, choć mógłbyś przygotować odpowiedni plik całkowicie ręcznie. Podaj nazwę pliku, oznacz, że główny element ma być typu *Selector* i zakończ przyciskiem *Finish* (rysunek 3.18). W folderze *drawable* powinien pojawić się nowy zasób — plik *stos_rybek_xml*.

Rysunek 3.18.
*Tworzenie nowego
 zasobu XML — plik
 stos_rybek_xml*



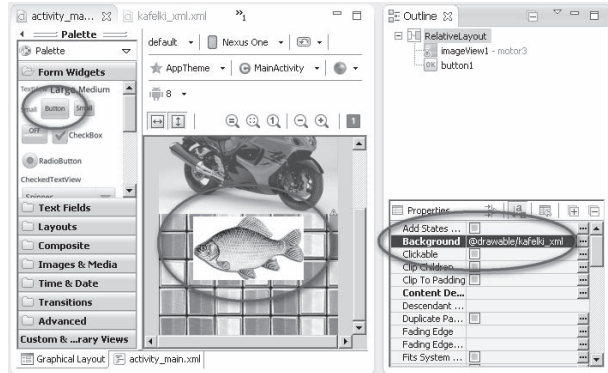
Jak zwykle dwa razy kliknij nowy plik, aby otworzyć go do edycji. Uzupełnij jego treść następująco:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:state_pressed="true"
    android:drawable="@drawable/ryba_r" />
  <item android:state_focused="true"
    android:drawable="@drawable/ryba_b" />
  <item android:drawable="@drawable/ryba" />
</selector>
```

Do wnętrza tagu `<selector>` wpisz kilka tagów `<item>`, odpowiadających różnym stanom przycisku, który za chwilę zaimplementujesz. Gdy przycisk jest w stanie `pressed`, czyli naciśnięty, powinna się wyświetlać mapka o nazwie `ryba_r` (rybka o czerwonym odcieniu). Gdy przycisk jest w stanie `focused`, czyli gotowy do naciśnięcia — `ryba_b`. Gdy z przyciskiem nic się nie dzieje albo coś się dzieje, ale tego tutaj nie opisałem, wyświetli się wersja zapisana jako `ryba`. Jest to tzw. normalny stan przycisku, który opisuje się zawsze za pomocą ostatniego tagu `<item>` w powyższym wyliczeniu stanów.

Należy jeszcze zaimplementować sam przycisk oraz podpiąć do niego ten oryginalny sposób kolorowania jego powierzchni. Główny plik wyglądu z folderu *layout* powinien mieć następującą zawartość:

Rysunek 3.19.
Do poprzedniej aplikacji należy dodać przycisk i opracować jego właściwość `background` (tło). Niech tłem będzie nowy zasób XML zwany `Listą stanów`



```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/kafelki_xml" >
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="50dp"
        android:src="@drawable/motor3" />
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/imageView1"
        android:layout_centerHorizontal="true"
        android:background="@drawable/stos_rybek_xml"/>
</RelativeLayout>
```

Duża część tego pliku jest Ci znana. Jest tu rozkład relatywny, czyli kolejne komponenty są układane względem poprzednich, ułożonych wcześniej. Rzeczywiście, komponent `Button` leży *pod* komponentem `ImageView`:

```
android:layout_below="@+id/imageView1"
```

Natomiast stos obrazków — zagadnienie opisane na kilku ostatnich stronach — pojawia się w linii opisującej tło przycisku:

```
android:background="@drawable/stos_rybek_xml"/>
```

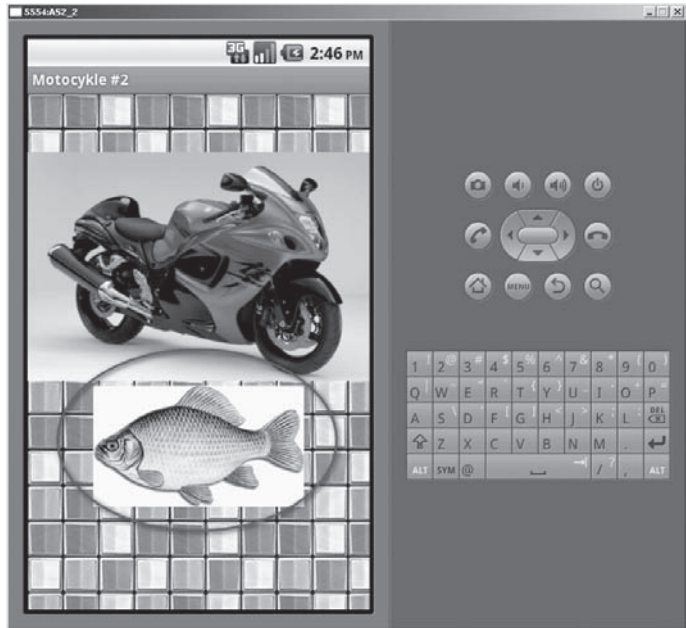
Jest to odwołanie do właśnie stworzonego zasobu XML, umieszczonego w folderze `drawable`.

Pora na podsumowanie. Znowu powtórzył się następujący schemat: gdzieś w pierwotnych folderach *drawable*-... znajdują się prawdziwe mapy bitowe, różniące się rozmiarami, może także jakością grafiki. Różnice między nimi konieczne są z tego względu, że na lepsze urządzenia przygotowuje się inną grafikę (grafika, która ma więcej detali i szczegółów) niż na gorsze.

Jakby nad surowymi mapami bitowymi znajdują się pliki XML definiujące specjalne właściwości rysunku. Pliki te znajdują się we wspólnym dla wszystkich urządzeń katalogu *drawable*. Dopiero te pliki wywoływane są w głównym pliku wyglądu. Tak pojawiły się wcześniej kafelki, a teraz aktywny przycisk zmieniający swój rysunek, gdy klikniesz go myszką (rysunek 3.20).

Rysunek 3.20.

Ten przycisk jest zasób XML, zwany State List (lista stanów). Każdy stan to jakiś inny obrazek — kliknięcie czy zatrzymanie się na obrazku zmienia jego wygląd



Skorowidz

A

AbsoluteLayout, 32, 34
ADT, Android Development Tools, 12
akcja, *Patrz* metoda
aktywności ściągnające dane, 206
aktywność
 poboczna, 208
 wtórna, 207
algorytm
 klasy publicznej, 81
 rysowania, 135
Android SDK, 9, 211
Android SDK Manager, 13, 211
animacja, 98
 łączenie efektów, 110
 poklatkowa, 113
 przycisku, 108
antialiasing, 130, 135
aparat graficzny, 188
aplikacja z ekranami, 191
atrybut
 angle, 68
 contentDescription, 46
 gravity, 50
 tileMode, 51
atrybuty XML, 48
AVD, Android Virtual Device, 14, 41, 211

B

bitmapy prywatne, 161
blokowanie
 komponentu, 163
 przycisku, 165, 171

błąd, 188
błąd w logice interfejsu, 164
budowanie interfejsu, 169

C

czas trwania animacji, 102
czyszczenie ekranu, 162

D

Dalvik, 9, 211
definiowanie
 efektów animacyjnych, 101
 egzemplarza aparatu graficznego, 188
 klasy w klasie, 176
 przycisku, 79, 84
 stylów, 87
 tła, 66
długotrwały proces, 168
dodawanie
 identyfikatora, 148
 zasobu XML, 49
dokumentacja online Androida, 119
dostęp
 do internetu, 208
 do zmiennych, 119
dp, device independent pixel, 64, 211
drzewo plików projektu, 98
dynamiczne tworzenie obrazów, 161
działanie aktywności, 147
dziedziczenie, 125

E

Eclipse, 9
Eclipse Plugin, 211
edytor Eclipse, 9
edytowanie pliku, 30, 67, 201
efekt, 97
 rozmycia obrazu, 157
 zamiany składowej koloru, 155
 zanikania, 102
efekty
 animacyjne, 98
 specjalne, 100
egzemplarz procesu, 176
ekran, 191
 HVGA, 41
 QVGA, 41
 WVGA, 41
element
 resource, 92
 shape, 89
elementy interfejsu, 25
emulator
 aplikacji, 18
 o ekranach różnej jakości, 41

F

fabryka map bitowych, 150, 187
film, 113, 116
folder, *Patrz* katalog
format fotograficzny, 188
funkcja, *Patrz* metoda
funkcje prywatne, 109

G

głębia koloru, 151
gra w kółko i krzyżyk, 87
graficzne zasoby aplikacji, 39

I

IDE, 11, 212
identyfikator, 148
implementacja klasy, 82
importowanie pakietu, 81, 130
informacja o rozmiarach komponentu, 150
inicjowanie
 bitmapy, 161
 zmiennej, 120
instalator Android SDK, 10
instalowanie
 Android SDK, 9
 IDE Eclipse, 9, 12
 Javy, 9
interfejs
 do oglądania filmu, 116
 użytkownika, 103, 146

J

Java, 9
JDK, 212
jednostka dp, 64

K

katalog
 bin, 17
 drawable, 39, 49
 layout, 22, 63
 res, 17–21, 39
 src, 17
 values, 20, 62
klasa
 Activity, 106, 147, 171
 Animation, 106
 AnimationDrawable, 118
 AsyncTask, 167, 173
 BitmapFactory, 150
 Bundle, 208
 Calendar, 85
 Canvas, 127, 152
 Figura, 133
 ImageView, 161
 Intent, 200

MainActivity, 81
OdczytObrazka, 186, 205
Paint, 127
Proces, 172
RectF, 136
View, 106, 122, 134

klasy

 bazowe, 127, 133
 implementacja,
 implements, 82
 konstruktor, 124
 kreator, 122
 nazwa, 124, 198
 rozszerzanie, extends, 82,
 132, 171, 174
 tworzenie, 122
klatki filmu, 113
klawisze Ctrl+Spacja, 68, 92,
 99, 198
kolejność
 budowania interfejsu, 169
 rozmieszczania elementów,
 192

kolor, 61, 65

kolor piksela, 156

komponent

 Button, 59, 145
 ImageView, 35, 103, 145,
 194
 ProgressBar, 177
 RelativeLayout, 91
 TableRow, 63
 TextView, 71, 145
 View, 122

komponenty

 blokowanie, 163
 graficzne, 121
 informacja o rozmiarach, 150
 nazywanie, 71
 położenie, 137, *Patrz także*
 rozkład
 tworzenie, 121
 wizualne, 121
 własne, 121
 wybieranie, 31
konfigurowanie Eclipse, 11–13
konstruktor klasy, 124
 bazowej, 125
 potomnej, 125
kółka, 129
kółko i krzyżyk, 87
kreator
 klasy, 122
 nowego projektu, 78, 194

 nowej aplikacji, 50
 zasobu XML, 48, 100, 115
kształty, 66

L

layout, 22, 58
 AbsoluteLayout, 32, 34
 LinearLayout, 24, 26
 RelativeLayout, 70, 90,
 158
 TableLayout, 27, 58
LinearLayout, 24, 26
linia, 68
lista stanów, 53
lokalizacja
 oprogramowania, 12
 pliku graficznego, 37
losowanie
 jednokrotne, 142
 liczb, 131
 pikseli, 157

Ł

łączenie
 animacji, 110
 klatek, 115

M

manifest, 200
mapy bitowe, 43, 48, 145
inicjalizowanie, 161
operacje graficzne, 154
podgląd, 145
przekształcenie, 154
rysowanie, 152
maszyna
 losująca Random, 131
 wirtualna Dalvik, 9
metoda
 data_i_godzina(), 84
 decodeResource(), 150
 doInBackground(), 180
 drawLine(), 162
 drawRect(), 163
 eraseColor(), 163
 execute(), 185
 findViewById(), 86, 106,
 148, 161
 finish(), 81
 getContentDescription(), 141

getInstance(), 85
 getPixel(), 156
 getString(), 208
 kliknieto(), 79, 95
 kliknieto_kreuj(), 161
 kliknieto_start(), 169
 kliknij(), 148
 losuj_dane(), 143
 obracaj(), 106
 onCreate(), 120, 171, 199
 onDraw(), 126, 130, 134
 onPostExecute(), 188
 onPreExecute(), 176, 188
 onProgressUpdate(), 179
 putExtra(), 207
 rgb(), 156
 rysuj(), 152, 155
 setARGB(), 162
 setText(), 86
 start(), 118
 startAnimation(), 107
 stop(), 118
 stosuj_wszystko(), 112
 super(), 125
 metody
 abstrakcyjne, 198
 finalne, 132
 nadpisane, 127
 prywatne, 109

N
 nadawanie
 nazwy plikowi .xml, 92
 wartości wspólnej, 64
 nadpisywanie metody, 125
 narzędzia Android SDK, 10
 nawigacja między ekranami, 202
 nazwa
 klasy, 124, 198
 metody, 124, 198
 pliku, 124
 nazywanie
 komponentów, 71
 pliku XML, 99

O
 obiekty ułożone
 obok siebie, 24
 w oczkach sieci, 27
 obraz klasy fotograficznej, 188

obrazek, 145
 obrót, 101, 109
 obrót ekranu, 96
 obsługa
 kliknięć, 105, 147, 192
 wyjątków, 188
 zdarzenia onClick, 95
 oczyszczanie pamięci, 171
 odczytywanie
 obrazka, 186
 pliku, 185
 odśmiecacz Javy, 171
 odświeżanie obrazka, 171
 okno
 emulatora aplikacji, 18
 kreatora klasy, 123
 okrągły wskaźnik postępu, 194
 określanie wartości atrybutu, 140
 opisywanie
 efektów specjalnych, 100
 zasobów, 114
 oprogramowanie ekranów, 202
 osadzanie elementów, 83

P

pakiet
 Eclipse Classic, 11
 Javy, 10
 piksel, 155
 plik
 activity_main.xml, 58
 AndroidManifest.xml, 183, 200
 Figura.java, 133
 MainActivity.java, 79, 117, 153, 206
 strings.xml, 20, 25, 34
 pliki
 class, 17
 wyglądu, 28, 79, 91, 104, 117, 138, 146, 159, 182, 196
 podgląd mapy bitowej, 145
 podpowiedź, 68, 92, 99, 198
 pole tekstowe, 69, 86
 polecenie Refresh, 114, 145
 polimorfizm, 96, 148
 położenie komponentu, 137
 postfixs
 hdpi, 40, 43
 ldpi, 39
 mdpi, 40
 xhdpi, 40, 43

pozyskiwanie mapy bitowej, 154
 proces, 167
 proces ściągania, 182
 projekcja filmu, 118
 prostokąt, 68
 prototyp metody, 148, 174
 przechwytywanie parametru, 208
 przekształcanie mapy bitowej, 154
 przesunięcie, 102, 110
 przesuwanie ekranu, 58–61
 przezroczystość, 101
 przycisk, 64
 „Koniec”, 77
 Button, 32, 194
 Obrót, 107
 przyciski
 animowanie, 98
 blokowanie, 165, 171
 definiowanie, 79, 84
 rozmieszczanie, 93
 wstawianie, 59
 przypisywanie kolorów, 65

R

RelativeLayout, 70, 90, 158
 rozkład
 AbsoluteLayout, 32, 34
 LinearLayout, 24, 26
 RelativeLayout, 70, 90, 158
 TableLayout, 27, 58
 rozłożenie swobodne, 32
 rozmiar
 komponentu, 149
 obrazka, 43
 rozmieszczanie
 przycisków, 93
 elementów, 192
 rozmycie obrazu, 157
 rozszerzanie
 funkcjonalności, 82
 klasy, 82, 132, 171, 174
 rysowanie
 na ekranie, 134
 na mapie bitowej, 152, 153
 ramki, 188
 w drugim planie, 167
 w tle, 178
 rzutowanie, 95

S

ScrollView, 57
 SDK Location, 12
 Shapes, 66
 silnik animacji, 119
 skalowanie, 102
 składowe koloru, 155
 słowo kluczowe

- extends, 82
- final, 132
- implements, 82
- super, 125, 127

 stan urządzenia, 180
 struktura katalogów drawable, 39
 strumień danych, 187
 sufler, 68, 92, 99, 198
 suwak przewijania, 60, 137

Ś

ściąganie danych z internetu, 181
 środowisko programistyczne, 9

T

TableLayout, 27, 58
 tag

- animation-list, 115
- bitmap, 113
- LinearLayout, 136
- ScrollView, 58, 136

 tło

- aplikacji, 66, 89
- gradientowe, 90

 TrueColor, 162
 tworzenie

- AVD, 15
- efektów, 97
- ekranów, 191
- ikony, 17, 36

klasy, 122
 komponentów, 121
 mapy bitowej, 157
 nazwy, 49
 pierwszej aplikacji, 15, 19
 pliku, 62, 195
 pliku XML, 99
 wirtualnego urządzenia, 41
 zasobu XML, 54
 typ

- Void, 180
- nieokreślony, 173

 typy wyświetlaczy, 41

U

układanie obiektów, 24, 27
 uruchamianie

- animacji, 105
- aplikacji, 47
- procesu w tle, 176, 178
- projekcji, 118

 urządzenia AVD, 46
 urządzenie

- QVGA, 42
- WVGA800, 42

 ustalanie atrybutów figur, 130
 ustawienia kursora, 67

W

wątek

- główny, 167
- poboczny, 167

 wersja systemu Android, 14
 wirujące kółko, 182
 wklejanie adresu internetowego, 185
 właściwość

- Background, 65, 66
- contentDescription, 138
- Enabled, 163
- src, 103

Style, 92
 Visibility, 176
 Workspace, 19, 212
 wskaźnik postępu, 177, 194
 wstawianie

- fraz XML, 67
- przycisku, 59
- wierszy, 63

 wybór

- komponentu, 31
- nazwy rysunku, 36
- zasobu, 45

 wygląd

- aplikacji, 19, 22, 57
- ekranu głównego, 194
- tabelaryczny, 58

 wyglądanie grafiki, 130, 135
 wyjątek, 80, 188
 wyświetlacze, 41
 wyświetlanie

- daty, 82
- grafiki, 194, 203
- obrazka, 161
- zmian, 153

Z

zakładka Images & Media, 45
 zamiana składowej koloru, 155
 zanikanie, 102
 zarządca systemów Android, 13
 zasoby

- graficzne, 39, 113, 145
- XML, 55

 zdarzenie onClick, 95, 147
 zegar, 82
 zezwolenie na dostęp do internetu, 208
 zmienne

- prywatne, 166
- typu Bitmap, 150

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>

ANDROID

Podstawy tworzenia aplikacji

Według najnowszych badań Android jest najpopularniejszym systemem operacyjnym dla urządzeń mobilnych. Środowisko to kontroluje już niemal 2/3 wszystkich używanych na świecie smartfonów i tabletów, a jego użytkownicy mają do dyspozycji ponad 450 tysięcy różnych aplikacji! Nic więc dziwnego, że w sieci pełno jest ofert zatrudnienia dla osób, które potrafią tworzyć programy dla tego systemu. Najlepsi wprost nie mogą opędzić się od rekruterów proponujących im coraz ciekawsze warunki finansowe...

Droga do sukcesu jest jednak długa i żmudna. Nauka programowania na potrzeby Androida nie wszystkim wydaje się rzeczą prostą, wielu zniechęca się już na wstępie. Najczęściej powodem jest niezajomość platformy, podstaw programowania obiektowego, języka Java czy choćby stosowanych tu narzędzi. Już samo korzystanie z przykładów zamieszczonych w sieci oraz ogólnodostępnej literatury wymaga zwykle minimalnej wiedzy na temat technologii i generalnego obycia w świecie urządzeń mobilnych.

Na szczęście jest już dostępna książka *Android. Podstawy tworzenia aplikacji*, która bezboleśnie wprowadzi Cię w krainę programowania komputerów mobilnych pracujących pod kontrolą systemu firmy Google. Dzięki niej poznasz podstawy języka Java i budowę aplikacji działających w Androidzie. Nauczysz się posługiwać odpowiednimi narzędziami i pisać własne programy. Dowiesz się, jak korzystać z zasobów graficznych i funkcji systemowych. Ponadto znajdziesz tu instrukcje, jak pobierać dane z sieci oraz uruchamiać nowe wątki.

- Podstawy programowania urządzeń mobilnych
- Wyposażenie warsztatu pracy programisty
- Tworzenie prostych aplikacji dla Androida
- Definiowanie układów obiektów na ekranie
- Korzystanie z zasobów graficznych aplikacji
- Uruchamianie i kontrolowanie wątków
- Tworzenie aplikacji wieloekranowych

Chwyć robota za antenki i zacznij samodzielnie tworzyć aplikacje dla Androida!

helion.pl
księgarnia
internetowa

Nr katalogowy: 13349



Księgarnia internetowa:

<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

• <http://helion.pl/nowosci>

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

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-246-7006-2



9 788324 670062

Cena: 39,90 zł

Informatyka w najlepszym wydaniu