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

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

Printed in Poland.

Spis treści

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 *Rozdzial_3*.

Struktura katalogów drawable

Już wiesz, że wyglądu aplikacji w Androidzie raczej się nie programuje (chociaż 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 wyświetlania na różnej klasy urządzeniach. Jeszcze niedawno były trzy foldery *drawable*, teraz jest ich więcej, bo pojawiają się coraz lepsze urządzenia mobilne. Postfiks *-ldpi* należy rozwinąć jako *low dot per inch*, co w tym wypadku oznacza „urządzenie o małej

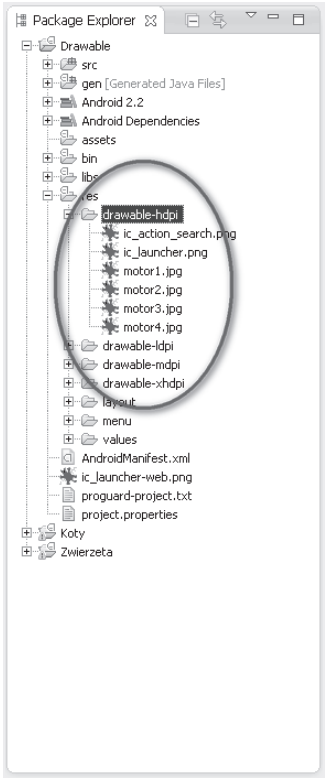
gęstości pikseli". W folderze *drawable-ldpi* umieścisz wersje grafik dla kiepskich ekraników. Grafiki te powinny być mniejsze, ale także mogą zawierać mniej detali.

Katalogi z postfiksami *-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*

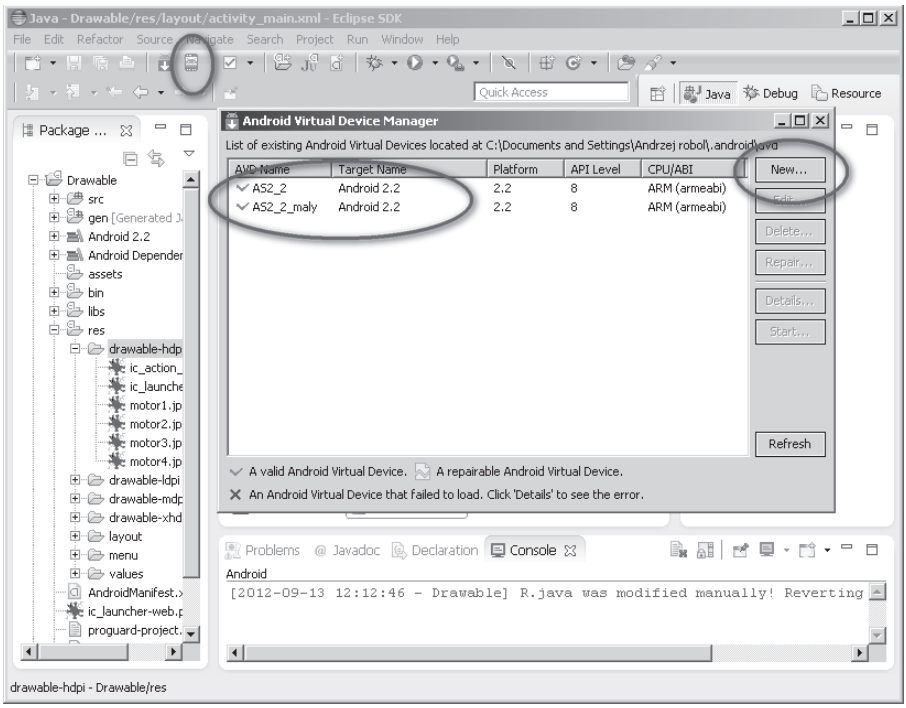


Nie ma dokładnych wytycznych, jak sporządzać i dzielić zasoby na poszczególne foldery *drawable*. Przyjmuje się, że jeśli dla urządzenia typowego, czyli średniego *-mldpi*, przygotuje się mapę bitową o boku np. 400 pikseli, to dla urządzenia małego *-ldpi* trzeba będzie ją zmniejszyć do 75 procent, dla 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.

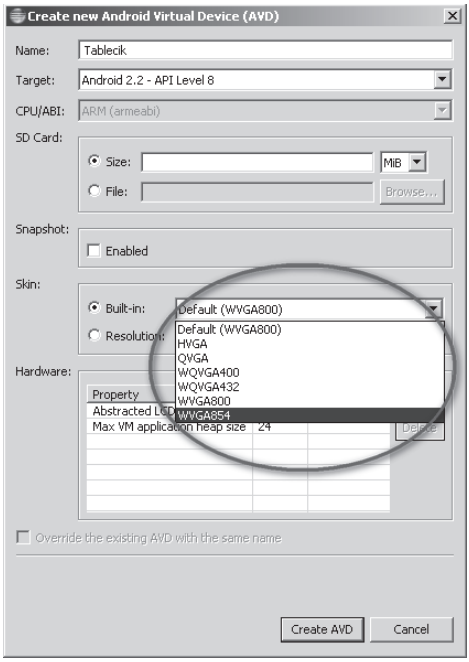
Emulatory o ekranach różnej jakości

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 ikonkę 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*

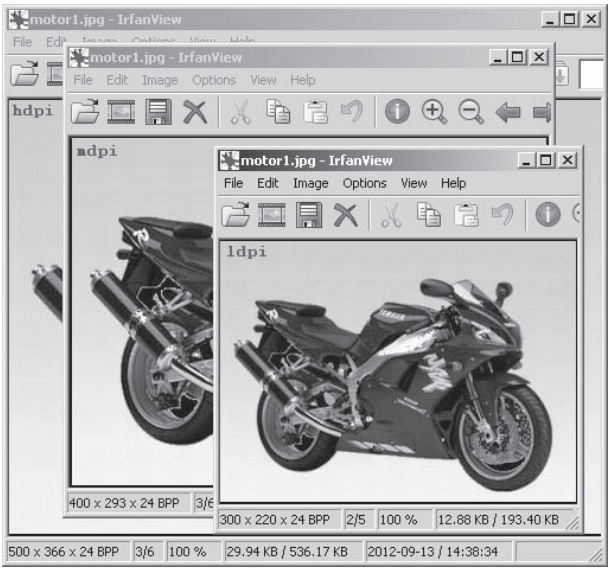
Rysunek 3.4. *Urządzenie QVGA na tle urządzenia WVGA800*

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 obsłuży takiego standardu.

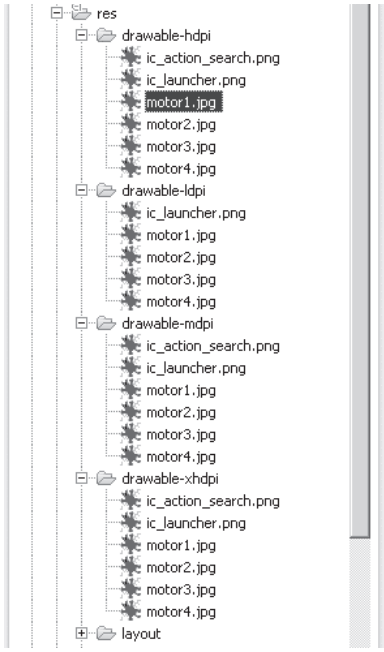
Wersje obrazka należy jakoś oznaczyć, np. nadrukowując na nie w programie graficznym napisy *ldpi*, *mdpi*, *hdpi* (rysunek 3.5). Dzięki temu będziesz mógł śledzić, jak konkretne urządzenie wybiera sobie grafikę do wyświetlenia.



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 zmieniałeś 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).

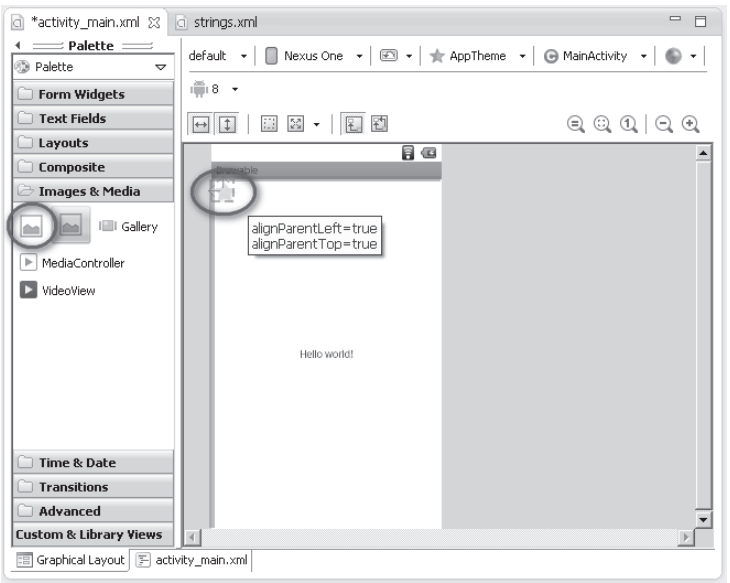
Jeśli już zaczyna Cię denerwować napis „Hello World", reprezentowany przez komponent `TextView`, kliknij go prawym klawiszem myszki i usuń z aplikacji.

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).

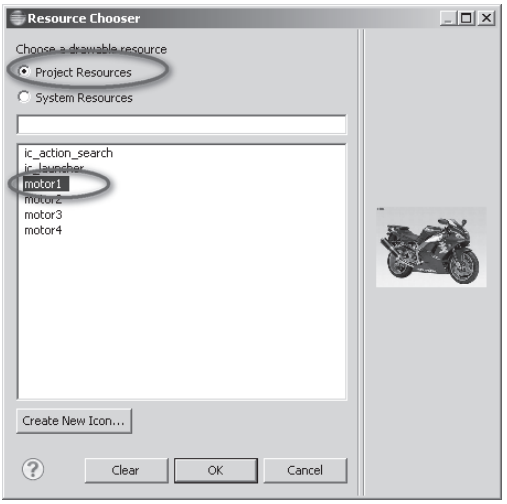
Po umieszczeniu komponentu w polu ekranu prawdopodobnie automatycznie pojawi się *Resource Chooser* (wybieranie zasobu). Jeśli zaznaczono *Project Resources* (a nie *System Resources* — przebogate zasoby własne Androida), okienko *Choosera* zajrzy do Twoich katalogów *drawable* i zaproponuje wybór któregoś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

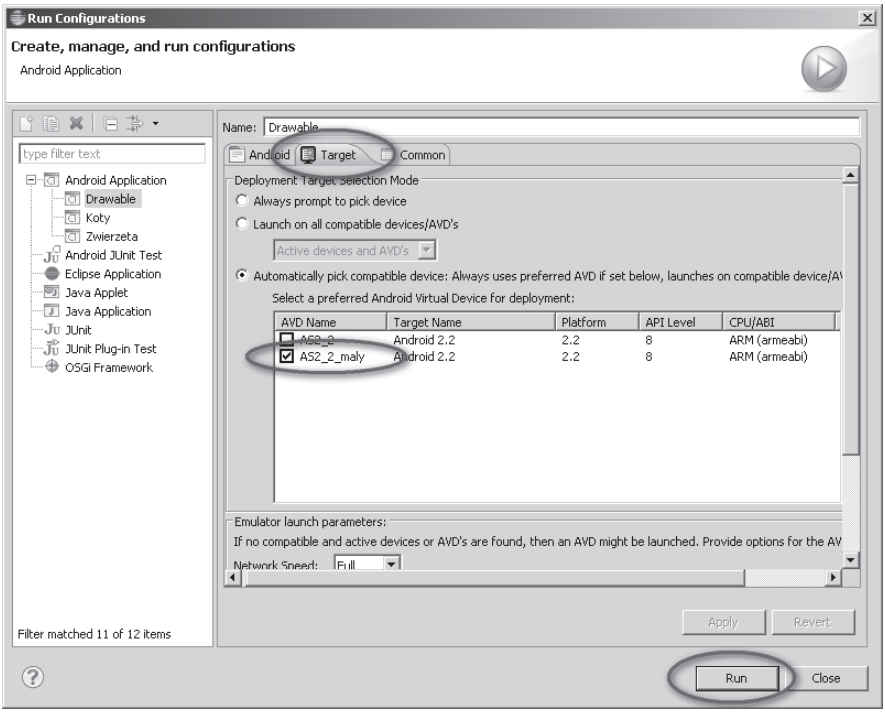
Najważniejszą linią jest połączenie komponentu o nazwie `imageView1` z mapą bitową *motor1.jpg* znajdującą się w folderach *drawable*:

```
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*!

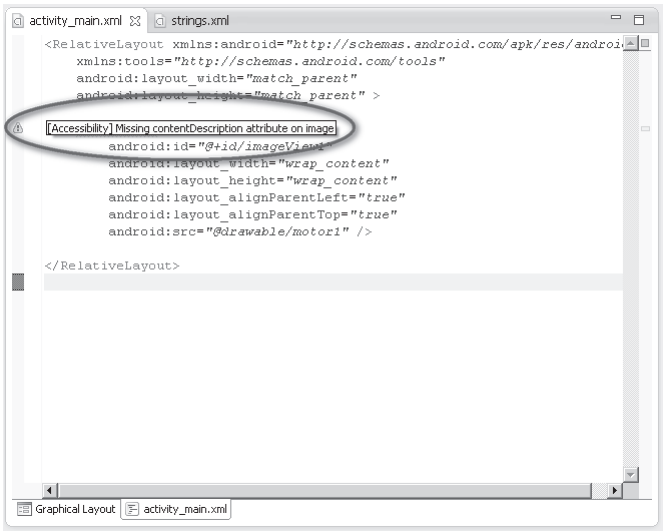
Być może zauważyłeś, że przy komponencie obrazka albo przy jego opisie w pliku XML znajduje się malutki żółty trójkącik uwagi, którą chce Ci podpowiedzieć kompilator: „*Missing contentDescription attribute.on image*" (zapomniałeś o parametrze *contentDescription*) — rysunek 3.11. Każdy element wizualny 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

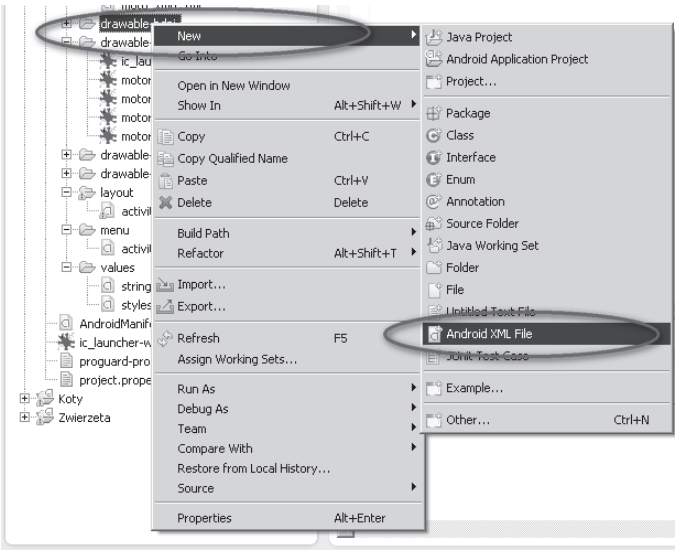
To, co zrobiłeś do tej pory z mapą bitową, można nazwać wyświetleniem surowego obrazka. Było to dobre i zazwyczaj jest wystarczające. Jednak można pójść o krok dalej, mianowicie: w folderach *drawable*, gdzieś obok obrazków, umieścić informację w rodzaju: „zrób z tego kafelki", „zastosuj antyaliasing", 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 „kafelek" w wersjach *-ldpi*, *-mdpi* i *-hdpi* (tak naprawdę może to być dowolny, mały obrazek) i poszczególne wersje umieść w katalogach *drawable*.

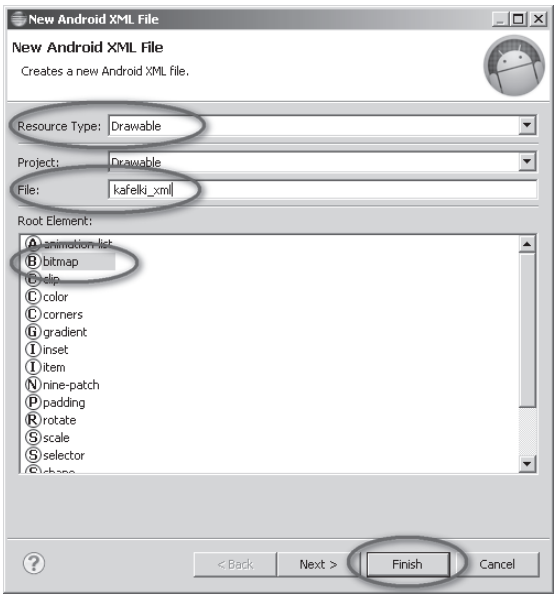
Potem przygotuj plik XML, który dodatkowo opisze naszą mapę bitową. Plik taki powinien znaleźć się w jednym, wspólnym dla wszystkich maszyn folderze *drawable*. Kreator tego pliku — opakowania na mapę bitową — automatycznie utworzy odpowiedni folder. Jak dodać zasób XML do folderów *drawable*? Kliknij 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. Okaże 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*

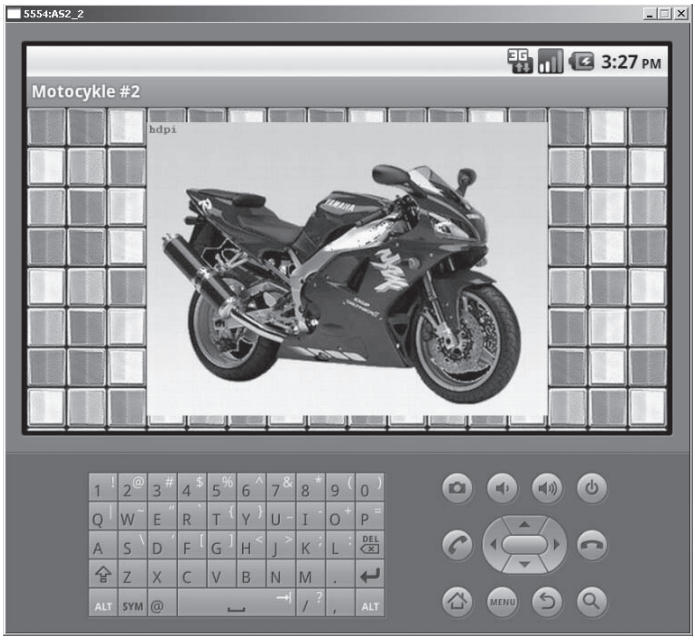
Atrybut `gravity` może przyjmować dość oczywiste wartości: `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill` (jest to wartość domyślna, ustawiana, gdy nie użyjesz atrybutu `gravity`), `clip_vertical`, `clip_horizontal`.

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 ściskaniu albo rozciąganiu mapy bitowej).

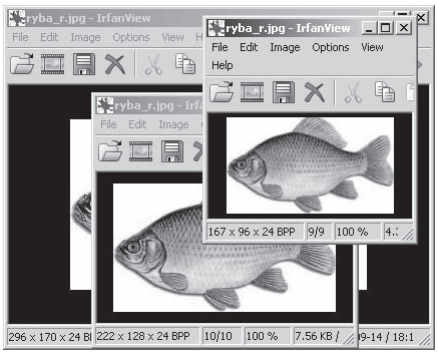


Rysunek 3.15. *Przy opakowywaniu pliku graficznegomożna zastosować różne atrybuty (gravity, tileMode, android:antialias, android:dither, android:filter)*

Wiele map bitowych w jednym opakowaniu XML

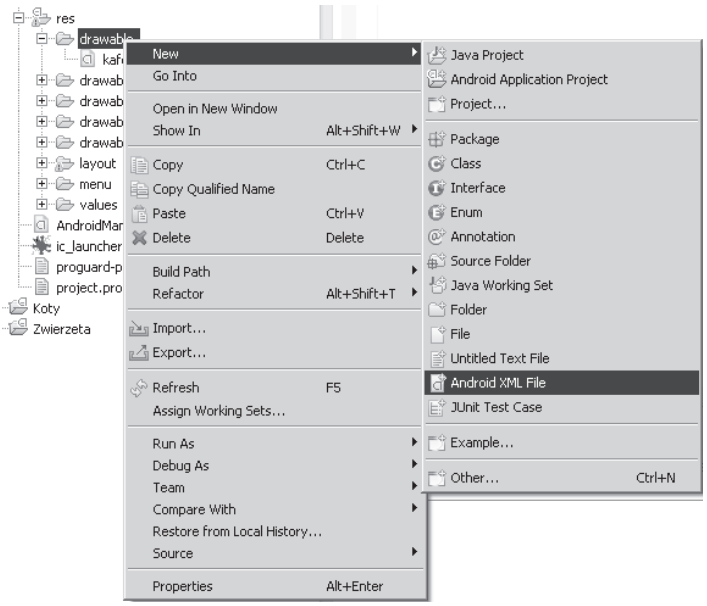
A jeśli chciałbyś mieć przycisk, na którym znajduje się zmieniająca się mapa bitowa: inna, gdy myszka wisi nad przyciskiem, inna, gdy go klika, inna, gdy go opuszcza? Zagadnienie to da się zrealizować bez programowania, za pomocą opisów w odpowiednich plikach XML. Gdybyś chciał poczytać o tym więcej, 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*.



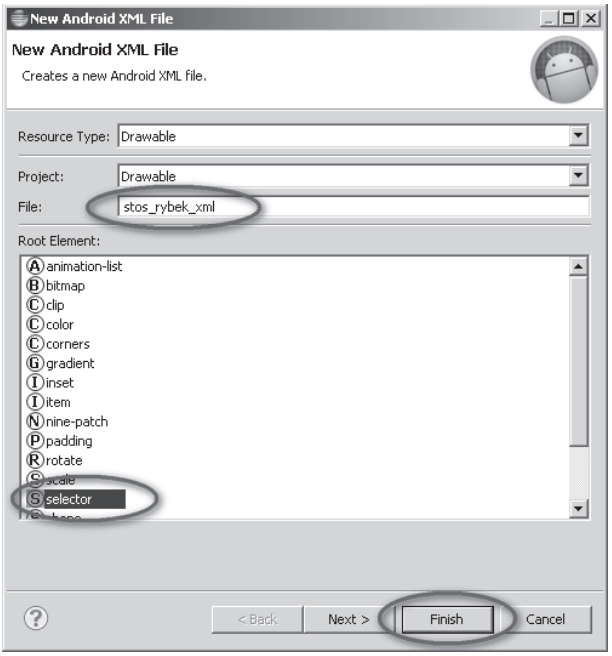
Rysunek 3.16. *Przygotowanie obrazka o boku długości kilkuset pikseli i różnej kolorystyce albo różnicy dotyczącej jakiegoś innego czytelnego szczegółu*

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*.



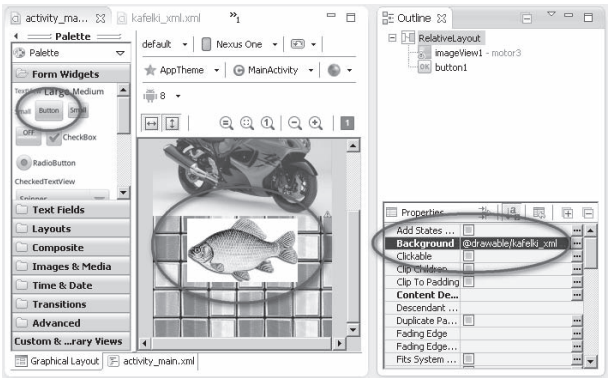
Rysunek 3.18. *Tworzenie nowego zasobu XML — plik stos_rybek_xml*

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 czerwonawym 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"
```

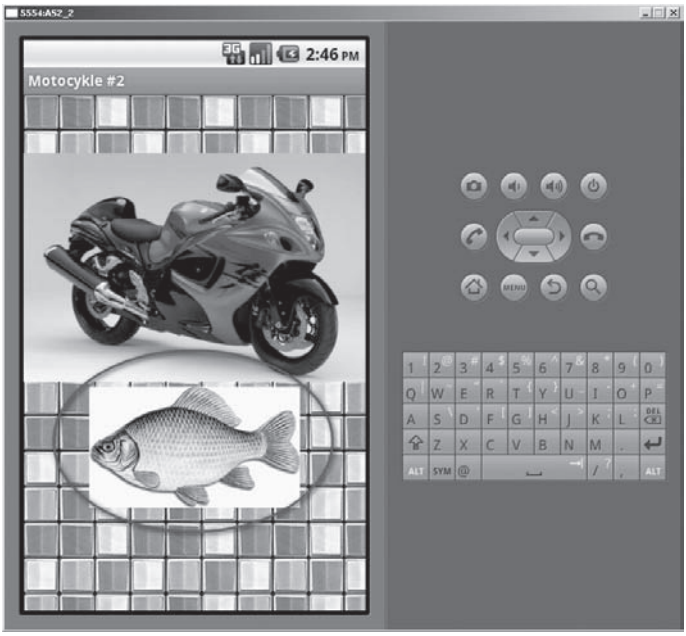
Natomiast stos obrazków — zagadnienie opisane na kilku ostatnich stronach — pojawia się w linii opisującej tło przycisku:

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

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

Pora na podsumowanie. Znów 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. *Tłem przycisku jest zasób XML, zwany State List (lista stanów). Każdy stan to jakiś inny obrazek — kliknięcie czy zatrzymanie się na obrazku zmienia jego wygląd*

Skorowidz

S

Ś

T

U

W

Z

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 nieznajomość 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

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

ISBN 978-83-246-7006-2

Cena: 39,90 zł

Informatyka w najlepszym wydaniu